Шнейвайс А.Б.

ОСНОВЫ ПРОГРАММИРОВАНИЯ

(ФОРТРАН, СИ)
Третий семестр

для студентов астрономического отделения
математико-механического факультета СПбГУ

2018

Шнейвайс А.Б.

Учебное пособие «Основы программирования (ФОРТРАН, СИ) (третий семестр) содержит информацию по дисциплине «Программирование», излагаемую студентам второго курса астрономического отделения математико-механического факультета СПбГУ. Основное внимание уделяется не только правильности использования синтаксических языковых конструкций, структурности записи исходных текстов и модульному подходу при разработке проектов программ, но и выработке навыков оценки программируемых формул (с точки зрения их пригодности для ведения расчётов на ЭВМ), а также умения приводить их к виду удобному для расчёта посредством аналитических приёмов. Пособие содержит множество примеров исходных текстов программ (с подробными пояснениями), что позволит студенту за относительно короткий срок освоить излагаемый материалю. Почти каждая тема, излагаемая в пособии, завершается кратким перечнем её основных моментов и списком задач соответствующего домашнего задания, которые необходимо выполнить в среде операционной системы LINUX. Пособие завершается приложением, содержащим набор зачётных задач. Результаты программ, приведённых в пособии, получены в основном на компиляторах gfortran и gcc.

Рекомендовано учебно-методической комиссией
математико-механического факультета СПбГУ
для студентов, обучающихся по специальности
«АСТРОНОМИЯ».

Содержание

1 Аргумент одной функции – имя другой.

1.1 Уяснение ситуации.

До сих пор в качестве аргументов процедур мы использовали простые переменные или массивы. Часто встречается ситуация, когда в качестве аргумента процедуре выгодно передать имя другой процедуры. Пусть, например, для расчета значения интеграла по формуле трапеций разработана ФОРТРАН-функция **trap0(a,b,n)**:

```
function trap0(a,b,n)        ! Функция реализует алгоритм расчета
implicit none                ! интеграла по промежутку [a, b] по
integer i, n                 ! формуле трапеций.
real a, b, trap0, f, x, s, h ! n - количество подучастков равномерного
h=(b-a)/n                    !     дробления промежутка [a,b]
s=(f(a)+f(b))/2;             ! Результат работы возвращается в
do i=2,n                     ! вызывающую программу через имя trap0
  x=a+(i-1)*h                ! (потому trap0 и функция).
  s=s+f(x)                   ! Для расчета значений подинтегральной
enddo                        ! функции используется функция с именем
trap0=s*h                    ! f и одним аргументом x.
end

function f(x)                ! В качестве примера подинегральной
implicit none                !  функции выбрана простейшая f(x)= 1.
real f, x
f=1
end

program tsttrap0             ! Пример программы, тестирующей trap0.
implicit none                !
integer n / 1 /              ! Количество участков дробления [a,b].
real    a /1.0/, b /10.0/    ! Пределы интегрирования.
real trap0                   ! Тип значения, возвращаемого trap0.
write(*,*) n, trap0(a,b,n)   ! Результат работы даже при n=1 верен в
end                          ! пределах 7 цифр: 9,000000.  (Почему?)
```

“Удобно ли пользоваться trap?”
На первый взгляд удобно: задали **a**, **b** и **n** – получили ответ для запрограммированной **f(x)**. Если надо вычислить значение интеграла для другой функции, то просто перепишем тело функции **f**.

Вопрос: Как быть, если основная программа должна вычислять в пределах одного пропуска интегралы не только от функции с именем **f**, но и от функций с другими именами (например, **g** и **w**)? Многократная перепись одного и того же алгоритма **trap0** с заменой имен **trap0** и **f** на очередные новые, очевидно, не вызывает энтузиазма. Так что, неудобство налицо. Хочется, чтобы вызов алгоритма интегрирования оформлялся, например, так: **trap1(f,a,b,n)**, где **f** – фактический параметр, являющийся именем функции, которая вычисляет подинтегральную функцию; **trap1** – имя функции интегрирования с желаемой формой вызова.

Как указать главной программе, что при обращении к функции **trap1(f,a,b,n)** имя **f** (в отличие от имен **a** и **b**) – это имя функции, а не простой переменной?

Указание дается посредством оператора **external**.

1.2 ФОРТРАН-оператор EXTERNAL

– определяет в единице компиляции, содержащей его, имена процедур, которые используются в качестве фактических аргументов при вызовах других процедур.
Общая форма оператора **external**:

```
external a1, a2, ..., an ! после external указываем имена внешних функций,
                         ! использующихся в качестве фактических
                         ! аргументов при обращении к другим функциям
```

Правильное оформление главной программы, вызывающей **trap1**:

```
program tsttrap2               !              Файл tsttrap2.f
implicit none
external f                     ! EXTERNAL - оператор описания.
integer n / 1 /                ! Поэтому должен предшествовать
real    a / 1.0 /, b / 10.0 /  ! всем операторам, выполняемым
real trap1, f                  ! программной единицей, в которой
write(*,*) n, trap1(f,a,b,n)   ! находится, а также описаниям
end                            ! операторов-функций.
```

Описание функции **trap1**:

```
function trap1(f,a,b,n)    !                  Файл trap1.f
implicit none              ! Функция trap1(f,a,b,n) реализует алгоритм
integer n, i               ! расчета по формуле трапеций интеграла по
real a, b, trap1, f        ! n подучасткам равномерного дробления [a,b]
real h, x, s               ! от функции f(x), имя которой (как "считает"
h=(b-a)/n                  ! trap1) передается  trap1 в качестве
s=(f(a)+f(b))/2            ! первого аргумента.
do i=2,n                   ! trap1 "знает", что f - это имя функции,
  x=a+(i-1)*h              ! так как компилятор при трансляции trap1
  s=s+f(x)                 ! "догадался" об этом по форме вызова f.
enddo                      ! А главная программа, если исключить из
trap1=s*h                  ! нее оператор external, "подсунет" вместо
end                        ! адреса функции адрес простой переменной.
```

И главная программа **tsttrap2**, и процедуры **trap1** и **f** компилируются независимо друг от друга. Если из текста главной программы **исключить** оператор **external**, то она при вызове **trap1** будет *рассматривать* имя **f** , как имя простой переменной, то есть подаст **trap1** в качестве первого аргумента адрес переменной, но не адрес области памяти, на который надо передавать управление для активации алгоритма расчета подинтегральной функции, как *полагает* **trap1**. Другими словами, налицо ошибка адресации: на вход к **trap1** подается не тот адрес, который ей нужен. При отсутствии **external** компиляция и линковка исполнимого файла завершатся успешно, но выполнение программы завершится неприятным сообщением

Segmentation fault (ошибка адресации).

1.2.1 Более содержательный пример с оператором EXTERNAL.

Используя функцию trap1(f,a,b,n), найти сумму интегралов

$$\int_{\mathbf{1}}^{\mathbf{10}} \mathbf{x}\mathbf{dx} + \int_{\mathbf{0}}^{\mathbf{2}} \mathbf{x^2}\mathbf{dx} + \int_{\mathbf{0.5}}^{\mathbf{1.5}} \mathbf{cos}(\mathbf{x})\mathbf{dx}$$

1. для упрощения контроля вычислений подинтегральные функции специально выбраны такими, чтобы интегралы брались аналитически.

2.
```
function f(x)              !  Поместим в файле  fun.f описания
implicit none              !  трех подинтегральных функций:
real f, x                  !
f=x                        !              f(x)=x ,
end                        !
function g(x)              !
implicit none              !
real g, x                  !
g=x*x                      !              g(x)=x*x,
end                        !
function w(x)              !
implicit none              !
real w, x                  !
w=cos(x)                   !              w(x)=cos(x)
end
```

3. В главной программе для упрощения контроля результатов работы функции **trap1** выведем и результаты расчета по формулам Ньютона-Лейбница:

```
program tsttrap3             !              Файл tsttrap3.f
implicit none
external f, g, w             !
integer nf, ng, nw           ! количества участков дробления
real af / 1.0 /, bf / 10.0 / ! пределы интегрирования функции f.
real ag / 0.0 /, bg /  2.0 / ! --"--"--"--"--"--"--"--"--"--  g.
real aw / 0.5 /, bw /  1.5 / ! --"--"--"--"--"--"--"--"--"--  w.
real trap1, f, g, w          ! тип значений, возвращаемых функциями.
real rf1, rg1, rw1, r1       ! результаты численного интегрирования.
real rf0, rg0, rw0, r0       ! --"--  аналитического интегрирования.
write(*,*) 'введи nf, ng, nw'              ! Ввод количеств участков
read (*,*) nf, ng, nw                      ! дробления.
write(*,*) ' nf=',nf, ' ng=',ng, ' nw=',nw
rf0=(bf*bf-af*af)/2                        ! Расчет интегралов по
rg0=(bg*bg*bg-ag*ag*ag)/3                  ! формуле Ньютона-Лейбница
rw0=sin(bw)-sin(aw)                        ! (для контроля)
r0=rf0+rg0+rw0
rf1=trap1(f,af,bf,nf); rg1=trap1(g,ag,bg,ng) ! Расчет интегралов через
rw1=trap1(w,aw,bw,nw); r1=rf1+rg1+rw1      ! вызов функции trap1.
write(*,'(4e15.7)') rf0, rg0, rw0, r0      ! Вывод результатов.
write(*,'(4e15.7)') rf1, rg1, rw1, r1      !
end
```

Пределы интегрирования у интегралов различны. Поэтому храним их в соответствующих переменных (например, **af,bf**, **ag,bg** и **aw,bw**).

4. Известно, что остаточный член $\mathbf{R_f(n)}$ формулы трапеций:

$$\int_{\mathbf{a}}^{\mathbf{b}} \mathbf{f(x)dx} = \mathbf{h} * \left[\frac{\mathbf{f(a)+f(b)}}{\mathbf{2}} + \sum_{\mathbf{k=2}}^{\mathbf{k=n}} \mathbf{f(x_k)}\right] + \mathbf{R_f(n)} \quad , \quad \mathbf{h} = \frac{\mathbf{b-a}}{\mathbf{n}} \quad ,$$

$$\mathbf{x_k} = \mathbf{a} + \mathbf{h} * (\mathbf{k-1}) \quad , \qquad \mathbf{R_f(n)} = -\frac{\mathbf{(b-a)^3}}{\mathbf{12n^2}} \cdot \mathbf{f''}(\xi)$$

пропорционален второй производной подинтегральной функции в какой-то точке ξ, принадлежащей промежутку интегрирования. Так что результат расчета интеграла от любой линейной функции $\mathbf{f(x)}$ должен быть практически идеален даже при одном-единственном промежутке дробления. Правда, тестирование функции полезно проводить не только при **n=1**, но и, например, при **n=2** для того, чтобы убедиться в правильности учета слагаемых под знаком $\sum$.

5. Результаты расчета можно свести в таблицу

Функция	f(x)=x	g(x)=x*x	w(x)=cos(x)	Сумма интегралов
Формула	rf0	rg0	rw0	r0
Ньютона-Лейбница	49,50000	2,666667	0,5180694	52,68473
Формула трапеций	rf1	rg1	rw1	r1
n				
1	49,50000	4,000000	0,4741599	53,97416
2	49,50000	3,000000	0,5072311	53,00723
3	49,50000	2,814815	0,5132636	52,82808
10	49,50000	2,680000	0,5176377	52,69764
100	49,50000	2,666800	0,5180652	52,68486
1000	49,50000	2,666668	0,5180698	52,68473E
10 000	49,50000	2,666669	0,5180705	52,68474
100 000	49,50003	2,666666	0,5180698	52,68476
1 000 000	49,53120	2,666756	0,5179611	52,71591

Сначала увеличение числа промежутков дробления ведет к повышению точности расчета, но при достижении некоторого предельного для используемой разрядности ячейки значения точность расчета ухудшается из-за распространения вычислительной ошибки. Для функции $\mathbf{f(x)=x}$ погрешность ее расчета при $\mathbf{x} \in [\mathbf{1}, \mathbf{10}]$ равна примерно $\mathbf{10^{-8}}$. При сложении приближенных величин их абсолютные погрешности складываются, так что на ячейках одинарной точности вклад от миллиона складываемых значений функции может оказаться порядка $\mathbf{1000000 \cdot 10^{-8} \approx 0.01}$, что и наблюдается (**49,53** вместо точного **49,50**).

6. Максимальные значения модулей вторых производных от функций $\mathbf{g(x)}$ и $\mathbf{w(x)}$ на промежутках интегрирования отличны от нуля. Поэтому для достижения необходимой точности можно оценить количество подучастков дробления **ng** и **nw**. Например, максимум модуля второй производной для функции $\mathbf{g(x)}$ не больше **2**. Из формулы для остатка, задавшись требуемой погрешностью (пусть $\mathbf{10^{-4}}$), находим, что **ng** можно выбрать примерно равным

$$\mathbf{ng} \geq \sqrt{\frac{\mathbf{2^3}}{\mathbf{12 \cdot 10^{-4}}} \cdot \mathbf{2}} = \mathbf{100} \cdot \sqrt{\frac{\mathbf{2}}{\mathbf{3}} \cdot \mathbf{2}} = \mathbf{200} \cdot \sqrt{\frac{\mathbf{1}}{\mathbf{3}}} \approx \mathbf{200/1.73} \approx \mathbf{116}$$

1.2.2 Возможные решения на ФОРТРАНе-95.

1. Предложенное выше решение должно проходить без каких-либо изменений исходных текстов и на ФОРТРАНе-95.

2. Однако, ФОРТРАН-95 для контроля при компиляции соответствия свойств фактических и формальных аргументов позволяет явно указать в главной программе интерфейс используемых функций. Например,

```
program tsttrap3                            ! Пример явного задания
implicit none                               ! в главной программе:
interface                                   !
  function f(x); real x,f; end function f ! интерфейса используемых
  function g(x); real x,g; end function g ! функций на ФОРТРАНе-95.
  function w(x); real x,w; end function w
  function trap1(f,a,b,n);         ! Можно и так:
    real trap1, a, b; integer n    ! real trap1, f, a, b; integer n
    real, external :: f            ! external f
end function trap1
end interface
integer nf, ng, nw           ! количества участков дробления
real af / 1.0 /, bf / 10.0 / ! пределы интегрирования функции f.
real ag / 0.0 /, bg /  2.0 / ! --"--"--"--"--"--"--"--"--  g.
real aw / 0.5 /, bw /  1.5 / ! --"--"--"--"--"--"--"--"--  w.
real rf1, rg1, rw1, r1       ! результаты численного интегрирования.
real rf0, rg0, rw0, r0       ! --"--  аналитического интегрирования.
write(*,*) 'введи nf, ng, nw'
read (*,*) nf, ng, nw;         ! Ввод количеств участков дробления.
write(*,*) ' nf=',nf, ' ng=',ng, ' nw=',nw
rf0=(bf*bf-af*af)/2; rg0=(bg*bg*bg-ag*ag*ag)/3; ! Контрольный расчет
rw0=sin(bw)-sin(aw); r0=rf0+rg0+rw0             ! по Ньютону-Лейбницу
rf1=trap1(f,af,bf,nf);                ! Численный расчет через вызов
rg1=trap1(g,ag,bg,ng)                 ! trap1 - функции численного
rw1=trap1(w,aw,bw,nw); r1=rf1+rg1+rw1 ! интегрирования.
write(*,'(4e15.7)') rf0, rg0, rw0, r0      ! Вывод результатов.
write(*,'(4e15.7)') rf1, rg1, rw1, r1      !
end
```

Так что, если при обращении **rg1=trap1(g,ag,bg,ng)** в спешке перепутали местами два последних аргумента (написали *rg1=trap1(g,ag,ng,bg)*), то компиляция главной программы завершится сообщением:

```
gfortran -c tsttrap3.f
 In file tsttrap3.f:24
    rg1=trap1(g,ag,ng,bg)
                   1
Error: Type/rank mismatch in argument 'b' at (1)
```

3. Использование модульного подхода может существенно сократить текст главной программы, создав модуль **quadra** с описанием функции **trap1**, и модуль **myfun** с описаниями функций **f**, **g** и **w**, численное интегрирование которых необходимо для решения задачи. Например,

```
program tsttrap3             !
use quadra                   ! Подключение модуля quadra
use myfun                    ! Подключение модуля myfun
implicit none
integer nf, ng, nw           ! количества участков дробления
real af / 1.0 /, bf / 10.0 / ! пределы интегрирования функции f.
real ag / 0.0 /, bg /  2.0 / ! --"--"--"--"--"--"--"--"--  g.
real aw / 0.5 /, bw /  1.5 / ! --"--"--"--"--"--"--"--"--  w.
real rf1, rg1, rw1, r1       ! результаты численного интегрирования.
real rf0, rg0, rw0, r0       ! --"--  аналитического интегрирования.
write(*,*) 'введи nf, ng, nw'
read (*,*) nf, ng, nw;       ! Ввод количеств участков дробления.
write(*,*) ' nf=',nf, ' ng=',ng, ' nw=',nw
rf0=(bf*bf-af*af)/2; rg0=(bg*bg*bg-ag*ag*ag)/3; ! Контрольный расчет
rw0=sin(bw)-sin(aw); r0=rf0+rg0+rw0             ! по Ньютону-Лейбницу
rf1=trap1(f,af,bf,nf);                ! Численный расчёт через
rg1=trap1(g,ag,bg,ng)                 ! вызов функции численного
rw1=trap1(w,aw,bw,nw); r1=rf1+rg1+rw1 ! интегрирования trap1.
write(*,'(4e15.7)') rf0, rg0, rw0, r0 ! Вывод результатов.
write(*,'(4e15.7)') rf1, rg1, rw1, r1 !
end
```

   (a) В модуль **quadra** можно включить и описания функций, реализующих другие методы численного интегрирования (по формуле прямоугольников, по формуле Симпсона и.т.д.) так, что подключение модуля **quadra** к главной программе обеспечит доступ к любой из описанных в **quadra** функций. При этом в главной программе не нужно указывать ни тип значения возвращаемого этими функциями, что было необходимо в ФОРТРАНе-77, ни явное описание интерфейса в ФОРТРАНе-95.

   (b) В раздел описаний модуля **myfun** можно поместить и имена параметров, от которых зависят подинтегральные функции (перед разделом **contains**). В этом случае имена этих параметров не надо описывать в программной единице, подсоединяющей модуль. Подобная техника (фактически) — путь определения в ФОРТРАНе глобальных переменных. В старых версиях ФОРТРАНа глобальны были только имена функций и **common**-блоков. Так что, вообще говоря, в современном ФОРТРАНе можно и выгоднее обходиться без **common**-блоков. Подробнее о **common**-блоках см. в следующей главе.

4. Исходные тексты модулей **quadra** и **myfun**:

```
module quadra  ! Модуль с процедурами численного интегрирования
contains       ! (пока в нём описана только одна функция trap1).
  function trap1(f,a,b,n);
  implicit none
  real trap1, a, b; integer n
  real, external :: f
  integer i
  real h, x, s
  h=(b-a)/n; s=(f(a)+f(b))/2
  do i=2,n
    x=a+(i-1)*h; s=s+f(x)
  enddo
  trap1=s*h
  end function trap1
end module quadra
```

```
module myfun                    ! Пример описания трёх подинтегральных
contains                        ! функций в модуле myfun:
  function f(x)                 ! функция
    implicit none               !
    real f, x;   f=x            !               f(x)=x ,
  end function f                !..............................
  function g(x)                 ! функция
    implicit none               !
    real g, x;   g=x*x          !               g(x)=x*x,
  end function g                !..............................
  function w(x)                 ! функция
    implicit none               !
    real w, x                   !
    w=cos(x)                    !               w(x)=cos(x)
  end function w                !..............................
end module myfun
```

5. Пример простого **make**-файла, обеспечивающего генерацию исполнимого файла и пропуск программы с модулями.

```
main :   tsttrap3.o myfun.o quadra.o
 gfortran tsttrap3.o myfun.o quadra.o -o main
myfun.o myfun.mod:  myfun.f
 gfortran -c myfun.f
quadra.o quadra.mod:  quadra.f
 gfortran  -c quadra.f
tsttrap3.o: tsttrap3.f myfun.mod quadra.mod
 gfortran -c tsttrap3.f
clear :
  rm *.o main
result : main
 ./main
```

1.3 ФОРТРАН-оператор INTRINSIC.

1.3.1 Уяснение ситуации

Пусть функция **trap1(f,a,b,n)** должна по ходу работы главной программы проинтегрировать функцию **sin(x)**. Если поступим формально – включим в главную программу оператор **external sin**, то на этапе компоновки исполнимого файла

```
program extersin               !              Файл extersin.f
implicit none
external sin
integer n / 1000 / ; real a / 1.0 /, b / 10.0 /
real trap1
write(*,*) n, trap1(sin,a,b,n)
end
```

```
function trap1(f,a,b,n)      !                  Файл trap1.f
implicit none                ! Функция trap1(f,a,b,n) реализует алгоритм
integer n, i                 ! расчета по формуле трапеций интеграла по
real a, b, trap1, f, h, x, s ! n подучасткам равномерного дробления [a,b]
h=(b-a)/n                    ! от функции f(x)
s=(f(a)+f(b))/2
    do i=2,n ;  x=a+(i-1)*h;  s=s+f(x); enddo; trap1=s*h
end
```

получим файла сообщение о том, что имя **sin** рассматривается компоновщиком не как имя встроенной ФОРТРАН-функции, а как имя внешней функции, которая должна быть написана программистом.

```
gfortran extersin.o trap1.o -o main
extersin.o(.text+0x55): In function ‘MAIN__’:
: undefined reference to ‘sin_’
```

Конечно, можно оформить свою специальную функцию **sinmy(x)**:

```
function sinmy(x)
implicit none
real sinmy, x
sinmy=sin(x)
end
```

обращение к которой из программы **extersin1.f**

```
program extersin1                    !      Файл extersin1.f
implicit none
external sinmy                       ! даст верный результат:
integer n / 1000 /                   !
real    a / 0.0 /, b / 1.5707963 /   !  1000   0.9999992
real trap1
write(*,*) n, trap1(sinmy,a,b,n)
end
```

В то же время ясно, что подобный выход – это расточительная трата времени на вызов своей специфической функции **sinmy(x)**, выполняющей роль своеобразного переходника к желаемой встроенной функции.

1.3.2 Назначение оператора INTRINSIC и пример.

Оператор **intrinsic**, как и оператор **external**, указывает атрибут имени функции, которая должна стать фактическим аргументом других процедур. Однако, в отличие от , служащего для указания имен процедур, напрсанных программистом (т.е. *внешних*), **intrinsic** служит для указания имён исключительно **встроенных** ФОРТРАН-функций. Например,

```
      program extersin2                   !  Файл extersin2.f
      implicit none
      external sinmy
      intrinsic sin, cos, exp, sqrt
      integer n / 1000 /                  ! число промежутков дробления
      real  a / 0.0 /, b / 1.5707963 /    ! a и b пределы интегрирования
      real trap1, sinmy, exp
      write(*,*) ' n=',n
      write(*,*) ' external  sinmy: ', trap1(sinmy,a,b,n)
      write(*,*) ' intrinsic sin  : ', trap1( sin,a,b,n)
      write(*,*) ' intrinsic cos  : ', trap1( cos,a,b,n)
      write(*,*) ' intrinsic exp  : ', trap1( exp,a,b,n),  exp(b)-1
      write(*,*) ' intrinsic sqrt : ', trap1(sqrt,a,b,n), 2*(b**1.5)/3
      end
```

```
$ g77 -c extersin2.f               !         Результат пропуска:
$ g77 -c trap1.f                   ! n= 1000
$ g77 -c sinmy.f                   ! external sinmy:   0.999999166
$ g77 extersin2.o trap1.o sinmy.o  ! intinsic sin  :   0.999999166
$ ./a.out                          ! intinsic cos  :   0.999999225
$                                  ! intinsic exp  :   3.81047487  3.81047678
$                                  ! intinsic sqrt :   1.3124541  1.31246746
```

1. Результат расчета интеграла от экспоненты получен и через **trap1** и по формуле Ньютона-Лейбница. В последнюю явно входит вызов встроенной функции **exp(b)**, которая вырабатывает значение типа **real**, что явно указано в 7-й строке исходного текста главной программы **real trap1, exp** и , вообще говоря, вызывает вопрос: "Зачем указывать тип значения возвращаемого встроенной функцией **exp**?" Если из этой строки **исключить** имя **exp**, то исполнимый код, полученный **g77**, при пропуске даст указанный выше результат, а первые версии компилятора **gfortran** могут сообщить об ошибке. В современных версиях **gfortran**а эта накладка исправлена.

2. В контексте выражения **exp(b)-1** имя **exp** родовое (тип результата определяется типом аргумента), а в контексте вызова **trap1(exp,a,b,n)** оно обязано быть специфическим. При отмене действия правила умолчания первые версии компилятора **gfortran**, видимо , требовали явного указания типа значения возвращаемого встроенной функцией со специфическим именем.

1.3.3 Родовое имя недопустимо в списке оператора INTRINSIC.

Перейдем в предыдущей программе на удвоенную точность:

```
program extersin3                              !   Файл extersin3.f
implicit none
external dsinmy
real(8), intrinsic :: dexp, dsqrt, dsin, dcos
integer n / 1000000/                           !
real*8  a / 0d0 /, b / 1.5707963267949d0 /, dtrap1, dsinmy
write(*,*) ' n=',n
write(*,'(a,d25.16)') ' external dsinmy : ', dtrap1(dsinmy,a,b,n)
write(*,'(a,d25.16)') ' intinsic dsin   : ', dtrap1(dsin,a,b,n)
write(*,'(a,d25.16)') ' intinsic dcos   : ', dtrap1(dcos,a,b,n)
write(*,'(a,d25.16,d25.16)') ' intinsic dexp   : ',dtrap1(dexp,a,b,n),dexp(b)-1
write(*,'(a,d25.16,d25.16)') ' intinsic dsqrt  : ', &
                            dtrap1(dsqrt,a,b,n), 2*(b**1.5d0)/3
1001 format(' n=',i7,' external sinmy: ',d25.16)
end


function dtrap1(f,a,b,n)        ! Файл dtrap1.f95  содержит исходный текст
implicit none                   ! функции dtrap1(f,a,b,n) расчета интеграла
integer n, i                    ! по формуле трапеций для n подучастков
real*8 a, b, dtrap1, f, h, x, s ! равномерного дробления промежутка [a,b].
h=(b-a)/n
s=(f(a)+f(b))/2
do i=2,n ;  x=a+(i-1)*h;  s=s+f(x); enddo;
dtrap1=s*h
end


function dsinmy(x)
implicit none
real*8 dsinmy, x
dsinmy=sin(x)
end
```

При вызовах **dtrap1** имя фактического аргумента, являющегося встроенной функцией, должно быть специфическим, а не родовым (т.е. **dsin**, а не **sin**; **dcos**, а не **cos**; **dexp**, а не **exp**; **dsqrt**, а не **sqrt**). В этом случае получим:

```
 n=     1000000
external dsinmy :    0.9999999999997863D+00
intinsic   sin  :    0.9999999999997863D+00
intinsic cos    :    0.9999999999998057D+00
intinsic exp    :    0.3810477380966059D+01   0.3810477380965368D+01
intinsic sqrt   :    0.1312467495067690D+01   0.1312467495476873D+01
```

Использование же родового имени в качестве имени фактического аргумента может спровоцировать нас на поиск ошибки там, где её нет. Например, в формулах для подинтегральных выражений:

```
program extersin3a   ! Использование родового имени в качестве
implicit none        ! фактического аргумента может спровоцировать
external dsinmy      ! нас на поиск ошибок совершенно иной природы,
real*8, intrinsic :: exp, sqrt, sin, cos     !например, в формулах.
integer n / 1000000/
real*8  a / 0d0 /, b / 1.5707963267949d0 /, dtrap1, dsinmy
write(*,*) ' n=',n
write(*,'(a,d25.16)') ' external dsinmy : ', dtrap1(dsinmy,a,b,n)
write(*,'(a,d25.16)') ' intinsic   sin  : ', dtrap1(sin,a,b,n)
write(*,'(a,d25.16)') ' intinsic cos    : ', dtrap1(cos,a,b,n)
write(*,'(a,d25.16,d25.16)') ' intinsic exp    : ',dtrap1(exp,a,b,n),exp(b)-1
write(*,'(a,d25.16,d25.16)') ' intinsic sqrt   : ', &
                            dtrap1(sqrt,a,b,n), 2*(b**1.5d0)/3
1001 format(' n=',i7,' external sinmy: ',d25.16)
end
```

```
 n=     1000000
external dsinmy :    0.9999999999997863D+00
intinsic   sin  :                       NaN
intinsic cos    :                       NaN
intinsic exp    :                       NaN   0.3810477380965368D+01
intinsic sqrt   :                       NaN   0.1312467495476873D+01
```

1.4 Передача имени С-функции в качестве аргумента.

Передача имени функции в качестве фактического аргумента другой функции требует правильного описания соответствующего формального аргумента.

В **ФОРТРАНе-77** правильное описание сводилось к описанию типа значения, возвращаемого через имя функции, так что в программной единице, вызывающей функцию, аргументом которой является имя другой функции, приходилось имя последней специально указывать в списке оператора **external**. В **ФОРТРАНе-95** добавились возможности указания явного интерфейса функции, либо модульного.

СИ сопоставляет имени функции **указатель на функцию**. Поэтому формальный аргумент, соответствующий имени функции, должен быть указателем на функцию. Сообщить об этом функции **trap** через ее заголовок можно, например, так:

```
double trap(double (*f)(double),double, double, int)

#include <stdio.h>                                      //  Файл tsttrap.c

double trap(double (*)(double),double,double,int);  // Прототипы: trap,
double f(double);                                    //            f
double g(double);                                    //            g
double w(double);                                    //            w
int main()
{double a=1.0, b=10.0, rf, rg, rw;
 int n;
 printf("input n\n"); scanf("%d",&n);
 printf("  a=%le    b=%le   n=%d\n", a, b, n);
 rf=trap(f,a,b,n); rg=trap(g,a,b,n); rw=trap(w,a,b,n);
 printf("    rf=%le    rg=%le  rw=%le\n", rf, rg, rw);
 return 0;
}

double trap(double (*f)(double), double a, double b, int n) //  Файл mytrap.c
{ double s, x, h; int i;
  s=(f(a)+f(b))/2; h=(b-a)/n;
  for (i=1;i<n;i++) { x=a+h*i; s+=f(x); } return(s*h);
}

double f(double x) { return x;      }            //  Файл myfun.c
double g(double x) { return x*x;    }
double w(double x) { return cos(x); }
```

Предложенная форма описания заголовка функции **trap** удобна, когда он помещается в одной строке. Тем не менее, видно, что описание первого аргумента функции **trap** не столь кратко, как описания остальных, поскольку содержит сведения не только о его типе, но и о типах и количестве аргументов первого.

СИ позволяет описать тип первого аргумента по аналогии с остальными, сопоставив этому типу имя, придуманное пользователем. Сопоставление, как вспоминаем, осуществляется посредством оператора **typedef**.

1.4.1 Напоминание об операторе typedef.

Оператор **typedef** определяет псевдоним имени существующего типа. Например,

1. **typedef float real** позволит вместо **float a, h, s** использовать **real a, h, s** так, что перевод программы на удвоенную точность сведётся к одному единственному переопределению типа **real** посредством **typedef double real**

```
#include <iostream>                    //  Файл tsttrap.c
using namespace std;                   //
typedef long double real;              //  real            Результат:
int main()                             //  float              inf
{ real a=1.0e200, h=1.0e300, s=a*h;    //  double             inf
  cout <<s<<endl; return 0;            //  long double        1e+500
}
```

2. При описании переменных типа массив бывает неудобно после каждого имени явно указывать структуру массива посредством конструктора **[]**. Проще один раз определить имя типа и его использовать для описания переменных:

```
typedef double xyz[3];  // Определили имя типа xyz (массив из трех элементов)
int main()              // Объявляем три массива a, b и c типа xyz, хотя
{ xyz a, b, c;}         // могли бы объявить и так: double a[3], b[3], c[3];
```

3. Аналогично через оператор **typedef** определим псевдоним типа подинтегральной функции, используемой функцией **trap**. Поместим его в файл **mytype.h**:

```
typedef double (*pfun)(double); // Определение псевдонима pfun.
```

pfun – имя типа указателя на функцию, имеющей один аргумент типа **double** и возвращающей через имя функции значение типа **double**. Включив в файлы **tsttrap.c** и **mytrap.c** инструкцию #**include "mytype.h"**, можем описать формальный параметр-функцию подобно описанию остальных параметров, исключив из заголовка функции **trap** лишние подробности:

```
#include <stdio.h>                                  // Файл tsttrap.c
#include "mytype.h"
double trap(pfun,double,double,int);
double f(double); double g(double); double w(double);
int main()
{ double a=1.0, b=10.0, rf, rg, rw;  int n;
  printf("input n\n"); scanf("%d",&n);
  printf("  a=%le    b=%le   n=%d\n", a, b, n);
  rf=trap(f,a,b,n); rg=trap(g,a,b,n); rw=trap(w,a,b,n);
  printf("    rf=%le    rg=%le  rw=%le\n", rf, rg, rw);  return 0; }
```

```
#include "mytype.h"                                 //  Файл mytrap.c
double trap(pfun f, double a, double b, int n)
{ double s, x, h; int i;
  s=(f(a)+f(b))/2; h=(b-a)/n;
  for (i=1;i<n;i++) { x=a+h*i; s+=f(x); } return(s*h);
}
```

1.4.2 Фактический аргумент – бибиотечная СИ-функция.

В СИ (в отличие от ФОРТРАНа) не требуется выделять каким-то особым служебным словом имена функций **sin**, **cos**, **exp**, **sqrt** и т.д.; достаточно лишь посредством оператора #**include** подключить нужный заголовочный файл с их прототипами.

```
typedef double (*pfun)(double); // Определение псевдонима pfun.

#include <stdio.h>                                  // Файл tsttrap.c
#include <math.h>
#include "mytype.h"
double sinmy(double x)
{return sin(x);}
double trap(pfun,double,double,int);
int main()
{ double a=0.0, b=1.5707963, rsmy, rsin;
  int n;
  printf("input n\n"); scanf("%d",&n);
  printf("  a=%e    b=%e   n=%d\n", a, b, n);
  printf(" sinmy : %e\n", trap(sinmy,a,b,n));
  printf(" sin   : %e\n", trap(sin  ,a,b,n));
  printf(" cos   : %e\n", trap(cos,a,b,n));
  printf(" exp   : %e   %e\n", trap(exp,a,b,n),exp(b)-1);
  printf(" sinmy : %e   %e\n", trap(sqrt,a,b,n),2*pow(b,1.5)/3);
  return 0;
}

#include "mytype.h"                                 //  Файл mytrap.c
double trap(pfun f, double a, double b, int n)
{ double s, x, h; int i;
  s=(f(a)+f(b))/2; h=(b-a)/n;
  for (i=1;i<n;i++) { x=a+h*i; s+=f(x); } return(s*h);
}

$ make result
gcc -c tsttrap.c
gcc -lm tsttrap.o mytrap.o -o main
./main
input n
1000
  a=0.000000e+00    b=1.570796e+00   n=1000
 sinmy : 9.999998e-01
 sin   : 9.999998e-01
 cos   : 9.999998e-01
 exp   : 3.810478e+00   3.810477e+00
 sinmy : 1.312455e+00   1.312467e+00
```

1.5 О чем узнали из первой главы? (что повторили?)

1. Часто бывает выгодно в качестве аргумента одной процедуры использовать имя другой процедуры (так в процедуре численного интегрирования полезен формальный параметр для имени подинтегральной функции).

2. ФОРТРАН-программа, вызывая процедуру, у которой фактическим аргументом служит имя другой процедуры, должна как-то *уметь отличать* ссылку на функцию от ссылки на простую переменную (в первом случае процедуре следует передать адрес функции, которой надо передать управление; во втором – обеспечить возможность работы с содержимым ячейки по указанному адресу).

3. **EXTERNAL** — оператор описания, информирующий ФОРТРАН-программу через свой список об именах процедур, которые служат фактическим параметрами при вызовах каких-то других процедур.

4. Рассмотрели пример описания и вызова процедуры расчета интеграла по формуле трапеций. Остаток формулы трапеций пропорционален второй производной подинтегральной функции. Поэтому начальное тестирование процедуры интегрирования выгодно проводить на линейных функциях (*Почему?*).

5. ФОРТРАН-77 при компиляции не контролирует соответствие числа фактических аргументов и их типов числу и типам формальных параметров, формируя неявный интерфейс процедур – просто по обращению к ним. Поиск возможных несоответствий не так прост как хотелось бы.

6. ФОРТРАН-95 предоставляет средства автоматического контроля интерфейса за счет описания в вызывающей процедуру программе **интерфейсного блока**. Это описание можно провести в нескольких формах (явно или через модуль).

7. В список оператора **EXTERNAL** следует включать только *специфическое* имя процедуры, то есть имя конкретной процедуры, а НЕ имя интерфейсного блока.

8. Если фактическим аргументом процедуры служит имя **встроенной** функции, то его следует включить в список оператора **INTRINSIC**, а НЕ **external**.

9. В СИ признаком того, что формальный параметр есть имя функции, служит его описание как **указателя на функцию**. Два варианта такого описания:

   (a) явно, согласно синтаксису описания указателя на функцию, например:

   ```
   double trap( double (*)(double), double, double, int)
   ```

   (b) посредством использования псевдонима типа указателя на функцию. Например,

   ```
   typedef double(*pfun)(double);          //   более
   double trap(pfun, double, double, int); //        практично
   ```

1.6 Первое домашнее задание (III-семестр)

Квадратурную формулу для расчета интеграла $\mathbf{S} = \int_{\mathbf{a}}^{\mathbf{b}} \mathbf{f}(\mathbf{x})\mathbf{dx}$ от функции f(x) по заданному промежутку **[a,b]**) можно записать в виде:

$$\mathbf{S} = \tilde{\mathbf{S}} + \mathbf{R_f(n)}$$

Здесь

1. $\tilde{\mathbf{S}}$ – **квадратурная сумма**, обеспечивающая в зависимости от схемы дискретизации (разбиения) промежутка **[a,b]** на **n** подучастков и правила аппроксимации подинтегральной функции, расчет приближенного значения интеграла по **n** или **n+1** узлам (точкам дискретизации промежутка **[a,b]**).

2. $\mathbf{R_f(n)}$ – остаток квадратурной формулы, зависящий не только от числа дроблений промежутка интегрирования, но и от подинтегральной функции.

3. **Простейшие из квадратурных формул:**

Название формулы	Квадратурная сумма: $\tilde{\mathbf{S}}$	Остаток $\mathbf{R_f(n)}$
средних прямоугольников	$\mathbf{h} \cdot \sum_{\mathbf{k=1}}^{\mathbf{n}} \mathbf{f}(\mathbf{x_k})$	$\frac{\mathbf{(b-a)^3}}{\mathbf{24n^2}} \cdot \mathbf{f}''(\xi)$
трапеций	$\mathbf{h} \cdot \left[\frac{\mathbf{f(a)+f(b)}}{\mathbf{2}} + + \sum_{\mathbf{k=2}}^{\mathbf{n}} \mathbf{f}(\mathbf{x_k})\right]$	$\frac{\mathbf{(b-a)^3}}{\mathbf{12n^2}} \cdot \mathbf{f}''(\xi)$
Симпсона или парабол	$\frac{\mathbf{h}}{\mathbf{3}} \cdot [\mathbf{f(a)+f(b)+4 \cdot S_1 + 2 \cdot S_2}]$ $\mathbf{S_1 = f_2 + f_4 + \cdots + f_{n=2m}}$ $\mathbf{S_2 = f_3 + f_5 + \cdots + f_{n-1}}$ $\mathbf{f_1 = f(a)};\quad \mathbf{f_{n+1} = f(b)};\quad \mathbf{f_k = f(x_k)}$	$\mathbf{n = 2m}$ $\frac{\mathbf{(b-a)^5}}{\mathbf{2880m^4}} \cdot \mathbf{f^{(4)}}(\xi)$

4. Пределы интегрирования **a** и **b** и число промежутков дробления **n** – задаем, но про ξ – аргумент производной, входящей в формулу остатка, знаем лишь, что $\xi \in [a, b]$. Поэтому при оценке остатка в худшем случае придется использовать некоторую оценку максимального значения производной на **[a,b]**.

5. В простейших квадратурах шаг дискретизации равномерный: $\mathbf{h} = \frac{\mathbf{b-a}}{\mathbf{n}}$.

6. **n** (количество подучастков дробления) в случае формулы прямоугольников равно числу узлов дискретизации и $\mathbf{x_k = a + (k - 0.5) \cdot h}$.

7. В случае формул трапеций и Симпсона **n** на единицу меньше числа узлов, т.е. при нумерации узлов с единицы имеем: $\mathbf{x_1 = a, x_{n+1} = b, x_k = a + (k - 1) * h}$.

8. Помним, что в случае формулы Симпсона **n** (число подучастков равномерного дробления промежутка интегрирования) должно быть обязательно **четным**

Задача N 1 (ФОРТРАН-77). Разработать функции расчёта квадратурной суммы и программу, тестирующую их, в режиме **real(8)** по формулам: трапеций – **trap(f,a,b,n)**; средних прямоугольников – **rectan(f,a,b,n)**; Симпсона – **sim(f,a,b,n)** от функции **f(x)**, вычисляемой на промежутке **[a,b]** ФОРТРАН-функцией.

Задача N 2 (ФОРТРАН-77). Разработать функцию **quadra(fqua,f,a,b,n)** расчета **квадратурной суммы** по формуле, задаваемой функцией **fqua**, моделирующей возможный фактический параметр при вызове **quadra**. Таким параметром может быть имя любой из функций (**trap**, **rectan** или **sim**). В качестве тестирующей программы удобно взять предыдущую, нарастив её соответствующими вызовами.

Задача N 3 (ФОРТРАН-95). Описать в тестирующей программе явно интерфейс функций **quadra(fqua,f,a,b,n)**, **trap**, **rectan** и **sim**, разработанных в предыдущих задачах, и продемонстрировать ее работоспособность.

Задача N 4 (ФОРТРАН-95). Разработать **модуль** с функциями **quadra**, **trap**, **rectan** и **sim**. Продемонстрировать его работу при решении предыдущей задачи.

Задача N 5 (ФОРТРАН-95). Дополнить модуль, разработанный в предыдущей задаче, функциями расчёта интеграла, разработанными в прошлом семестре, которым в качестве аргумента, задающего функцию, вместо имени функции подаётся одномерный массив её значений. Обеспечить вызов по единому имени как квадратуры с аргументом функцией, так и квадратуры с аргументом массивом. Протестировать во всех вариантах правильность работы квадратур за один пропуск программы. Обеспечить вывод, наглядно демонстрирующий сравнение результатов.

Задача N 6 (ФОРТРАН-95). Оформить алгоритм метода двойного пересчета подпрограммой **quadrauto(fqua,f,a,b,eps,n,k,ier,res)**. Здесь

1. **eps** – предельно допустимая величина разности между двумя последовательными приближениями метода двойного пересчета;
2. **n** – начальное число подучастков дробления промежутка интегрирования;
3. **k** – количество удвоений подучастков дробления, при котором достигнута требуемая величина **eps**;
4.

ier	код причины завершения работы подпрограммы:
0	требуемое eps достигнуто за разумное количество удвоений.
1	требуемое eps НЕ достигается за разумное количество удвоений

5. остальные параметры имеют тот же смысл, что и в **quadra**.

Добавить описание подпрограммы **quadrauto** в модуль, используемый для решения задачи **N 4**, после чего разработать программу тестирования функции **quadrauto** и продемонстрировать её работоспособность.

Задача N 6 Разработать подпрограмму, которая получает матрицу обратную исходной.

О методе двойного пересчета. Во многих задачах подинтегральные функции настолько сложны, что оценка остатка квадратурной формулы весьма трудоемка. Поэтому на практике часто используют доведение величины интеграла до заданной точности посредством метода двойного пересчета (правило Рунге): именно, перерасчет интеграла с уменьшением вдвое шага интегрирования до тех пор, пока разность между двумя последовательно полученными значениями интеграла не окажется достаточно мала.

Пусть $\tilde{\mathbf{S}}_{\mathbf{n}}$ квадратурная формулы при постоянном шаге дробления промежутка интегрирования

$$\tilde{\mathbf{S}}_{\mathbf{n}} = \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{C_i f(x_i)} \quad .$$

Здесь $\mathbf{n = 2^k}$ – количество узлов. Построим последовательность $\{\tilde{\mathbf{S}}_{\mathbf{2^k}}\}$:

k	1	2	3	4	$\cdots$	p	$p+1$
$\tilde{S}_{2^k}$	$\tilde{S}_2$	$\tilde{S}_4$	$\tilde{S}_8$	$\tilde{S}_{16}$	$\cdots$	$\tilde{S}_{2^p}$	$\tilde{S}_{2^{p+1}}$

Допустим, что при $\mathbf{k \to \infty}$ $\tilde{\mathbf{S}}_{\mathbf{2^k}}$ стремиться к истинной величине интеграла.

Зададимся некоторым $\epsilon > \mathbf{0}$ и прекратим вычисление $\tilde{\mathbf{S}}_{\mathbf{2^k}}$ как только окажется, что при очередном удвоении количества узлов

$$|\tilde{\mathbf{S}}_{\mathbf{2^p}} - \tilde{\mathbf{S}}_{\mathbf{2^{p+1}}}| \leq \epsilon \quad ,$$

после чего в качестве основного результата (величины искомого интеграла $\mathbf{S}$ примем значение $\mathbf{S_{2^{p+1}}}$. Строго говоря, в общем случае, нельзя гарантировать, что

$$|\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{2^{p+1}}}| \leq \epsilon \quad .$$

Однако, если предположить, что поведение остатка квадратурной формулы такое же как у функции $\mathbf{h^m}$, где $\mathbf{m \geq 1}$ или, другими словами,

$$\mathbf{R_f(n) = O(h^m)} \quad ,$$

или, что то же

$$\mathbf{R_f(n)} = \lim_{\substack{\mathbf{n \to \infty} \\ \mathbf{h \to 0}}} \frac{\mathbf{R_f(n)}}{\mathbf{h^m}} = \mathbf{const} \quad ,$$

или, иначе говоря,

$$\mathbf{R_f(n) = M \cdot h^m} \quad ,$$

то будем иметь

$$\frac{\mathbf{R_f(n1)}}{\mathbf{R_f(n2)}} = \frac{\mathbf{M \cdot h_{n1}^m}}{\mathbf{M \cdot h_{n2}^m}} = \frac{\left(\dfrac{\mathbf{b-a}}{\mathbf{n1}}\right)^{\mathbf{m}}}{\left(\dfrac{\mathbf{b-a}}{\mathbf{n2}}\right)^{\mathbf{m}}} = \left(\frac{\mathbf{n2}}{\mathbf{n1}}\right)^{\mathbf{m}} = \frac{\mathbf{1}}{\alpha^{\mathbf{m}}} \quad .$$

Здесь $\alpha \equiv \dfrac{\mathbf{n1}}{\mathbf{n2}} < \mathbf{1}$ – отношение количества узлов квадратуры до и после уменьшения шага. Заметим, что пока правило уменьшения шага (или что по сути – то же, правило увеличения количества узлов с **n1** до **n2**) не используется. Важно лишь, чтобы оно приводило к справедливости неравенства $\mathbf{n1} < \mathbf{n2}$. Тогда

$$\frac{\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n1}}}{\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n2}}} = \frac{\mathbf{R_f(n1)}}{\mathbf{R_f(n2)}} = \frac{\mathbf{1}}{\alpha^{\mathbf{m}}} \quad ,$$

то есть отличие текущего приближения от истинной величины интеграла больше соответствующего отличия следующего (поскольку $\dfrac{\mathbf{1}}{\alpha^{\mathbf{m}}} > \mathbf{1}$). Таким образом, если надеемся, что остаток квадратурной формулы пропорционален $\mathbf{h^m}$, то приближение значения квадратурной суммы к правильному результату гарантируется. Поскольку правильный результат – неизвестен, то полезно выяснить: "Как отличие двух последовательных приближений связано с отличием окончательного приближения от точного значения?"

$$\frac{\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n1}}}{\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n2}}} = \frac{\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n2}} + \tilde{\mathbf{S}}_{\mathbf{n2}} - \tilde{\mathbf{S}}_{\mathbf{n1}}}{\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n2}}} = \mathbf{1} + \frac{\tilde{\mathbf{S}}_{\mathbf{n2}} - \tilde{\mathbf{S}}_{\mathbf{n1}}}{\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n2}}} = \frac{\mathbf{1}}{\alpha^{\mathbf{m}}} \quad .$$

Так что

$$\frac{\tilde{\mathbf{S}}_{\mathbf{n2}} - \tilde{\mathbf{S}}_{\mathbf{n1}}}{\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n2}}} = \frac{\mathbf{1}}{\alpha^{\mathbf{m}}} - \mathbf{1} = \frac{\mathbf{1} - \alpha^{\mathbf{m}}}{\alpha^{\mathbf{m}}} \quad .$$

И, окончательно,

$$|\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n2}}| = \frac{\alpha^{\mathbf{m}}}{\mathbf{1} - \alpha^{\mathbf{m}}} \cdot |\tilde{\mathbf{S}}_{\mathbf{n2}} - \tilde{\mathbf{S}}_{\mathbf{n1}}| \quad .$$

Это означает, что абсолютная погрешность метода двойного пересчета $|\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n2}}|$ будет составлять от требуемой разности двух последовательных приближений ϵ не более доли $\dfrac{\alpha^{\mathbf{m}}}{\mathbf{1} - \alpha^{\mathbf{m}}}$, то есть окажется меньше ϵ в $\dfrac{\mathbf{1} - \alpha^{\mathbf{m}}}{\alpha^{\mathbf{m}}}$ раз.
В частности, при двукратном увеличении количества узлов ($\alpha = \mathbf{0.5}$) имеем:

Формула	$\mathbf{R_f(n)}$	$\mathbf{m}$	$\dfrac{\mathbf{1} - \alpha^{\mathbf{m}}}{\alpha^{\mathbf{m}}}$
средних прямоугольников	$\mathbf{O(h^2)}$	**2**	**3**
трапеций	$\mathbf{O(h^2)}$	**2**	**3**
Симпсона	$\mathbf{O(h^4)}$	**4**	**15**

Из таблицы видно, что если добились между двумя последовательными приближениями метода двойного пересчета отличия меньшего, например, $\mathbf{0,01}$, то реально достигнутая точность в случае метода Симпсона в **15** раз выше.

Замечание Метод двойного пересчёта часто называют методом Рунге.

2 Передача функции дополнительных параметров.

2.1 Уяснение ФОРТРАН-ситуации.

Пусть **trap1(f,a,b,n)** — функция расчета интеграла по формуле трапеций. Здесь первый аргумент – имя подинтегральной функции, второй и третий – пределы интегрирования, а четвертый – число подучастков дробления.

В предыдущем разделе, подинтегральные функции зависели только от одной независимой переменной интегрирования **x**: **f(x)=x**, **g(x)=x*x**, **w(x)=sin(x)**.

Уместен вопрос:

"Что делать, если подинтегральная функция зависит еще и от некоторых дополнительных параметров? Как передать их значения подинтегральной функции?

Например, **f(x)=x+p**, **g(x)=x*x+p*x+q**, **w(x)=r*sin(p*x+q)**).

Здесь **p**, **q**, и **r** – дополнительные параметры, которые вводятся или вычисляются программной единицей, обращающейся к **trap1**.

"Как их значения передать функциям f(x), g(x) и w(x)?"

1. В языке СИ есть понятие **глобальной переменной**. Так что дополнительные параметры **p**, **q**, **r**, входящие в функции **f(x)**, **g(x)**, **w(x)**, можно оформить **глобальными переменными**, которые будут доступны требуемым функциям (см. пункт **2.7 Моделирование common-блока или "module" в СИ**).

2. Альтернативный подход — сопоставить дополнительным параметрам соответствующие формальные аргументы (т.е. (**f(x,p)**, **g(x,p,q)**, **w(x,r,p,q)**) — резко снижает универсальность функции **trap1**, так как требует изменения ее текста в местах обращения к функции расчета подинтегрального выражения. Например, вместо вызова **f(x)** придется писать **f(x,p)**; вместо **g(x)** – **g(x,p,q)** и вместо **w(x)** – **w(x,r,p,q)**). Нас не устраивает переписывать каждый раз алгоритм **trap1** при изменении количества дополнительных параметров. Поэтому надежным и простым средством СИ сохранить универсальность **trap1** остается механизм передачи через глобальные параметры.

3. **В ФОРТРАНе-77 нет понятия глобальной переменной.** Любая переменная описанная внутри какой-либо единицы компиляции — локальна по отношению к ней, т.е. существует только в её пределах и не видна другим единицам компиляции. *Кажется*, что передать дополнительные параметры функции ФОРТРАНа-77, минуя список дополнительных формальных аргументов, невозможно.

4. **На самом деле** ФОРТРАН-77 такой механизм предоставляет.

5. Это — механизм совмещения по оперативной памяти переменных из разных единиц компиляции, который запускается через оператор **common** (оператор **common**-блока или **общего**-блока)

2.2 Оператор COMMON.

Общей областью или **COMMON**-блоком в ФОРТРАНе называется область оперативной памяти, предназначенная для размещения в ней данных нужных при выполнении программы нескольким единицам компиляции.
Для определения общих областей и указания помещаемых в них переменных и массивов используется оператор **COMMON**. Типичная форма его записи:

common / имя общей области / Список ее элементов

1. **Имя общей области** – желательный, но необязательный параметр (если присутствует, то область называют **именованной**, иначе – **неименованной**).

2. **Список элементов** – имена переменных или массивов. Они **НЕ должны быть формальными аргументами** процедуры, использующей данную общую область. Элементы списка отделяются друг от друга запятыми.

3. Одна программная единица может иметь несколько операторов **common**, хотя и один **common** может определять несколько общих областей, если после завершения списка элементов предыдущей общей области поместить имя следующей (ограниченное с обеих сторон значком /) со списком ее элементов.

2.2.1 Пример использования оператора COMMON.

```
program tstcommon    !                          Файл tstcommon.f
implicit none        ! Отмена действия правила умолчания.
external f, g, w     ! Имена функций, которые будут  фактич. аргументами.
integer nf, ng, nw             ! количества участков дробления
real af / 1.0 /, bf / 10.0 /   ! пределы интегрирования функции f.
real ag / 0.0 /, bg /  2.0 /   ! --"--"--"--"--"--"--"--"--  g.
real aw / 0.5 /, bw /  1.5 /   ! --"--"--"--"--"--"--"--"--  w.
real trap1, f, g, w            ! тип значений, возвращаемых функциями.
real rf1, rg1, rw1, r1         ! Результаты численного интегрирования.
real rf0, rg0, rw0, r0         ! --"--  аналитического интегрирования.
real p, q, r                   ! Тип дополнительных параметров.
common / par / p, q, r         ! Объединение их в common-блоке /par/
write(*,*) 'введи nf, ng, nw'
read (*,*) nf, ng, nw;  write(*,*) ' nf=',nf,' ng=',ng,' nw=',nw
write(*,*) ' введи параметры p, q, r:'
read (*,*) p, q, r;     write(*,*) ' p=',p,' q=',q,' r=',r
rf0=((bf+p)**2-(af+p)**2)/2                     ! Расчет по формуле
rg0=(bg**3-ag**3)/3+p*(bg**2-ag**2)/2+q*(bg-ag) !  Ньютона-Лейбница
rw0=r*(sin(p*bw+q)-sin(p*aw+q))/p               ! (для контроля)
r0=rf0+rg0+rw0
rf1=trap1(f,af,bf,nf); rg1=trap1(g,ag,bg,ng) ! Расчет интегралов
rw1=trap1(w,aw,bw,nw); r1=rf1+rg1+rw1;       ! через вызов trap1.
write(*,'(4e15.7)') rf0, rg0, rw0, r0        ! Вывод
write(*,'(4e15.7)') rf1, rg1, rw1, r1        !        результатов.
end
```

1. Здесь приведен текст главной программы, демонстрирующей использование оператора **common** для решения задачи, поставленной в пункте **2.1**.

2. После описания типа дополнительных параметров **real p, q, r** осуществлено их размещение в **common**-блоке с именем **par** (первым располагается параметр **p**, следом **q**, а за ним **r**).

3. Как только будут заданы их значения – любая функция, в которой встретится оператор **common / par / p, q, r**, сможет использовать их.

4. В данном примере один **common**-блок с именем **par** нацелен на обслуживание каждой из трех подинтегральных функции (и **f(x)**, и **g(x)**, и **f(x)**), несмотря на то, что для расчета функции **f(x)**, например, параметры **q** и **r** не нужны; а для расчета функции **g(x)** не нужен параметр **r**.

```
function f(x)          ! В файле fun.f даны описания трех функций
 implicit none         ! с парметрами,  передаваемыми через
real f, x, p, q, r     ! common-блок с именем   / par /. Так как
common /par/ p, q, r   ! внешние функции компилируются полностью
f=x+p                  ! независимо друг от друга, то оператор common
end                    ! должен присутствовать в  каждой из них (если,
function g(x)          ! конечно, ей нужны  параметры, передаваемые
implicit none          ! через common).
real g, x, p, q, r     !
common /par/ p, q, r     ! Достоинство оператора common: имена
g=x*x+p*x+q              ! переменных одного и того же common-блока
end                      ! в разных единицах компиляции не обязаны
function w(x)            ! одинаковыми: w(x) первую переменную common-
implicit none            ! блока именует pp, а главная программа - p,
common /par/ pp, qq, rr  ! т.е. common позволяет не переобозначать
real w, x, pp, qq, rr    ! переменные,  что невозможно при
w=rr*cos(pp*x+qq)        ! использовании глобальных параметров.
end
```

5. Результат пропуска демонстрационной программы:

```
 nf=        1000  ng=        1000  nw=        1000

 p=  0.6000000      q=  0.7000000      r=  0.8000000

0.5490000E+02  0.5266667E+01  0.2108035E+00  0.6037747E+02
0.5489999E+02  0.5266668E+01  0.2108036E+00  0.6037746E+02
```

6. Важно, что текст функции **trap1** никак не затрагивается **common**-блоком /**par**/. Так что **trap1** позволяет вычислять интеграл по заданному промежутку от подинтегральной функции, зависящей от любого конечного количества дополнительных параметров, несмотря на то, что внутри **trap1** обращение к подинтегральной функции формально остается обращением к функции с одним аргументом.

2.2.2 Некоторые примеры описания оператора common.

1. ```
   common a, b / par1 / d, e / par2 / f  /par1/ c  ! Вряд ли удобно,
   common g, h / par2 / w, x / par1 / c1 // z      ! НО ДОПУСТИМО.
   ```

   Эти два оператора **common** действуют как один:

   ```
   common a, b, g, h, z / par1 / d, e, c, c1 / par2 / f, w, x
   ```

2. Данные, располагаемые в некотором **common**-блоке одной единицы компиляции, будут занимать ту же память, что и данные, расплагаемые в этом же **common**-блоке, используемом другой единицей компиляции. Например,

   ```
   program test       !   subroutine sub1     !   subroutine sub2
                      !   common vect(5)      !   common ssss(5)
   call sub1          !   do i=1,5            !   write(*,*) ssss
   call sub2          !     vect(i)=5*i       !   end
   end                !   enddo
                      !   end
   ```

   Здесь главная программа **test** вообще не использует **common**-блок, но вызывает две подпрограммы **sub1** и **sub2**, которые массив из пяти элементов, расположенный в неименованном **common**-блоке, именуют по разному **vect** и **ssss**.

2.2.3 Некоторые рекомендации по использованию оператора common.

1. В ФОРТРАНе-95 есть альтернативные оператору **common** и более безопасные способы использования одной и той же области оперативной памяти разными программными единицами (см. параграф **2.3**). Поэтому пользуемся оператором **common** только в том случае, если связаны синтаксисом ФОРТРАНа-77.

2. В ФОРТРАНе-77 **глобальны** только имена процедур и **common**-блоков. Именно поэтому в любой внешней единице компиляции ФОРТРАНа имеем право обращаться к любой процедуре, подсоединенной компоновщиком, и включать в любую из них **common**-блоки. Поэтому же имя **common**-блока должно отличаться от любого другого глобального имени (то есть имени единицы компиляции или другого **common**-блока). Например,

   ```
   program tstcom1       ! Здесь par - имя common-блока. Оно не должно
   common / par/  a      ! совпадать ни с именем подпрограммы,
   a=3.0                 !           ни с именем другого - common-блока.
   call b                !  Результат пропуска этой программы на
   end                   !  компиляторе
   subroutine b          !               g77:    a=  3.0
   common / par /  a     !          gfortran:    a=  3.000000
   write(*,*) ' a=',a    !
   end
   ```
   ```

```
program tstcom1a      ! Если назвать подпрограмму именем par, то
common / par/  a      ! gfortran сообщит:
a=3.0                 ! /tmp/ccpcscRL.s: Assembler messages:
call par              !  Error: symbol ‘par_’ is already defined
end                   !
subroutine par        !
common / par /  a; write(*,*) ’ a=’,a
end
```

```
program tstcom1a    ! Имя common-блока может совпадать с именем
common / par/  par  ! локального объекта программной единицы.
par=3.0             ! Например, в список элементов common-блока
call b              ! с именем par можно включить и переменную
end                 ! с именем par (удобно ли только это?).
subroutine b        !
common / par /  a;  write(*,*) ’ a=’,a  !
end
```

3. Используем именованные **common**-блоки.

4. Переменные и массивы **common**-блока располагаются в оперативной памяти в том порядке, в котором они указаны в операторе **common**. Поэтому надо:

   (a) **соблюдать соответствие типов и длин** элементов, которые включены в список **common**-блока, используемого в разных программных единицах;

   (b) **не допускать** появления в одном **common**-блоке одинаковых имен среди списка его элементов;

   (c) **не допускать** в пределах одной программной единицы появления одного и того же имени в списках элементов двух различных **common**-блоков.

5. Располагаем имена переменных, входящих в список элементов **common**-блока, в порядке убывания длин их типов. Так, если список элементов **common**-блока **par** начинается с имени переменной типа **integer*1**, после которого указано имя переменной типа **real*4**, то при компиляции получим предупреждение:

```
 program tstcom2     ! $ gfortran tstcom2.f
 integer*1 i         ! In file tstcom2.f:3
 common /par/ i, a   !       common /par/ i, a
 a=1.3               !                      1
 i=2                 ! Warning: COMMON ’par’ at (1) requires
 write(*,*) ’ a=’,a, !          3 bytes of padding at start
>           ’  i=’,i ! т.е. перед a нужны 3 грунтовочных байта
 end                 !                    (выравнивающих)
```

   Вообще говоря, рекомендация о расположении переменных списка **common**-блока в порядке убывания их длин – не строго обязательна. Просто ее соблюдение не допустит случайного (или по незнанию) нарушения **правила выравнивания границ**, по которому адрес переменной, включаемой в **common**-блок, должен быть кратен длине переменной.

6. Для задания начальных значений элементам списков именованных **common**-блоков используется особая программная единица с именем **BLOCK DATA**. Ее особенность состоит в том, что она никогда не вызывается – данные, помещаемые в нее используются на этапе компиляции. Подробно о правилах ее описания и использования можно прочитать, например, в [4].

7. В ФОРТРАНе-77 имеется оператор **EQUIVALENCE**, который нескольким переменным одной единицы компиляции сопоставляет одну и ту же область оперативной памяти, то есть позволяет именовать одну и ту же ячейку памяти разными именами. Так, если при разработке разных частей одной подпрограммы три ее соавтора, забыв договориться об имени переменной для скорости, использовали каждый соответственно обозначения **v**, **velo** и **velocity**, то оператор **equivalence (v, velo, velocity)** позволял каждому из соавторов не переобозначать идентификатор скорости в своей части.

   Основное достоинство и одновременно недостаток оператора **equivalence**: в список группы эквивалентности могут входить имена переменных разных типов, что допускает без вызова каких-либо встроеных функций использование содержимого одной и той же ячейки памяти в качестве значений разных типов. Например, **ASCII**-код литеры, введенной в переменную **a** типа **character*1** при наличии оператора **equivalence** можно получить так:

```
program tsteqv
integer*1   i               ! Оператор equivalence предоставляет
character*1 a               ! возможность программе трактовать
equivalence (a,i)           ! одно и то же данное как значения
write(*,*) 'введи символ'   ! разных типов: a - character*1
read (*,*) a                !               i - integer*1
write(*,*) ' a=',a, ' i=',i  !
write(*,*) ' achar(i)=',achar(i), ' iachar(a)=', iachar(a)
write(*,*) ' char (i)=', char(i), ' ichar (a)=', ichar (a)
end
```

   Недостаток: подобное использование оператора **equivalence** не может быть формальным, то есть необходимо помнить, например, что тип **integer*1** трактует единичку, стоящую в старшем разряде байта, как знак числа, и поэтому символы, с кодами большими 127 будут отрицательными; при употреблении типа **integer*2** обязательно потребуется выяснить какой байт двубайтового целого попадает на однобайтовый символ. Короче предоставляется масса возможностей просчитаться, не говоря уже о том, что на компьютере с другой системой команд все выясненное может оказаться иным. Особую осторожность следует проявлять при включении в группы эквивалентности оператора **equivalence** переменных, входящих в **common**-блок: необходимо следить, чтобы оператор **equivalence** не *разрывал* **common**-области, например, сопоставляя следующим друг за другом переменным, входящим в нее, *несоседние* элементы массива.

   Вывод: Стараемся не использовать оператор EQUIVALENCE.

2.2.4 Информация к размышлению.

В ФОРТРАНе-77 есть два способа подачи данных в подпрограмму или функцию: через список формальных аргументов и через **common**-блок.

1. При небольшом числе аргументов функции удобен первый способ (через список формальных аргументов), так как при обращении проще и нагляднее соблюсти соответствие между формальными и фактическими параметрами. Поэтому при отсутствии объективно обоснованных причин подачи данного через **common**-блок используем первый способ, то есть передачу данных через список формальных аргументов.

2. Передача через данных **common**-блок – объективно обоснована, когда передача через список формальных аргументов невыгодна (или даже недопустима) в силу каких-то причин. Пример: необходимость использования имени функции (или подпрограммы), в качестве фактического аргумента некоторой другой подпрограммы, которая по синтаксису требует вызова первой с одним аргументом, хотя по существу дела эта первая зависит от нескольких (см. случай из пунктов **2.1-2.2**).

3. Иногда бывает разумным сочетать оба способа передачи, распределяя данные между списком формальных аргументов и **common**-блоками. При этом необходимость использования последних определяется в значительной мере не столько синтаксисом языка, сколько опытом и навыками программиста.

4. **Помним, что современный ФОРТРАН предоставляет более безопасные чем common-блок способ передачи дополнительных данных в подпрограммы, а именно:**

   (a) Передача параметров через модуль (см. пункты 2.3.1, 2.3.2).

   (b) Передача параметров внутренним функциям (см. пункт 2.6).

   Правда, эти альтернативные варианты не проходят на ФОРТРАНе-77. В то же время они обеспечивают более высокую надежность программирования в том смысле, что при их использовании невозможно в принципе совершить ошибки, которые могут встретиться при неграмотном, невнимательном или необдуманном применении оператора **common**.

   С указанными альтернативами ФОРТРАНа-95 мы уже знакомились в первой главе (см. пункты **1.2.2**, **1.2.3** первой части курса). Там они использовались для **передачи интерфейса процедур**.

   Здесь же их можно использовать для **передачи процедуре дополнительных параметров**, которые по каким-то обстоятельствам не хочется передавать через список формальных аргументов. В **1.2.3** уже отмечалось, что передача параметров через **модуль** позволяет вообще обойтись без **common**-блоков. С другой стороны можно, описав функции внутренними, использовать переменные вызывающей их программной единицы, как глобальные.

2.3 Альтернативы ФОРТРАНа-95 оператору COMMON.

2.3.1 Передача параметров через модуль (1-й вариант).

Оформим расчёт подинтегральных выражений внешними функциями

```
function f(x)        !                                      Файл fun.f
use modpar           ! Необходимые дополнительные параметры
implicit none        ! каждая  из  описанных здесь функций
real f, x            ! импортирует из модуля modpar, который
f=x+p                ! подключается оператором use modpar.
end                  ! Заметим, что подключение происходит
function g(x)        ! только к функции, содержащей оператор USE.
use modpar
implicit none        ! Полезно проследить за работой программы
real g, x            ! при комментировании в функции g(x)
g=x*(x+p)+q          ! операторов
end                  !             use modpar  и   implicit none
function w(x)        !
use modpar           ! и задании некоторых опций работы компилятора
implicit none        ! gfortran, именно:
real w, x            !                  либо -fimplicit-none,
w=r*cos(p*x+q)       !                  либо -Wuninitialized и -O1
end
```

используя для передачи дополнительных парметров вместо **common**-блока **модуль** ФОРТРАНа-95. Назовем модуль, например, именем **modpar**. Его исходный текст:

```
module modpar
implicit none
real p, q, r
end module modpar
```

1. Каждая из описанных в файле **fun.f** функций импортирует дополнительные параметры из модуля **modpar**, подключаемого оператором **use**.

2. Если, например, убрать из функции **g(x)** строку **use modpar**, то при компиляции получим сообщение о том, что параметры **p** и **q** неявно не описаны. Иначе говоря, **p** и **q** восприняты **g(x)** как имена ее внутренних локальных переменных, которые в силу отключения правила умолчания оказались ей неизвестны.

3. Очень опасная ситуация возникнет, если не включим в **g(x)** оба оператора (и **use modpar**, и **implicit none**). Вместо переменных **p** и **q** из модуля, функция **g(x)** *возьмет* значения своих одноименных внутренних переменных, которые неопределены. Можно довольно долго полагать результат интегрирования функции **g(x)** верным, не замечая две упомянутые выше неточности. Хорошо еще, что в главной программе мы намерены выводить значения интегралов, полученные по формуле Ньютона-Лейбница.

4. Наряду с модулем **modpar** будем использовать и модуль **quadra** (с процедурами численного интегрирования). Его исходный текст:

```
module quadra                   !                    Файл quadra.for:
contains                        !
  function trap1(f,a,b,n);      ! Функция trap1(f,a,b,n) вычисляет
  implicit none                 ! интеграл от функции f(x) по формуле
  real trap1, a, b; integer n   ! трапеций на отрезке [a,b], разделенном
  integer i                     ! на n равных подучастков.
  real h, x, s, f
  h=(b-a)/n
  s=(f(a)+f(b))/2
  do i=2,n
    x=a+(i-1)*h
    s=s+f(x)
  enddo
  trap1=s*h
  end function trap1
end module quadra
```

5. Исходный текст главной программы, использующей модули **modpar** и **quadra**:

```
program tsquadout
use quadra
use modpar
implicit none
real af / 1.0 /, bf / 10.0 /  ! Параметры p, q, r описывать не надо.
real ag / 0.0 /, bg /  2.0 /  ! Они со своими атрибутами импортируются
real aw / 0.5 /, bw /  1.5 /  ! главной программой из модуля modpar
real rf1, rg1, rw1, r1        ! так же, как trap1 из модуля quadra.
real rf0, rg0, rw0, r0
integer nf, ng, nw
interface                                          ! Описание
 function f(x); real f,x; end function f           ! интерфейса
 function g(x); real g,x; end function g           ! подинтегральных
 function w(x); real w,x; end function w           ! функций
end interface
write(*,*) 'введи nf, ng, nw'                      ! Ввод и печать:
read (*,*) nf, ng, nw;                             ! числа участков
write(*,*) ' nf=',nf, ' ng=',ng, ' nw=',nw         !        дробления,
write(*,*) 'введи параметры p, q, r'               ! дополнительных
read (*,*) p, q, r    ;                            !       параметров.
write(*,*) '  p=', p, '  q=',q, '  r=',r          !
rf0=(bf-af)*(0.5*(bf+af)+p)                        ! Контрольный
rg0=(bg-ag)*((bg*bg+ag*bg+ag*ag)/3+0.5*p*(bg+ag)+q) ! расчет по
rw0=r*(sin(p*bw+q)-sin(p*aw+q))/p;                 ! Ньютону-Лейбницу
r0=rf0+rg0+rw0                                     !
rf1=trap1(f,af,bf,nf)                    ! Численный  расчет  через
rg1=trap1(g,ag,bg,ng)                    ! вызов функции trap1,
rw1=trap1(w,aw,bw,nw)                    ! хранимой в модуле quadra.
r1=rf1+rg1+rw1
write(*,'(4e15.7)') rf0, rg0, rw0, r0 ! Результат по Ньютону-Лейбницу.
write(*,'(4e15.7)') rf1, rg1, rw1, r1 ! -"--"--"- численного расчёта.
end
```

6. В данном примере параметры, передаваемые через модуль, известны программным единицам, их использующих, именно под теми именами, которые описаны в модуле. Оператор же **common** позволяет из разных единиц компиляции обращатся к одной и той же области оперативной памяти под разными именами.

7. Оператор **use** тоже предоставляет подобную возможность. Более того, оператор **use** допускает **ограничение** доступа, единицы компиляции, использующей его, к описанным в нем объектам (подробнее см. далее пункт **2.4**), что является важным преимуществом **модуля** перед **common**-блоком, не говоря уже о правиле выравнивания границ, которое в случае модуля неактуально.

8. При компиляции на **gfortran** полезно подстраховаться опцией **-fimplicit-none**, выводящей предупреждение об использовании неявно объявленных переменных или функций. В этом случае не заметить в функции **g(x)** опасный одновременный пропуск сразу двух операторов (и **use modpar**, и **implicit none**), о котором шла речь пункте *3*, фактически невозможно.

9. С учётом пункта *8* можно предложить для работы с нашей программой следующие варианты **make**-файла:

```
comp=gfortran -c -fimplicit-none                # первый вариант
main :   quadra.o modpar.o tsquadrout.o fun.o
 gfortran quadra.o modpar.o tsquadrout.o fun.o -o main
fun.o:   fun.f
 $(comp) fun.f
quadra.mod quadra.o :  quadra.f
 $(comp) quadra.f
modpar.mod modpar.o :  modpar.f
 $(comp) modpar.f
tsquadrout.o: tsquadrout.f modpar.mod quadra.mod
 $(comp) tsquadrout.f
clear :
  rm *.o main *.mod
result : main
 ./main


  comp:=gfortran                            # второй вариант
   opt:= -c -fimplicit-none
   pattern:=*.f
source:=$(wildcard $(pattern))
   obj:=$(patsubst %.f, %.o, $(source))
main :   $(obj)
 $(comp) $^ -o $@
 %.mod %.o : %.f
 $(comp) $(opt) $<
 tsquadrout.o : tsquadrout.mod modpar.mod
     fun.o    : modpar.mod
clear :
  rm *.o main *.mod
result : main
 ./main
```

При отсутствии в функции **g(x)** описаний **implicit none** и **use modpar** любой из приведённых **make**-файлов, в частности, получит:

```
In file fun.f:11
                g=x*(x+p)+q
                       1    Error: Symbol 'p' at (1) has no IMPLICIT type
make: *** [fun.o] Ошибка 1
```

Игнорирование же опции **-fimplicit-none** приведёт к явно неверному результату при вызове **trap1(g,a,b,n)**:

```
   p=  0.6000000       q=  0.7000000       r=  0.8000000
  0.5490000E+02  0.5266667E+01  0.2108035E+00  0.6037747E+02
  0.5489999E+02  0.2666668E+01  0.2108036E+00  0.5777747E+02
```

10. Альтернативный вариант подстраховки аналогичной ситуации (отсутствия в программной единице оператора **implicit none** и оператора **use** , которые на самом деле нужны, — включение опций компиляции **-Wuninitalized** и **-O1** (без опции оптимизации **-O1** нужная нам опция **-Wunitialized**, информирующая о наличии неинициализированных автоматических переменных, не работает). При использовании указанных опций компиляции получаем:

```
gfortran -c -Wuninitialized -O1 fun.f
fun.f: In function 'g':
fun.f:11: warning: 'p' is used uninitialized in this function
fun.f:11: warning: 'q' is used uninitialized in this function
gfortran quadra.o modpar.o tsquadrout.o fun.o -o main
./main
 введи nf, ng, nw
1000   1000  1000
  nf=        1000  ng=        1000  nw=        1000
 введи параметры p, q, r
0.6    0.7    0.8
    p=  0.6000000       q=  0.7000000       r=  0.8000000
   0.5490000E+02  0.5266667E+01  0.2108035E+00  0.6037747E+02
   0.5489999E+02            NaN  0.2108036E+00            NaN
```

Видим, что включение опций наряду с предупреждающими сообщениями компиляции (допустим их не заметили) при пропуске программы привело к получению в качестве результата численного интегрирования **g(x)** значение **NaN** (*Not a Number*), которое заметить, безусловно, легче чем число *0.5266667E+01*, маскирующееся под правильный результат.

11. Вместо опции **-Wuninitialized**, нацеленную именно на отлов фактов использования неинициализированных переменных, можно включать опцию **-Wall**, которая объединяет опции **-Wuninitialized** и **-Wunused**. Последняя нацелена на вывод предупреждения о факте неиспользования переменной, указанной в описаниях. Не забываем при включении **-Wall** и об опции **-O1**.

2.3.2 Передача параметров через модуль (2-й вариант).

Размещение в модуле только дополнительных параметров (без описания в нём функций, использующих их) вряд ли удобно. Приведем пример их размещения в одном модуле:

```
module myfunpar       !                          Файл myfunpar.f
implicit none         !                          ===============
real p, q, r
contains
  function f(x)       !
  implicit none       ! Каждой  из  описанных здесь функций
  real f, x           ! доступны переменные p, q и r, которые
  f=x+p               ! описаны в разделе описания переменных этого
  end function f      ! модуля.
  function g(x)
  implicit none
  real g, x
  g=x*(x+p)+q
  end function g
  function w(x)
  implicit none
  real w, x
  w=r*cos(p*x+q)
  end function w
end module myfunpar

program tsquadra2             !               Файл tsquadra2.f
use quadra                    !               ================
use myfunpar                  ! Указывать интерфейс модульных функций f,
implicit none                 ! g, w в отличие от внешних здесь НЕ НАДО,
real af / 1.0 /, bf / 10.0 /  ! поскольку все импортируется из модуля
real ag / 0.0 /, bg /  2.0 /  ! myfunpar (в частности, и то, что  в
real aw / 0.5 /, bw /  1.5 /  ! операторах вызова trap1 первый параметр
real rf1, rg1, rw1, r1        ! - имя функции). Поэтому оператор
real rf0, rg0, rw0, r0        ! external излишен, хотя и возможен.
integer nf, ng, nw
write(*,*) 'введи nf, ng, nw'  ! Ввод и контрольная печать:
read (*,*) nf, ng, nw;                                ! числа участков
write(*,*) ' nf=',nf, ' ng=',ng, ' nw=',nw            !        дробления,
write(*,*) 'введи параметры p, q, r'                  ! дополнительных
read (*,*) p, q, r    ;                               !       параметров.
write(*,*) '  p=', p, '  q=',q, '  r=',r              !
rf0=(bf-af)*(0.5*(bf+af)+p)                           ! Контрольный
rg0=(bg-ag)*((bg*bg+ag*bg+ag*ag)/3+0.5*p*(bg+ag)+q)   ! расчет по
rw0=r*(sin(p*bw+q)-sin(p*aw+q))/p;                    ! Ньютону-Лейбницу
r0=rf0+rg0+rw0                                        !
rf1=trap1(f,af,bf,nf)                          ! Численный  расчет  через
rg1=trap1(g,ag,bg,ng)                          ! вызов функции trap1,
rw1=trap1(w,aw,bw,nw)                          ! хранимой в модуле qudra.
r1=rf1+rg1+rw1
write(*,'(4e15.7)') rf0, rg0, rw0, r0          ! Вывод результатов.
write(*,'(4e15.7)') rf1, rg1, rw1, r1
end
```

2.4 Оператор USE.

До сих пор оператор **USE** использовали в самой простой форме. Поэтому может показаться, что оператор **common** имеет некоторое преимущество перед модулями.

1. В разных программных единицах соответствующие элементы, включаемые в один и тот же **common**-блок, могут иметь разные имена. Так в файле с набором подинтегральных функций из пункта **2.2.1** функция **w(x)** имела описание **common**-блока: **common /par/ pp, qq, rr**, хотя в главной программе элементы, входящие в этот **common**-блок обозначались иначе: **common /par/ p, q, r**. Разные люди, работая над разными единицами компиляции одного проекта, вполне могли для обозначения переменных нужного **common**-блока использовать разные имена, сохраняя лишь их очерёдность и соответствие типов.

   На самом деле оператор **use** позволяет изменять имена параметров, передаваемых через модуль. Например, если при написании функции **w(x)**, использующей параметры, передаваемые через модуль **modpar**, программист вместо имен **p**, **q** и **r** использовал имена **pp**, **qq** и **rr**, то нужное подсоединение модуля достигается оператором

   ```
   use modpar, pp=>p, qq=>q, rr=>r
   ```

   Здесь после имени модуля идет так называемый список переименований. Этот список указывает соответствие между новыми и изначальными именами параметров, которые импортируются программой. Короче, файл с главной программой из пункта **2.3.2** и сам модуль **modpar** остаются неизменными, а файл **fun.f** принимает вид:

   ```
   function f(x)                    !                              Файл fun.f
   use modpar                       ! Необходимые дополнительные параметры
   implicit none                    ! функции f(x) и g(x) импортируют из
   real f, x                        ! из модуля modpar, под именами p, q и r,
   f=x+p                            ! опредедленными в модуле modpar.
   end                              !
   function g(x)
   use modpar
   implicit none
   real g, x
   g=x*(x+p)+q
   end
   function w(x)
   use modpar, qq=>q, rr=>r, pp=>p  ! Функция  w(x)  импортирует  те же
   implicit none                    ! параметры под именами, указанными в
   real w, x                        ! списке переименований оператора USE
   w=rr*cos(pp*x+qq)                ! (соответственно pp, qq и rr).
   end
   ```

   Так что, оператор **use** никак не уступает по возможностям оператору **common**. Более того, указание оператора **use** о переименовании может быть дано в произвольном порядке, в то время как оператор **common** требует *строжайшего*

соблюдения соответствия очередности типов и имен, указываемых в **common**-списке. Другими словами, оператор **use** в принципе не позволит совершить ошибку, довольно часто встречающуся при использовании **common**-блока.

2. Оператор **use** имеет еще одну форму подсоединения модуля к единице компиляции. Это так называемое **ONLY**-подсоединение (**ONLY**-доступ), что даёт **100%** гарантию использования только тех параметров модуля, которые указаны в операторе **use**, т.е. обладает свойством, которого нет у оператора **common**.

   Осмыслим ситуацию. Когда целое семейство функций использует одни и те же параметры, то довольно естественно и функции, и параметры описать в одном модуле. Именно такой пример разобран в пункте **2.3.3**.

   Однако нередки случаи, когда функция использует не все параметры, а лишь некоторые из них. При этом, используя **common**-блок, можно в качестве какой-то рабочей переменной ошибочно указать имя элемента **common**-блока. Аналогичное возможно и при использовании модуля, подключаемого оператором **use**, если не знать об **ONLY**-доступе, который позволяет единице компиляции *видеть* **только те параметры**, которые укажет программист при подсоединении модуля в **ONLY**-списке оператора **use**, полностью устраняя несанкционированный доступ к остальным параметрам. Обратимся к предыдущему примеру.

   Пусть функция **f(x)** использует не только дополнительный параметр **p**, описанный в разделе описаний модуля, но и некоторую локальную рабочую переменную, для которой в силу каких-то причин выбрано имя **q**, хотя никакого отношения эта переменная к одноименному параметру модуля не имеет. Укажем оператору **use** при подсоединении модуля, что функция **f(x)** должна видеть в нем только параметр **p**:

```
function f(x)        !                          Файл fun.f
use modpar, ONLY: p; ! При таком подсоединении модуля функция f(x)
implicit none        ! НЕ ВИДИТ параметры q и r. В нее импортируется
real f, x, q, r      ! только параметр p. Поэтому имена q и r можно
r=0.5*x              ! использовать в качестве внутренних локальных
q=2*r+p              ! переменных, не боясь "испортить" одноименные
f=q                  ! параметры модуля modpar.
end
function g(x)
use modpar, ONLY: bbb=>q, aaa=>p !<--= Здесь дается указание, что
implicit none                    ! после подсоединения модуля функция
real g, x, r                     ! g(x)должна "видеть" только два из
r=x+aaa                          ! из трех описанных в нем параметров,
g=x*r+bbb                        ! причем параметр p под именем aaa,
end                              !       а параметр q под именем bbb.
function w(x)            !  При этом получаем возможность имя r
use modpar               !  использовать в качестве имени локальной
implicit none            !  переменной функции g(x).
real w, x
w=r*cos(p*x+q)
end
```

Подробнее об операторе **use** см., например, [4].

2.5 Атрибуты PUBLIC и PRIVATE.

2.5.1 Уяснение ситуации.

Часто в модуль приходится помещать несколько функций, решающих одну задачу разными методами. Так в модуле численного интегрирования могут находиться функции расчета квадратурной суммы по разным квадратурным формулам (трапеций, прямоугольников, Симпсона и др.; см., например, **Приложение 7**).

С одной стороны у каждой из этих функций есть имя и формальные параметры. С другой — каждая из этих функций имеет свои внутренние локальные переменные для обозначения текущих значений независимой переменной интегрирования, накапливаемой квадратурной суммы, шага дискретизации, номера очередного используемого узла. Например, у функции **trap1**

```
function trap1(f,a,b,n)    !                   Файл trap1.f
implicit none              ! Функция trap1(f,a,b,n) реализует алгоритм
integer n, i               ! расчета по формуле трапеций интеграла по
real a, b, trap1, f        ! n подучасткам равномерного дробления [a,b]
real h, x, s               ! от функции f(x), имя которой передаётся
h=(b-a)/n                  ! trap1) передается  trap1 в качестве
s=(f(a)+f(b))/2            ! первого аргумента.
do i=2,n
  x=a+(i-1)*h
  s=s+f(x)
enddo
trap1=s*h
end
```

эти внутренние локальные переменные (см. также текст модуля **quadra** из пункта **2.3.1**) обозначены **x**, **s**, **h** и **i** соответственно и описываются внутри **trap1**. Аналогичные внутренние переменные встречаются и в остальных функциях расчета квадратурной суммы, так что их придётся описывать в каждой функции, хотя типы и имена у них такие же как и в случае **trap1**.

Поэтому возникает мысль: сделать эти переменные **глобальными** по отношению к функциям, использующим их, то есть описать их в модуле в разделе описания переменных. Правда, при этом указанные переменные станут доступны и единице компиляции, подключающей модуль, что чрезвычайно **НЕУДОБНО** – программа, вызывающая функцию, может использовать эти имена в совершенно ином смысле.

Таким образом, с одной стороны *хочется* за счет модульной организации уменьшить исходный код каждой модульной функции, а с другой *не хочется* конфликта с упомянутыми переменными программы, подсоединяющей модуль.

Разрешение проблемы достигается посредством атрибута **private**, который будучи указан при описании модульной переменной как раз и снабжает ее указанным свойством: переменная доступна функциям модуля, но не доступна программе, подключающей модуль:

```
real,    private :: x, h, s
integer, private :: i
```

2.5.2 Примеры использования атрибута PRIVATE.

Для демонстрации сказанного смоделируем ситуацию. Создадим модуль **quadra1**, в котором опишем две функции расчета квадратурной суммы **trap1** (по формуле трапеций) и **rectan** (по формуле средних прямоугольников), перенеся из их текста описания упомянутых выше локальных переменных **x**, **s**, **h** и **i** в раздел описаний переменных модуля, и снабдив эти описания служебным словом **private**.

```
module quadra1                 ! Модуль с функциями     Файл quadra1.for
                               !   и подпрограммами     ================
                               ! численного интегрирования.
real,    private :: h, x, s    !<- Эти переменные доступны только модуль-
integer, private :: i          !<- ным функциям и не экспортируются
                               !   в программу, подключающую модуль.
contains
function trap1(f,a,b,n);       ! Функция trap1(f,a,b,n) вычисляет
  implicit none                ! интеграл от функции f(x) по формуле
  real trap1, a, b; integer n  ! трапеций на отрезке [a,b], разделенном
  real, external :: f          ! на n равных подучастков.
  h=(b-a)/n
  s=(f(a)+f(b))/2
  do i=2,n
    x=a+(i-1)*h
    s=s+f(x)
  enddo
  trap1=s*h
  end function trap1
  function rectan(f,a,b,n);    ! Функция rectan(f,a,b,n) вычисляет
  implicit none                ! интеграл от функции f(x) по формуле
  real rectan, a, b; integer n ! средних прямоугольников на отрезке
  real, external :: f          ! [a,b], разделенном на n равных
  h=(b-a)/n                    !  подучастков.
  s=0
  do i=1,n
    x=a+(i-0.5)*h
    s=s+f(x)
  enddo
  rectan=s*h
  end function rectan
end module quadra1
```

Тело каждой функции стало короче на описание локальных рабочих переменных. В качестве тестирующей программы используем слегка модифицированную программу из пункта **2.3.1**, к которой подключается модуль. Добавим в нее:

1. операторы соответствующих вызовов функции **rectan**;
2. операторы вывода соответствующих результатов;
3. закомментированный вывод переменных **x**, **s**, **h** и **i**, дабы убедиться после раскомментирования, что одноименных переменных модуля программа *в упор не видит* из-за действия атрибута **private**.

```
      program tsquadra3              !               Файл tsquadra3.f
      use quadra1                    !               ================
      use myfunpar
      implicit none
      real af / 1.0 /, bf / 10.0 /
      real ag / 0.0 /, bg /  2.0 /
      real aw / 0.5 /, bw /  1.5 /
      real rf0, rg0, rw0, r0
      real rf1, rg1, rw1, r1
      real rf2, rg2, rw2, r2
      integer nf, ng, nw
      write(*,*) 'введи nf, ng, nw'                       ! Ввод и печать:
      read (*,*) nf, ng, nw;                              ! числа участков
      write(*,*) ' nf=',nf, ' ng=',ng, ' nw=',nw          !        дробления,
      write(*,*) 'введи параметры p, q, r'                ! дополнительных
      read (*,*) p, q, r    ;                             !       параметров.
      write(*,*) '  p=', p, '  q=',q, '  r=',r           !
      rf0=(bf-af)*(0.5*(bf+af)+p)                         ! Контрольный
      rg0=(bg-ag)*((bg*bg+ag*bg+ag*ag)/3+0.5*p*(bg+ag)+q) ! расчет по
      rw0=r*(sin(p*bw+q)-sin(p*aw+q))/p;                  ! Ньютону-Лейбницу
      r0=rf0+rg0+rw0                                      !
      rf1=trap1(f,af,bf,nf); rg1=trap1(g,ag,bg,ng)   ! Расчёт интеграла через
      rw1=trap1(w,aw,bw,nw); r1=rf1+rg1+rw1          ! вызов функции trap1,
      rf2=rectan(f,af,bf,nf); rg2=rectan(g,ag,bg,ng) ! Расчёт интеграла через
      rw2=rectan(w,aw,bw,nw); r2=rf2+rg2+rw2         ! вызов функции rectan,
      write(*,'(4e15.7)') rf0, rg0, rw0, r0          ! Вывод результатов.
      write(*,'(4e15.7)') rf1, rg1, rw1, r1
      write(*,'(4e15.7)') rf2, rg2, rw2, r2
c      write(*,*) x, s, h, i    ! При раскомментировании получим сообщение
                               ! об отсутствии описания этих переменных,
                  ! т.к. одноимённые модульные наделены атрибутом private
      end
```

1. Результат пропуска:

```
./main
 введи nf, ng, nw
 1
 1000
 1000
   nf=           1  ng=        1000  nw=        1000
  введи параметры p, q, r
0.6
0.7
0.8
   p=  0.6000000       q=  0.7000000       r=  0.8000000
  0.5490000E+02  0.5266667E+01  0.2108035E+00  0.6037747E+02
  0.5490000E+02  0.5266668E+01  0.2108036E+00  0.6037747E+02
  0.5490000E+02  0.5266666E+01  0.2108034E+00  0.6037747E+02
```

2. Если оператор **private** задать без списка объектов модуля, то его действие распространяется на все объекты модуля. Например, если подсоединим к нашей главной программе не модуль **quadra1**, а модуль **quadra2**:

```
module quadra2   ! Подобное применение атрибута PRIVATE делает
real   h, x, s   ! все объекты модуля недоступными, подключающей
private          ! его программе: и переменные h, x, s и i,
integer i        ! и сами модульные функции trap1 и rectan.
contains
        Содержимое раздела реализации процедур полностью совпадает
        с соответствующим разделом модуля quadra1.
end module quadra2
```

то при компиляции получим массу сообщений типа:

```
 In file tspriv.f:23            ! Несмотря на то, что f - внутренняя функция,
     rf1=trap1(f,af,bf,nf)      ! при компиляции вызова trap1 - этот факт еще
               1                                        ! неизвестен.
Error: Symbol 'f' at (1) has no IMPLICIT type           ! Из-за PRIVATE по всем
 In file tspriv.f:24                                    ! объектам модуля quadra2
     rg1=trap1(g,ag,bg,ng)                              ! программе неизвестно
        1                                               ! даже, что и сама-то
Error: Function 'trap1' at (1) has no IMPLICIT type     ! trap1 - функция
```

3. Альтернативой атрибуту **private** служит атрибут **public**. По умолчанию все объекты создаваемого **модуля** имеют атрибут **public**, если не используется **private**. Эти атрибуты могут характеризовать не только имена переменных, но и констант, процедур, **namelist**-групп, **производных типов** и родовых описаний. Например,

```
module mycomp        !                                    Файл mycomp.f
type comp            ! Описание имени типа comp с двумя
  real re, im        ! полям типа real: re - для хранения вещественной
end type comp        ! части; im - для мнимой.
contains
  subroutine wrtcomp(x)  ! Подпрограмма печати переменной типа comp.
  implicit none          ! Заметим, что имя поля от имени переменной
  type(comp) x           ! отделяетя значоком %, который в C
  write(*,*) x%re, x%im  ! обозначает операцию нахождения остатка.
end subroutine wrtcomp
  subroutine addcomp(a,b,c)  ! Подпрограмма сложения двух переменных
  implicit none              ! типа comp
  type(comp) a, b, c
  c%re=a%re+b%re; c%im=a%im+b%im
  end subroutine addcomp
end module mycomp


program tstcomp
use mycomp
implicit none
type(comp) a, b, c                    ! Имя типа comp известно главной
a%re=3.0; a%im=4.0;  call wrtcomp(a); ! программе, так импортируется
b%re=2.0; b%im=5.0;  call wrtcomp(b); ! из модуля, в котором по
call addcomp(a,b,c); call wrtcomp(c); ! умолчанию имеет атрибут PUBLIC
end
```

2.6 Передача глобальных параметров внутренним функциям.

Модифицируем программу из пункта **2.2.1**: опишем алгоритмы расчёта подинтегральных функций **f(x)**, **g(x)** и **w(x)** внутренними по отношению к главной программе функциями. Тогда любая переменная главной программы будет видна любой внутренней функции.

Если версия компилятора допускает использование имени внутренней функции в качестве фактического параметра (по [1] стандарт ФОРТРАНа-95 такое исключает), то вполне работоспособной оказывается программа:

```
program tsttrap3
implicit none                    !
real af / 1.0 /, bf / 10.0 /     ! Пределы интегрирования функции f,
real ag / 0.0 /, bg /  2.0 /     ! --"--"--"--"--"--"--"--"--"--  g,
real aw / 0.5 /, bw /  1.5 /     ! --"--"--"--"--"--"--"--"--"--  w.
real rf1, rg1, rw1, r1           ! Результаты численного интегрирования.
real rf0, rg0, rw0, r0           ! --"--  аналитического интегрирования.
integer nf, ng, nw               ! Количества участков дробления.
real p, q, r                     ! Дополнительные параметры.
real trap1
write(*,*) 'введи nf, ng, nw'
read (*,*) nf, ng, nw;                              ! Ввод числа участков
write(*,*) ' nf=',nf, ' ng=',ng, ' nw=',nw          !           дробления.
write(*,*) 'введи параметры p, q, r'
read (*,*) p, q, r    ;                             ! Ввод дополнительных
write(*,*) '  p=', p, '  q=',q, '  r=',r           !          параметров.
rf0=(bf-af)*(0.5*(bf+af)+p)                           ! Контрольный
rg0=(bg-ag)*((bg*bg+ag*bg+ag*ag)/3+0.5*p*(bg+ag)+q) ! расчет по
rw0=r*(sin(p*bw+q)-sin(p*aw+q))/p;                    ! Ньютону-Лейбницу
r0=rf0+rg0+rw0
rf1=trap1(f,af,bf,nf)                          ! Численный  расчет  через
rg1=trap1(g,ag,bg,ng)                          ! вызов функции численного
rw1=trap1(w,aw,bw,nw)                          ! интегрирования trap1.
r1=rf1+rg1+rw1
write(*,'(4e15.7)') rf0, rg0, rw0, r0      ! Вывод результатов.
write(*,'(4e15.7)') rf1, rg1, rw1, r1      !
contains                ! Раздел описания трех внутренних функций,
  function f(x)         ! которые вычисляют подинтегральные функции:
  implicit none         !
  real f, x             !  Первая зависит от одного дополнительного
  f=x+p                 !  параметра p: f(x)=x+p.
  end function f
  function g(x)         !  Вторая от двух p и q:
  implicit none         !                          g(x)=x^2+p*x+q,
  real g, x             !
  g=x*(x+p)+q           !
 end function g
 function w(x)          !  Третья от трех   r, p, q:
 implicit none          !
 real w, x              !
 w=r*cos(p*x+q)         !                          w(x)=r*cos(p*x+q)
 end function w
end
```

2.7 Моделирование common-блока или “module” в СИ

При переложении ФОРТРАН-программы на язык программирования СИ может возникнуть необходимость смоделировать единицу компиляции **module** или (в случае ФОРТРАНа-77) **common**-блока. Рассмотрим СИ-решение задачи из пункта **2.3.1** о численном расчёте трёх интегралов по формуле трапеций, когда требующиеся подинтегральным функциям **f(x)=x+p**, **g(x)=x*(x+p)+q** и **w(x)=r*sin(px+q)** дополнительные параметры **p**, **q** и **r** определяются как глобальные.

```
#include <stdio.h>                                          //  Файл main.c
double trap1(double (*)(double), double, double, int); // Описания
double f(double);                                           // прототипов
double g(double);                                           // функций.
double w(double);
double p, q, r;       // Описание глобальных дополнительных параметров
int main()
{ double af=1.0, bf=10.0, ag=0.0, bg=2.0, aw=0.5, bw=1.5;
  double rf1, rg1, rw1, r1, rf0, rg0, rw0, r0;
  int nf, ng, nw;
  printf("Input nf, ng, nw\n");                 // Ввод числа узлов
  scanf("%i %i %i",&nf, &ng, &nw);              // по первому, второму
  printf(" nf=%i  ng=%i  nw=%i\n",nf, ng, nw);  // и третьему промежуткам,
  printf("Input p, q, r\n");                    // и дополнительных пара-
  scanf("%lf %lf %lf",&p, &q, &r);              // метров подинтегральных
  printf("  p=%f   q=%f  r=%f\n", p, q, r);     // функций.
  rf0=(bf-af)*(0.5*(bf+af)+p);                          // Контрольный
  rg0=(bg-ag)*((bg*bg+ag*bg+ag*ag)/3+0.5*p*(bg+ag)+q); // расчет по
  rw0=r*(sin(p*bw+q)-sin(p*aw+q))/p;                    // Ньютону-Лейбницу
  r0=rf0+rg0+rw0;                                       //
  rf1=trap1(f,af,bf,nf);                        // Расчет  интегралов через
  rg1=trap1(g,ag,bg,ng);                        // вызов функции trap1
  rw1=trap1(w,aw,bw,nw);   r1=rf1+rg1+rw1;
  printf(" %15.7le %15.7le %15.7le %15.7le\n",rf0,rg0,rw0,r0);
  printf(" %15.7le %15.7le %15.7le %15.7le\n",rf1,rg1,rw1,r1);
  return 0;
}

double trap1(double (*f)(double), double a, double b, int n) // Файл
{ double s, x, h; int i;                                      // mytrap.c
  s=(f(a)+f(b))/2; h=(b-a)/n;
  for (i=1;i<n;i++) { x=a+h*i; s+=f(x); } return(s*h);
}

extern double p, q, r;                                  //  Файл myfun.c
double f(double x) { return x+p;}
double g(double x) { return x*(x+p)+q;}
double w(double x) { return r*cos(p*x+q);}
```

1. Здесь файл **myfun.c** содержит определения подинтегральных функций. Кроме того, через посредство модификатора **extern double p, q, r;** объявляется, что переменные **p**, **q** и **r**, имена которых встречаются в этом файле, определяются в какой-то другой единице компиляции, решающей поставленную задачу совместно с данной.

2. В данном случае такой другой единицей компиляции является файл, содержащий, в частности, и главную программу. Приведём пример результата работы соответствующего исполнимого кода:

```
Input nf, ng, nw
 nf=1000  ng=1000  nw=1000
Input p, q, r
  p=1.000000   q=2.000000  r=3.000000
   5.8500000e+01   8.6666667e+00  -2.8477661e+00   6.4318901e+01
   5.8500000e+01   8.6666680e+00  -2.8477659e+00   6.4318902e+01
```

3. Если объявление **extern double p, q, r;** не поместить в файл **myfun.c** (например, закомментировать его), то на этапе компиляции получим:

```
$ gcc main.c trap1.c myfun.c -lm
myfun.c: In function ‘f’:
myfun.c:2: error: ‘p’ undeclared (first use in this function)
myfun.c:2: error: (Each undeclared identifier is reported only once
myfun.c:2: error: for each function it appears in.) ...
```

4. Определение **double p, q, r;** можно и не включать в файл **main.c**. Достаточно его поместить в отдельный файл, например, с именем **mypar.c**. Правда, в этом случае необходимо расширить область видимости **main.c** так, чтобы нужные переменные из файла **mypar.c** были ей видны. Требуемое расширение области видимости **main.c** достигается посредством модификатора **extern** (аналогично тому как было сделано в **myfun.c**). Если единиц компиляции, которым требуется расширение области видимости на имена **p**, **q** и **r** из **mypar.c**, — много (или список, входящих в неё имён, велик), то есть шанс при очередном включении объявления **extern double p, q, r;** случайно допустить досадную опечатку. Во избежание подобного выгодно объявление **extern double p, q, r;** поместить в отдельный файл, назвав его, например, **mypar.h**, подключая нужное объявление через инструкцию препроцессора #**include “mypar.h”**.

5. Для сохранения мнемонической связи с ФОРТРАНом можно вместо имени **mypar.h** выбрать имя **common_mypar** или **use_mypar.mod**, что напомнит фортран-программисту, работающему на СИ, о подключении аналога фортрановского **common**-блока **mypar** или же единицы компиляции **module mypar**.

6. Таким образом, СИ-модель организации фортрановских **common**-блока или единицы компиляции **module** может выглядеть так:

```
                //                                      Файл  mypar.c
double p, q, r;  // СИ-модель common / mypar / p, q, r
                //       или module mypar


                      //                                  Файл mypar.h
extern double p, q, r; // Расширение области видимости на  p, q, r
                      // той единицы компиляции, в которой встретится
                      // это объявление.

#include <stdio.h>    //                                   Файл main.c
#include "mypar.h"    // Подключение расширения области видимости
#include "interface.h" // Подключение объявлений прототипов функций
int main()
{ double af=1.0, bf=10.0, ag=0.0, bg=2.0, aw=0.5, bw=1.5;
  double rf1, rg1, rw1, r1, rf0, rg0, rw0, r0;
  int nf, ng, nw;
  printf("Input nf, ng, nw\n");                 // Ввод числа узлов
  scanf("%i %i %i",&nf, &ng, &nw);              // по первому, второму
  printf(" nf=%i  ng=%i  nw=%i\n",nf, ng, nw);  // и третьему промежуткам,
  printf("Input p, q, r\n");                    // и дополнительных пара-
  scanf("%lf %lf %lf",&p, &q, &r);              // метров подинтегральных
  printf("  p=%f   q=%f  r=%f\n", p, q, r);     // функций.
  rf0=(bf-af)*(0.5*(bf+af)+p);                          // Контрольный
  rg0=(bg-ag)*((bg*bg+ag*bg+ag*ag)/3+0.5*p*(bg+ag)+q); // расчет по
  rw0=r*(sin(p*bw+q)-sin(p*aw+q))/p;                    // Ньютону-Лейбницу
  r0=rf0+rg0+rw0;                                       //
  rf1=trap1(f,af,bf,nf);                        // Расчет  интегралов через
  rg1=trap1(g,ag,bg,ng);                        // вызов функции trap1
  rw1=trap1(w,aw,bw,nw);   r1=rf1+rg1+rw1;
  printf(" %15.7le %15.7le %15.7le %15.7le\n",rf0,rg0,rw0,r0);
  printf(" %15.7le %15.7le %15.7le %15.7le\n",rf1,rg1,rw1,r1);
  return 0;
}

double trap1(double (*f)(double), double a, double b, int n) // Файл
{ double s, x, h; int i;                                     // mytrap.c
  s=(f(a)+f(b))/2; h=(b-a)/n;
  for (i=1;i<n;i++) { x=a+h*i; s+=f(x); } return(s*h);
}

#include "mypar.h"                                  //  Файл myfun.c
double f(double x) { return x+p;}
double g(double x) { return x*(x+p)+q;}
double w(double x) { return r*cos(p*x+q);}
```

2.8 О чем узнали из второй главы?.

1. Обычно математические функции, встречающиеся в проблемных задачах, зависят не только от аргумента(ов), но и от дополнительных параметров.

2. При оформлении процедур расчета этих функций необходимо позаботиться о передаче им упомянутых дополнительных параметров.

3. **C** и **C++** предоставляют программисту возможность описания переменных глобальных по отношению к использующим их функциям. Так что при определении функции достаточно включить в неё соответствующее расширение области видимости.

4. В ФОРТРАНе-77 нет понятия глобальной переменной. Все переменные локальны по отношению к использующей их процедуре. Поэтому возникает вопрос:

   "Как передать процедуре параметры, введенные главной программой?"

5. В ФОРТРАНе-77 есть два способа его решения:

   (a) Включить в заголовок процедуры описание соответствующих формальных параметров, а при обращении требуемые дополнительные параметры подставлять как фактические. **Это очень неудобно!** *Почему?*

   (b) Включить дополнительные параметры в список элементов так называемой **общей** области (или **common**-блока), содержимое которой доступно из разных единиц компиляции без передачи через формальные параметры.

6. **common**-блок совмещает по памяти данные из разных единиц компиляции.

7. Используем (если уж надо) только именованные **common**-блоки.

8. В ФОРТРАНе глобальны лишь имена **common**-блоков и внешних процедур.

9. Доступ программной единицы к переменным нужного **common**-блока обеспечивается наличием в ней соответствующего **common**-оператора.

10. Оператор **common** – оператор описания и поэтому должен располагаться перед описанием операторов-функций и всеми выполняемыми операторами.

11. Переменные, описанные в **common**-блоке размещаются в оперативной памяти в порядке их указания в списке **common**-оператора.

12. При описании **common**-блока переменные, входящие в него, лучше располагать в порядке убывания длин их типов. В противном случае следует соблюдать **правило выравнивания границ:**

    Адрес любой переменной, входящей в common-блок, должен быть кратен длине, соответствующей ее типу,

    например, описывая неиспользуемые фиктивные переменные.

13. Копирование описания **common**-блока в программные единицы, использующие его, утомительно и чревато случайным уничтожением какой-нибудь его переменной. Поэтому описание **common**-блока **выгодно** поместить в отдельный файл, содержимое которого включать в текст нужной единицы компиляции директивой **include**. Объективно именно этим и удобна ФОРТРАН-директива **include**.

14. В ФОРТРАНе-95 наряду с **common**-блоком существует и альтернативные способы передачи дополнительных параметров:

    (a) оформление подходящих функций **внутренними** так, чтобы дополнительные параметры оказались глобальными по отношению к этим функциям;

    (b) описание нужных параметров в соответствующем **модуле**.

15. **Модуль** — особая единица компиляции современного ФОТРАНа, которая после подключения ее посредством оператора **USE** к нужной единице компиляции обеспечивает доступ последней к описанным в модуле объектам (переменным, типам и процедурам).

16. Оператор **USE** позволяет процедуре, подсоединяющей модуль,

    (a) именовать его объекты иначе нежели они именуются в модуле;

    (b) указывать имена доступных объектов через **only**-подсоединение.

17. **Модуль** предоставляет возможность назначать своим объектам один из двух атрибутов: **public** и **private**.

18. По умолчанию все объекты модуля имеют атрибут **public**, т.е. есть все доступны программной единице, подсоединяющей модуль.

19. Атрибут **private** обычно назначается тем объектам модуля, которые по каким-то причинам должны быть доступны модульным процедурам, но **не должны быть доступны** программе, подсоединяющей модуль.

20. Для расширения области видимости СИ-функции на требуемые дополнительные параметры в функцию помещается **объявление**, состоящее из модификатора **extern**, после которого указывается тип и имена нужных глобальных переменных.

21. Выгодно **определение** и **extern**-объявление глобальных переменных располагать в отдельных файлах, подключая последнее к нужным функциям посредством директивы препроцессора #**include**.

22. Аналогом ФОРТРАН-слова **private** в СИ служит служебное слово **static**, информирующее компилятор о том, что соответствующие глобальные переменные или функции видны только в определяющей их единице компиляции.

2.9 Второе домашнее задание (III-семестр)

Решение каждой из задач должно обеспечивать

- практически мгновенный (необременительный для человека) переход на любую ФОРТРАН-разрядность, допускаемую семейством **real(4,8,10,16)**.
- оценку временны́х затрат на вызовы и работу функций численного интегрирования (в единицах временны́х затрат функции **rectan**). Результаты оценки привести в таблице

```
---------------------------------
         :    З а д а ч а  N ?  :   N 1 : common
---------------------------------   N 2 : common + module
 Тип     : rectan   trap   sim  :   N 3 : module + module
---------------------------------
real( 4) :  1                   :
real( 8) :  1                   :
real(10) :  1                   :
real(16) :  1                   :
---------------------------------
```

1. Модифицировать подинтегральные функции из задачи N 1 первого домашнего задания так, чтобы они через посредство оператора **common** зависели не только от независимой переменной интегрирования, но и от некоторых дополнительных параметров. Модифицировать соответствующую тестирующую программу так, чтобы обеспечить ввод нужных дополнительных параметров из файла.
2. Провести аналогичную модификацию решения задачи N 4 первого домашнего задания с использованием **common**-блока(ов).
3. Провести соответствующую модификацию решения задачи предыдущей задачи, заменив **common**-блок(и) соответствующим(и) **модулем**(ями) и используя при описании переменных модуля **quadra** атрибуты **public** и **private**.
4. Письменно сформулировать: “Какие функции рассматриваемой задачи выгодно разместить в одном модуле и почему?”
5. Задача N 1 первого домашнего задания на СИ.
6. Задача N 4 первого домашнего задания на СИ.

3 Символьный тип данных ФОРТРАНа

В ФОРТРАНе объект символьного типа **character** *(константа, переменная, массив или функция)* предназначен для хранения, использования или выработки нужной последовательности символов. Смысловая нагрузка элементов описания:

```
CHARACTER  длина_элемента    ,  атрибут_элемента  ::   имя_элемента

         |_(если нужна)_|  |_____ (если нужен) ____|
```

Наряду с **длиной_элемента** можно указывать и параметр разновидности (**kind**), который обозначает количество байт для размещения элементарной единицы данного соответствующего типа. Для типа **integer**, например, параметр разновидности может равняться и единице, и двойке, и четверке, и даже восьмерке. Для типа **character** параметр разновидности всегда равен **единице**, то есть полагается, что для размещения одного символа необходим и достаточен один байт.

В качестве **атрибута_элемента** можно, например, взять атрибут **parameter**, если желательно обеспечить **100%** гарантию невозможности изменения содержимого элемента программными средствами.

3.1 Элементарная работа с данными символьного типа

Рассмотрим программу с различными вариантами описания данных символьного типа. В ней выводятся значения символьных переменных и их некоторые числовые характеристики, получаемые посредством вызова встроенных функций **len** и **len_trim**. Первая возвращает длину строки; вторая — число символов в строке без завершающих ее пробелов.

```
program tz; implicit none;     integer, parameter :: n=2147483
character*5      a ; character(7)      b
character(9/3*4) c ; character(len=12) d; character s
character e*5, f*10; character(5-7)    t; character(kind=1) h
character*(n) g; integer i
write(*,*) '123456789a123456789b123456789c123456789d1234567890'
                     write(*,*) a,'=a', len(a),    len_trim(a)
a="abcdef"         ;  write(*,*) a,'=a', len(a),    len_trim(a)
b='a"c''efgh'      ;  write(*,*) b,'=b', len(b),    len_trim(b)
c="1''""4"         ;  write(*,*) c,'=c', len(c),    len_trim(c)
d=a//b             ;  write(*,*) d,'=d', len(d),    len_trim(d)
s='sxy';              write(*,*) s,'=s', len(s),    len_trim(s)
e=a                ;  write(*,*) e,'=e', len(e),    len_trim(e)
f=b                ;  write(*,*) f,'=f', len(f),    len_trim(f)
                      write(*,*) t,'=t', len(t),    len_trim(t)
g=''; write(*,*) 'len(g)=', len(g), "len_trim(g)=",len_trim(g)
do i=1,n
  g(i:i)=char(ichar('0')+mod(i,10))
enddo;
write(*,*) 'g=',g(n-10:n)
end
```

В описании для указания длины элемента можно использовать:

1. Значок ***** (*“звездочка”*) со следующей за ней целочисленной константой без знака (**character*5 a**). Значок ***** с указанием длины можно расположить и после имени очередного элемента, позволяя одним оператором **character** описать несколько строковых переменных разной длины (**character e*5, f*10**).

2. **Заключение в круглые скобки** упомянутой константы (**character(7) b**) или предваряя ее опцией **len=** (**character(len=12) d**). Вместо записи числовой константы возможна запись ее имени (см. описание константы **n** и строковой переменной **g**):

```
integer, parameter :: n=2147483   ! При наличии в описании присваивания
character(len=n) g                ! или атрибута значок :: НЕОБХОДИМ.
```

   или даже константное выражение (см. **character(9/3*4) c**)

Результат пропуска программы **tz.f**:

```
 123456789a123456789b123456789c123456789d1234567890
 Ч*к?=a           5           5
  abcde=a           5           5
  a"c'efg=b           7           7
  1''"4       =c          12           5
  abcdea"c'efg=d          12          12
  s=s           1           1                    ! Если для константы n
  abcde=e           5           5                ! и переменной i указать
  a"c'efg   =f          10           7           ! тип integer*8 и задать
  =t           0           0                    !
  g=34567890123                                 !  n=2147483648, то при
$ gfortran tz.f                                 !
tz.f: In function 'MAIN__':                     ! компиляции получим:
 tz.f:6: error: size of variable 'e' is too large
```

1. Печать строки **123456789a123456789b123456789c123456789d1234567890** поможет отследить размещение печатаемого далее по позициям строки.

2. Если символьной переменной не присвоено начальное значение, то ее содержимое неопределено (см. первую печать переменной **a**).

3. По умолчанию (без явного указания длины элемента) оператор **character** *полагает* ее равной единице. Если содержимое строки содержит больше значков чем объявлено в описании, то избыточные значки отбрасываются (см., например, печать переменных **s**, **b** и вторую печать переменной **a**).

4. Символьная константа ограничивается либо апострофами, либо двойными кавычками. В первом случае, значок апострофа, если он нужен, записывается двумя апострофами, стоящими рядом (**’ ’**). Аналогично, во втором случае, значок *двойная кавычка* записывается двумя рядом стоящими значками *двойной кавычки* (**““**) (см., например, печать переменных **b** и **c**).

5. Для строковых переменных определена операция **конкатенации**, обозначаемая парой, рядом стоящих, прямых слешей // (сравните значения **a**, **b** и **d**).

6. Если начальное значение, присваиваемое символьной переменной, содержит меньше символов чем задано при ее описании, то хвостовые символы заполняются **пробелами** (см. печать значения переменной **f**).

7. Символьный элемент с отрицательным значением длины полагается строкой **нулевой длины**, как и константа, обозначаемая **двумя**, стоящими рядом, апострофами или двойными кавычками (см. первую печать **len(g)** и **t**.

8. Строковая переменная допускает выделение любой подстроки, входящей в нее. Для выделения достаточно после ее имени указать номера начального и конечного значков желаемой подстроки, разделенные двоеточием. В частности,

   (a) **g(i:i)** – значок из **i**-й позиции строковой переменной **g**.

   (b) **g(:i)** – подстрока из символов строки **g**, начиная с первого по **i**-й.

   (c) **g(i:)** – подстрока из символов строки **g**, начиная с с **i**-го до последнего.

   (d) **g(k:m)** – подстрока, ограниченная **k**-м и **m**-м символами строки **g** (**k>0**, **m>0**, **k<=m**, **m** не должно превышать длину строки).

9. Множество символов, допускаемых языком программирования для вывода, это – некоторый алфавит, в котором каждый символ имеет свой номер (код). Буквы латиницы в нем следуют друг за другом в алфавитном порядке причем строчные и прописные буквы представляются двумя разными наборами. Набор десятичных цифр также расположен в порядке возрастания изображаемых ими чисел. При решении текстовых задач важна возможность получать по номеру символа из алфавита сам символ и, наоборот, по символу – его номер. Соответствующие встроенные ФОРТРАН-функции:

   (a) **ichar(c)** – возвращает значение типа **integer** равное коду значка, содержащегося в символьной переменной **c** (так **ichar('0')** – код цифры нуль).

   (b) **char(i)** возвращает значение типа **character*1** (значок, код которого в алфавите допустимых символов равен **i**).

10. **g(i:i)=char(ichar('0')+7** означает: поместить в **i**-ю позицию строки **g** значок с кодом равным коду цифры **'0'**, увеличенному на семь, т.е. коду цифры **'7'**, которая и занесется в элемент **g(i:i)**; а **char(ichar('a')+1)** – это буква **'b'**.

11. Таким образом, подстрока **g(n-10:n)** будет содержать младшие десятичные цифры последних одиннадцати значений параметра цикла. Помним, что результат вызова функции **mod(i,10)** (находящей остаток от деления нацело целого значения **i** на десять) равен младшей десятичной цифре числа, хранящегося в переменной **i**. Функции **ichar** и **char** ФОРТРАНа – аналоги соответственно функций **ord** и **chr** ПАСКАЛя.

3.2 Значок “звездочка” при описания символьных данных

Значок “*****” может использоваться при описания символьного объекта и сам по себе, а не только как символ, предваряющий значение длины строки. В частности, описание

1. **character(*), parameter :: a=’абракадабра’** означает, что объявленная длина строковой константы **a** равна количеству символов, указанных при ее инициализации.

```
 program tststr1
 implicit none
 character(*) s0, s1, s2
 parameter  ( s0="(*) при описании длины строковой константы",
>             s1=’означает, что ее длина равна’,
>             s2=’числу заданных символов’)
 character(*), parameter :: a=’абракадабра’
 character(53)  s3
 character(70) :: s4=
>"123456789a123456789b123456789c123456789d123456789e123456789f"
 write(*,’(1x,a,t60,i7,t70,i7)’) s0,len(s0),len_trim(s0)
 write(*,’(1x,a,t60,i7,t70,i7)’) s1, len(s1), len_trim(s1)
 write(*,’(1x,a,t60,i7,t70,i7)’) s2, len(s2), len_trim(s2)
 write(*,’(1x,a,t60,i7,t70,i7)’)  a,  len(a), len_trim(a)
 s3=s1//s2
 write(*,’(1x,a,t60,i7,t70,i7)’) s0, len(s0), len_trim(s0)
 write(*,’(1x,a,t60,i7,t70,i7)’) s3, len(s3), len_trim(s3)
 write(*,’(1x,a,t60,i7,t70,i7)’) s4, len(s4), len_trim(s4)
 s4=s1//’ ’//s2
 write(*,’(1x,a,t60,i7,t70,i7)’) s4, len(s4), len_trim(s4)
 end
```

   (a) Формат вывода указан строковой константой: **’(1x,a,t60,i7,t70,i7)’**

   (b) В ней буква **a** (дескриптор преобразования для символьных данных) указывает, что из соответствующего строкового элемента списка вывода будут выведены все имеющиеся в нем символы (дескриптор **a5** – выведет лишь пять первых символов).

   (c) **t60** – установка вывода очередного данного с 60-й позиции.

   Результат работы **tststr1.f**:

```
(*) при описании длины строковой константы                    42        42
означает, что ее длина равна                                  28        28
числу заданных символов                                       23        23
абракадабра                                                   11        11
(*) при описании длины строковой константы                    42        42
означает, что ее длина равначислу заданных символов          53        51
123456789a123456789b123456789c123456789d123456789e12345678    70        60
означает, что ее длина равна числу заданных символов         70        52
```

2. Явное задание длины формального параметра символьного типа в виде конкретного числа не всегда выгодно. Выгоднее указать, что формальный параметр перенимает длину фактического, делая текст процедуры более универсальным. Наличие **(*)** после **character** это и означает.

```
      program tststr2                              ! Файл tststr2.f
      implicit none
      character(70) t0; character t1*47, t2*65, t3*35
      t0=" (*) при описании формального параметра символьного типа"
      t1=" означает, что"
      t2=" при обращении к подпрограмме этот формальный параметр"
      t3=" перенимает размер фактического"
      call prtstr('123456789.123456789.123456789.123456789.123456789.'//
     >            '1234567')
      call prtstr(t0)  ! Длины переменных t0, t1, t2, t3 различны.
      call prtstr(t1)  ! prtstr выводит эти переменные в полном
      call prtstr(t2)  ! соответствии с их размером, указанном в главной
      call prtstr(t3)  ! программе. Сопоставьте номера колонок вывода ':'
            end        ! с заявленной длиной фактических аргументов.
      subroutine prtstr(s)
      implicit none    ! При описании параметра строкового типа можно в
      character (*) s  ! качестве длины строки указать (*) или (len=*).
      write(*,1) s, len(s), len_trim(s)  ! Тогда формальный параметр
    1 format(1x,a,':',t60,i5,t73,i2)     ! перенимает длину фактического
      end
```

Результат работы программы **tststr2.f**:

```
 123456789.123456789.123456789.123456789.123456789.1234567:   57        57
  (*) при описании формального параметра символьного типа     70       :56
  означает, что                                  :            47        14
  при обращении к подпрограмме этот формальный параметр       65  :     54
  перенимает размер фактического    :                        35        31
```

3.
```
      program tststr3
      implicit none
      character(*) :: b(4)=(/'1','1234',"1234567890",''/)
      integer i           ! Указание (*) при описании длины элемента
      do i=1,4            ! символьного массива устанавливает
        call prtstr(b(i)) ! длину каждого из элементов, равной наибольшей
      enddo               ! длине из инициализирующих массив констант.
      end
```

Результат работы **tststr3.f**:

```
 1                   10          1
 1234                10          4
 1234567890          10         10
                     10          0
```

4. При описании функции, возвращающей через своё имя строковое значение, значок (*) может указывать и длину этого значения. Естественно, эта длина будет равна значению, указанному в программе, вызывающей функцию. Однако, исходный текст функции при использовании (*) формально не будет зависеть от конкретных числовых значений, и, тем самым, окажется более универсальным.

```
program tststr4
implicit none
character(4)  :: s1=':/*+';  character(2)  :: s2='12'
character(1)  :: s3='#'   ;  character(9)  :: s4='_шахматы_'
character(15) :: s5=' 1234.56789.abcd'; character(50) cp4
write(*,*)'123456789.123456789.123456789.123456789.123456789.'
call prtstr(cp4(s1)); call prtstr(cp4(s2))
call prtstr(cp4(s3)); call prtstr(cp4(s4)); call prtstr(cp4(s5))
end
function cp4(s)        ! Функция cp4(s) сцепляет в одну строку
implicit none          ! четыре копии аргумента s.
character (*) cp4      ! Длина результата равна 50 (так указано
character (*) s        ! в главнойпрограмме. Однако,  исходный
character (4*len(s)) t ! текст cp4 не зависит явно ни от этой
t=s//s//s//s           ! константы, ни от длины фактичекого
cp4=trim(t)            ! параметра (лишь бы 4*len(s)<=50).
end
```

Результат работы **tststr4.f**:

```
$ gfortran tststr4.f prtstr.f
$ ./a.out
 12345678901234567890123456789012345678901234567890
 :/*+:/*+:/*+:/*+                                          50          16
 12121212                                                  50           8
 ####                                                      50           4
 _шахматы__шахматы__шахматы__шахматы_                      50          36
  1234.56789.abc 1234.56789.abc 1234.56789.abc 1234        50          50
```

В функции **cp4** встречается обращение к встроенной функции **trim**, которая возвращает строку без завершающих ее пробелов.

3.3 Встроенные функции обработки символьных данных

Некоторые из них (**len**, **len_trim**, **char**, **ichar**, **trim**) уже знаем. Все они – **элементные**, то есть – их аргументами могут быть и скаляры, и массивы (см. [2]). В случае массива результатом является массив, согласованный с массивами-параметрами. Значение элемента массива-результата получается применением элементной функции к соответствующему элементу массива-аргумента.

Имеются и другие элементные функции обработки символьных данных.

- **achar(i)** возвращает значок типа **character*1**, **ASCII**-код которого равен значению **i** типа **integer** (1<= i<= 127).
- **iachar(c)** возвращает для аргумента **c** его **ASCII**-код типа **integer*1**.

```
program tststr55;     ! Программа вызывает процедуру
implicit none         ! tr1016, которая используя
character(10) s       ! встроенные функции achar и iachar
integer n             ! формирует символьную строку
write(*,*) 'input n'  ! из шестнадцатеричных цифр,
read (*,*) n          ! получаемых в результате перевода
write(*,*) ' n=',n    ! заданного целого в шестнадцатеричную
call tr1016(n,s)      ! систему счисления.
write(*,*) ' s=', s
end
subroutine tr1016(nn,s)
implicit none
integer, parameter :: ia0=iachar('0'), iaa=iachar('A')
integer nn, n, i, f
character (*) s; character*1 c
n=nn; i=10; s=''
do; f=mod(n,16)
    if (f.lt.10) then; c=achar(ia0 + f   )
                 else; c=achar(iaa + f-10)
    endif
    s(i:i)=c; n=n/16
    i=i-1
    if (n.eq.0) exit
enddo
end
```

Пример результата работы **tststr55.f**

```
input n
32762
  n=       32762
  s=      7FFA
```

Наряду с **achar** и **iachar** имеются функции с похожими именами **char** и **ichar**, выполняющие те же действия по отношению к последовательности символов, поддерживаемой операционной системой.

- **Элементные функции сравнения строк:**

  lge(a,b) возвращает **true**, если строка **a** $\geq$ строки **b** (иначе **false**)
  lgt(a,b) возвращает **true**, если строка **a** $>$ строки **b** (иначе **false**)
  lle(a,b) возвращает **true**, если строка **a** $\leq$ строки **b** (иначе **false**)
  llt(a,b) возвращает **true**, если строка **a** $<$ **e** строки **b** (иначе **false**)

```
      program tststr6;   implicit none
      character*30 a, b; character(3) ta, tb
      character*7 w(3) /'кура','река', 'шахматы'/
      character*7 v(0:2) /'рука','река', 'шашки'  /; logical l(3)
      a='a'; b='a';
      ta=trim(a); tb=trim(b); write(*,*)' ta=',ta; write(*,*)' tb=',tb
      write(*,1001) 'lge(ta,tb)',' .ge. '  ,'  lgt(ta,tb)',' .gt. ',
     >           '  lle(ta,tb)',' .le. ',','  llt(ta,tb)',' .lt. '
      write(*,1002) ' ',lge(ta,tb),ta.ge.tb,lgt(ta,tb),ta.gt.tb,
     >                  lle(ta,tb),ta.le.tb,llt(ta,tb),ta.lt.tb
      a='abc'; b='bac';
      ta=trim(a); tb=trim(b); write(*,*)' ta=',ta; write(*,*)' tb=',tb
      write(*,1002) ' ',lge(ta,tb),ta.ge.tb,lgt(ta,tb),ta.gt.tb,
     >                  lle(ta,tb),ta.le.tb,llt(ta,tb),ta.lt.tb
      l=lge(w,v); write(*,*) w; write(*,*) v; write(*,'(l3,l7,l9)') l
      l=w>=v    ; write(*,*) w; write(*,*) v; write(*,'(l3,l7,l9)') l
          write(*,'(1x,a,i5,a,3i5)') 'w: ',len(w), '...',len_trim(w)
          write(*,'(1x,a,i5,a,3i5)') 'v: ',len(v), '...',len_trim(v)
 1001 format(1x,7x, a,a,a,a,a,a,a,a)
 1002 format(1x,a5,l9,l7,l11,l7,l11,l7,l11,l7,l11,l7)
      end
```

```
 ta=a                                ! Результат работы tststr6.f:
 tb=a
      lge(ta,tb) .ge.   lgt(ta,tb) .gt.   lle(ta,tb) .le.   llt(ta,tb) .lt.
           T      T          F      F          T      T          F      F
ta=abc
tb=bac
            F      F          F      F          T      T          T      T
 кура   река   шахматы    !  Слово 'шахматы' длиннее слова 'шашки', но слово
 рука   река   шашки      !  'шашки' БОЛЬШЕ слова 'шахматы', т.к. 'ш'>'х'.
  F      T        F
 кура   река   шахматы      ! Операция >= с массивами работает поэлементно,
 рука   река   шашки        ! т.е. так же, как и функция lge.
  F      T        F         ! Вопрос: если можно обойтись операциями
 w:     7...    4    4    7 !   сравнения, то зачем нужны функции сравнения?
 v:     7...    4    4    5
```

1. Строка меньшей длины перед сравнением дополняется пробелами справа.
2. Сравнение символов выполняется покодово слева направо.
3. Параметрами могут быть и согласованные массивы (см.**l=lge(w,v)**).
4. Обычные операции сравнения применимы и к строковым значениям.

- **Элементные функции выравнивания строк:**

  adjustl(a) – перенос всех ведущих пробелов строки **a** в ее хвост.

  adjustr(a) – перенос всех хвостовых пробелов стоки **a** в ее начало.

```
program tststr7
implicit none;
character(25) p(12); integer i
p(1)=' Жена водоноса,'           ; p(2)='  Как это ни странно, '
p(3)='   Обет - нет вопроса - '  ; p(4)=' Держала исправно, '
p(5)='  Хоть масса счастливых  ' ; p(6)='   Возможностей рядом:'
p(7)='   Старух говорливых'      ; p(8)='С рентгеновским взглядом,'
p(9)='    Подружек-прелестниц,'  ; p(10)='     Всегда задушевных, '
p(11)='  И кумушек-сплетниц '    ; p(12)='Профессии верных. '
write(*,'(7x,a,10x,a,5x,a)') &
       ' исходный ','выравнивание по левому', 'выравнивание по правому',&
       '  текст   ','краю           adjustL', 'краю            adjustR',&
       ''','''','''
do i=1,4; write(*,1000) p(i),'|',adjustl(p(i)),'|',adjustr(p(i)) ! Выравнивание:
enddo                                                            !  строки,
p(5: 8)=adjustl(p(5:8))                                          ! вырезки из
p(9:12)=adjustr(p(9:12));  write(*,1005) (p(i),p(i+4), i=5,8)    ! вектора и
p=adjustl(p); write(*,'(/(1x,t26,"|",a,"|"))') (p(i),i=9,12)     ! всего вектора
write(*,*) '12345678901234567890123456789012345678901234567890123456789012345'
   1000 format(1x,a,a,a,a,a,a)
   1005 format(/(1x,t26,'|',a,'|',a))
end
```

Результат работы **tststr7.f**:

```
        исходный           выравнивание по левому     выравнивание по правому
         текст             краю           adjustL     краю            adjustR

  Жена водоноса,          |Жена водоноса,           |          Жена водоноса,
   Как это ни странно,    |Как это ни странно,      |     Как это ни странно,
    Обет - нет вопроса -  |Обет - нет вопроса -     |    Обет - нет вопроса -
  Держала исправно,       |Держала исправно,        |       Держала исправно,

                          |Хоть масса счастливых    |    Подружек-прелестниц,
                          |Возможностей рядом:      |      Всегда задушевных,
                          |Старух говорливых        |      И кумушек-сплетниц
                          |С рентгеновским взглядом,|       Профессии верных.

                          |Подружек-прелестниц,     |
                          |Всегда задушевных,       |
                          |И кумушек-сплетниц       |
                          |Профессии верных.        |
 12345678901234567890123456789012345678901234567890123456789012345
```

Последняя строка вывода приведена для уяснения номеров позиций. Действие спецификаторов формата **x** и **t** похоже – определяет позицию данного в строке (**x** – относительно позиции предыдущего данного, а **t** – относительно начала строки).

- **Другие функции по работе со строками:**

  repeat(s,n) возвращает строку, состоящую из **n** копий строки **s**.

index(a,suba)	**index(a,suba,.true.)**
возвращает номер позиции, на которой в строке **a**	
начинается самое левое	**завершается самое правое**
вхождение подстроки **suba**.	

scan(a,suba)	**scan(a,suba,.true.)**
возвращает номер позиции, на которой в строке **a**	
встречается самый левый	**встречается самый правый**
символ подстроки **suba**.	

verify(a,b)	**verify(a,b,.true.)**
возвращает **0**, если каждый символ строки **a** присутствует в строке **b**, иначе номер	
самого левого символа	**самого правого символа**
строки **a**, которого нет в строке **b**	

```
program tststr7
implicit none
character*25 p(2)
integer lf(2), lr(2), i
character*3 s /'не '/
p(1)='Я не муха и не щука,  '; p(2)='И не чижик, и не кот, '
write(*,'(a)') '12345678901234567890123456789012345678901234567890'
write(*,'(a)') repeat(p(1),3)   ! если копий мало, то проще использовать //
write(*,1010); write(*,1020) s; lf=index(p,s); lr=index(p,s ,.true.)
write(*,'(a, i3, 5x, i3, 5x,i3, 5x, i3, 5x, i3, 5x, i3)/')            &
         (        p(i),          lf(i),    lr(i),                     &
                          scan(p(i),s),   scan(p(i),s,.true.),        &
                        verify(p(i),s), verify(p(i),s,.true.), i=1,2)
s='нет'; write(*,*) 's=',s;
lf=index(p,s); lr=index(p,s,.true.)
write(*,'(a, i3, 5x, i3, 5x,i3, 5x, i3, 5x, i3, 5x, i3)')             &
         (        p(i),          lf(i),    lr(i),                     &
                         scan(p(i),s),scan(p(i),s,.true.),            &
                       verify(p(i),s), verify(p(i),s,.true.), i=1,2)
s='чиж'; write(*,&
'(38x,"verify(''чиж'',p(2))=",i3,i8)') verify(s,p(2)),verify(s,p(2),.true.)
1010 format(25x,'index',3x,'index',3x,'scan',3x,'scan',3x,'verify',3x,'verify')
1020 format(1x,'s=',a3,27x,' true',3x,'     ',1x,' true',13x,'true')
      end

$ ./a.out                !            Результат работы tststr8:
12345678901234567890123456789012345678901234567890
Я не муха и не щука,     Я не муха и не щука,     Я не муха и не щука,
                         index   index   scan   scan   verify   verify
   s=не                          true           true              true
Я не муха и не щука,       3      13       2      25       1      20
И не чижик, и не кот,      3      15       2      25       1      21
   s=нет
Я не муха и не щука,       0       0       3      14       1      25
И не чижик, и не кот,      0       0       3      20       1      25
                                       verify('чиж',p(2))=  0       0
```

3.4 Символьные переменные в качестве внутреннего файла

До сих пор ввод данных осуществляли либо с экрана, либо из внешнего файла на диске. ФОРТРАН позволяет вводить данные и из, так называемого, **внутреннего** файла, в качестве которого может использоваться переменная типа **character** подходящей длины (переменная размещается в оперативной памяти – потому файл и называется внутренним):

```
program tststr9 ! Программа tststr9 вводит в строковую переменную f десятичную
implicit none   ! запись данного типа real, а затем читает из f эту запись и
character(8) f  ! переводит ее во внутреннее машинное представление типа REAL
real r          !.....................
write(*,*) 'введи данное типа real' ! Когда подобное может быть полезным?
read(*,*) f                         !
write(*,*) ' f=',f                  !               Очевидно, тогда,  когда
 ! Здесь можно было бы обратиться   ! хотим, чтобы  программа  после  ввода
 ! к подпрограмме, проверяющей      ! данного сначала проверила правильность
 ! и синтаксическую правильность    ! его синтаксической записи;  и только
 ! набора данного, и его проблемную ! после подтверждения правильности
 ! корректность.                    ! осуществила бы преобразования символьной
read(f,*) r                         ! строки в данное типа real.
write(*,*) ' r=',r
end
```

```
program tststr9a   ! Программа tststr9a демонстрирует, что ошибка в записи
implicit none      ! данного отлавливается автоматически и без использования
real r             ! внутреннего файла. Например, при вводе вместо 3.e6 записи
write(*,*) 'введи данное типа real'  ! 3.w6 получим сообщение:
read(*,*) r                          ! At line 6 of file tststr9a.f95. Fortran
write(*,*) ' r=',r ! runtime error: Bad real number in item 1 of list input
end
```

Заметим, что программа **tststr9a** завершилась аварийно, то есть закончила свою работу. Если нас устраивает подобный режим, то обходимся без внутренних файлов. Однако, иногда приходиться писать программы (рассчитанные на диалог с человеком), которые даже в случае ошибки ввода со стороны пользователя должны не прекращать работу, а, напротив, сообщив о некорректном наборе, предоставить возможность повторного ввода. В этом случае возможно, например, ввести данное как строковое значение в строковую переменную, затем синтаксически формально проверить корректность записи чисел и, если она верна, прочитать их из строковой переменной по соответствующему типу данного спецификатору оператора **format**. Причину некорректности читаемого данного можно запомнить посредством опции **iostat** оператора **open**.

Заметим также, что иногда может потребоваться неоднократное чтение одной и той же записи. При чтении с диска придется возвращать считывающий элемент на её начало, например, командой **backspace**, в то время как при хранении записи в строковой переменной (т.е. использования внутреннего файла) никакого возврата на начало записи организовывать не нужно.

Символьная переменная, в качестве внутреннего файла, оказывается полезной, когда при форматном выводе данных повторитель формата желателен в виде переменной, хотя по синтаксису ФОРТРАНа во многих компиляторах это невозможно.

Пусть, например, в переменную **n** (целого типа) занесено значение, хранящее число используемых элементов одномерного массива **a(n)**. Требуется оформить вывод так, чтобы сначала в одну строку вывелся лишь один элемент массива **a(1)**, в следующую – два элемента **a(1)** и **a(2)**, в следующую три — **a(1)**, **a(2)** и **a(3)** и т.д., и, наконец, в **n**-ую — все элементы с **a(1)** по **a(n)**. Использование в качестве повторителя константы равной некоторому заранее разумно выбранному максимальному числу, вообще говоря, возможно. Однако, для разных задач это максимальное число может оказаться различным. Если же упомянутая форма вывода массива генерируется в подпрограмме, то хотелось бы, чтобы исходный текст процедуры не зависел от конкретных числовых констант.

Возможность использования переменной (например, **s**) строкового типа в качестве имени внутреннего файла позволяет вывести в неё оператором

write(s,'(i3)') i

значение, хранящееся в переменной **i**, после чего включить строку **s** в форматную строку (уже в качестве повторителя) требуемого формата:

```
program tststr14
implicit none
integer, allocatable :: a(:)
integer i, n, k
write(*,*) 'input integer n'; read (*,*) n; write(*,*) ' n=',n
allocate (a(1:n))
a=(/ (i,i=1,n) /);   write(*,*) a;   call frmvar(a,n)
end

subroutine frmvar(a,n)
implicit none
integer  a(n)
integer i, n, k
character*3 s
character*12 f
do i =1, n
  write(s,'(i3)') i;    f='('//s//'i3)';   write(*,f) (a(k),k=1,i)
enddo
end
```

```
 input integer n
  n=           5
           1           2           3           4           5
  1
  1  2
  1  2  3
  1  2  3  4
  1  2  3  4  5
```

3.5 О чем узнали из третьей главы?

1. Единственной операцией ФОРТРАНа над данными строкового типа служит операция **конкатенации**, которая обозначается двумя следующими непосредственно друг за другом прямыми слэшами (//).

2. Результат операции **конкатенация** зависит от перемены мест операндов.

3. При описании символьной переменной её длина указывается константой целого типа, расположенной либо после **character***, либо внутри круглых скобок, следующих после **character**.

4. Строковая или символьная константа в исходном ФОРТРАН-тексте заключается либо в апострофы (старые версии ФОРТРАНа), либо в двойные кавычки.

5. Если начальное значение, присваиваемое строковой переменной содержит меньше символов чем указано при её описании, то её хвостовые элементы заполняются пробелами.

6. Из строковой переменной всегда можно выделить любую её подстроку посредством конструкции

   ```
   имя_переменной (левая_граница : правая_граница)
   ```

7. При описании **character(*)** массива строкового типа совместно с его инициализацией длина элемента массива устанавливается равной наибольшей из длин инициализирующих значений.

8. Длину строковой переменной, являющейся формальным параметром, удобно указывать описанием **character(*)**.

9. Встроенные ФОРТРАН-функции, нацеленные на работу с символьными данными – **элементные**, т. е. способны обработать не только скаляр, но и массив типа **character**.

10. Наиболее востребованными из встроенных функций, обеспечивающих работу со строками являются:

    len(s) – определение длины строки;

    len_trim – определение длины строки без хвостовых пробелов.

    achar(k) – получение **ASCII**-символа по его коду (1<=k<=127);

    iachar(c) – получение кода **ASCII** символа **c**;

11. Имеются элементные встроенные функции сравнения, выравнивания и сканирования строк.

12. Строковые переменные иногда удобно использовать в качестве **внутреннего** файла, т.е. в качестве своеобразных устройств ввода-вывода.

3.6 Третье домашнее задание (III-семестр).

1. Разработать функцию, которая по однозначному десятичному целому получает обозначающую его десятичную цифру (т.е. значение типа **character*1**).

2. Разработать функцию, которая по заданной десятичной цифре (т.е. значению типа **character(1)**) находит его числовой эквивалент целого типа.

3. Написать программу, которая выводит строку с буквами латиницы в прямом и обратном порядках.

4. Разработать ФОРТРАН-функцию, которая по заданному неотрицательному целому числу возвращает через свое имя в вызывающую программу строковое значение, моделирующее результат перевода этого числа в **шестнадцатеричную** систему счисления.

5. Разработать ФОРТРАН-функцию, которая по заданному неотрицательному целому числу возвращает через свое имя в вызывающую программу строковое значение, моделирующее результат перевода этого числа в **Z**-ичную систему счисления (2<=Z<=36).

6. Разработать ФОРТРАН-подпрограмму, которая в качестве входных аргументов использует две строковые переменные, моделирующие многозначные десятичные числа, и получает в качестве результата строку, моделирующую их сумму.

7. Написать программу, которая, построчно вводя из файла русский текст, кодирует его *соблюдая* правила:

   (a) каждая гласная заменяется следующей гласной **алфавита** (буква "**я**" – на букву "**а**");

   (b) каждая согласная заменяется следующей согласной **алфавита**

   (c) буквы **"й, ь, ъ"**, как и знаки препинания, остаются без изменения

   При замене гласной на гласную рифма, как правило, сохраняется. Поэтому есть шанс из стихов получить рифмованную "*бессмыслицу*", которая при сохранении ударений звучит довольно забавно. Например,

   Муха, муха-цокотуха перейдет в **Ныце, ныце-чулуфыце**.

8. Обработка русского текста ФОРТРАН-программой может оказаться не такой простой, как в случае латиницы, если операционная система поддерживает кодировку UTF8. Попробуйте, разобраться в проблеме самостоятельно. (работа встроенных функций **achar** и **char**, чтение смешанного текста с кириллицей, латиницей и знаками препинания). Если же русские символы однобайтовые, как и латиница, (например, кодировка **koi8r**, то проблем не возникает).

4 Тип данных СТРУКТУРА (каталог Hipparcos)

В прошлом семестре (раздел **Немного о массивах, структурах и указателях**) мы уже знакомились с типом данных **структура**. Применим его при написании программ, обрабатывающих каталог **Hipparcos**, описание работы с которым имеется в книге [20] (есть в астрономической библиотеке).

Здесь же СИ- и ФОРТРАН-программы из [20] приведены просто для демонстрации правильной организации работы с данными, для описания которых идеально подходит тип **структура**. Отличие приводимых здесь ФОРТРАН-программ от первоисточника [20] состоит в замене символа . (*точка*), разделяющего имя структурированной переменной от имён её полей, на символ %, и неиспользовании функции **eof** (определения признака окончания файла), так как обозначения . и **eof** не допускались по синтаксису стандарта имеющейся версии компилятора **gfortran**.

4.1 Краткий справочник по форматам полей каталога

bytes	Format	Units	Label	Explanation	
1	A1	—	Catalog	[H] Catalogue (H=Hipparcos)	(H0)
5- 14	I12	—	HIP	Identifier (HIP number)	(H1)
16	A1	—	Proxy	[HT] Proximity flag	(H2)
18- 28	A11	—	RAhms	Right ascension in h m s, ICRS (J1991.25)	(H3)
30- 40	A11	—	DEdms	Declination in °, ′ ″, ICRS (J1991.25)	(H4)
42- 46	F5.2	mag	Vmag	Magnitude in Johnson V	(H5)
48	I1	—	VarFlag	*[1,3]? Coarse variability flag	(H6)
50	A1	—	r_Vmag	*[GHT] Source of magnitude	(H7)
52- 63	F12.8	deg	RAdeg	*? alpha, degrees (ICRS, Epoch=J1991.25)	(H8)
65- 76	F12.8	deg	DEdeg	*? delta, degrees (ICRS, Epoch=J1991.25)	(H9)
78	A1	—	Astroref	*[*+A-Z] Reference flag for astrometry	(H10)
80- 86	F7.2	mas	Plx	? Trigonometric parallax	(H11)
88- 95	F8.2	mas/yr	PmRA	? Proper motion mu_alpha.cos(delt), ICRS	(H12)
97-104	F8.2	mas/yr	PmDe	? Proper motion mu_delta, ICRS	(H13)
106-111	F6.2	mas	e_RAdeg	? Standard error in RA*cos(DEdeg)	(H14)
113-118	F6.2	mas	e_DEdeg	? Standard error in DE	(H15)
120-125	F6.2	mas	e_Plx	? Standard error in Plx	(H16)
127-132	F6.2	mas/yr	e_PmRa	? Standard error in PmRA	(H17)
134-139	F6.2	mas/yr	e_PmDE	? Standard error in PmDE	(H18)
141-145	F5.2	–	DE:RA	[-1/1]? Correlation, DE/RA*cos(delta)	(H19)
147-151	F5.2	–	Plx:RA	[-1/1]? Correlation, Plx/RA*cos(delta)	(H20)
153-157	F5.2	–	Plx:DE	[-1/1]? Correlation, Plx/DE	(H21)
159-163	F5.2	–	PmRA:RA	[-1/1]? Correlation, PmRA/RA*cos(delta)	(H22)
165-169	F5.2	–	PmRA:DE	[-1/1]? Correlation, PmRA/DE	(H23)
171-175	F5.2	–	PmRA:Plx	[-1/1]? Correlation, PmRA/Plx	(H24)
177-181	F5.2	–	PmDE:RA	[-1/1]? Correlation, PmDE/RA*cos(delta)	(H25)
183-187	F5.2	–	PmDE:DE	[-1/1]? Correlation, PmDE/DE	(H26)
189-193	F5.2	–	PmDE:Plx	[-1/1]? Correlation, PmDE/Plx	(H27)
195-199	F5.2	–	PmDE:PmRA	[-1/1]? Correlation, PmDE/PmRA	(H28)
201-203	I3	–	F1	? Percentage of rejected data	(H29)
205-209	F5.2	–	F2	*? Goodness-of-fit parameter	(H30)
211-216	I6	–	—	HIP number (repetition)	(H31)
218-223	F6.3	mag	BTmag	? Mean BT magnitude	(H32)
225-229	F5.3	mag	e_BTmag	? Standard error on BTmag	(H33)

bytes	Format	Units	Label	Explanation	
231-236	F6.3	mag	VTmag	? Mean VT magnitude	(H34)
238-242	F5.3	mag	e_VTmag	? Standard error on VTmag	(H35)
244	A1	—	m_BTmag	*[A-Z*-] Reference flag for BT and VTmag	(H36)
246-251	F6.3	mag	B - V	? Jonson B - V colour	(H37)
253-257	F5.3	mag	e_B - V	? Standard error on B - V	(H38)
259	A1	—	r_B - V	[GT] Source of B - V from Ground and Tycho	(H39)
261-264	F4.2	mag	V - I	? Colour index in Cousins' system	(H40)
266-269	F4.2	mag	e_V - I	? Standard error on V - I	(H41)
271	A1	—	r_V - I	[A-T] Source V - I	(H42)
273	A1	—	Combmag	[*] Flag for combined Vmag, B - V, V - I	(H43)
275-281	F7.4	mag	Hpmag	Median magnitude in Hipparcos system	(H44)
283-288	F6.4	mag	e_Hpmag	Standard error on Hpmag	(H45)
290-294	F5.3	mag	Hpscat	? Scatter on Hpmag	(H46)
296-298	I3	—	o_Hpmag	? Number of obaservations for Hpmag	(H47)
300	A1	—	m_Hpmag	*[A-Z*-] Reference flag for Hpmag	(H48)
302-306	F5.2	mag	Hpmax	? Hpmag at maximum (5th percentile)	(H49)
308-312	F5.2	mag	Hpmin	? Hpmag at minimun (95th percentile)	(H50)
314-320	F7.2	d	Period	? Variability period (days)	(H51)
322	A1	—	HvarType	*[CDMPRU]? variability type	(H52)
324	A1	—	moreVar	*[12] Additional data about variability type	(H53)
326	A1	—	morePhoto	*[ABC] Light curve Annex	(H54)
328-337	A10	—	CCDM	CCDM identifier	(H55)
339	A1	—	n_CCDM	*[HIP] Historical status flag	(H56)
341-342	I2	—	Nsys	? Number entries with same CCDM	(H57)
344-345	I2	—	Ncomp	? Number components in this entry	(H58)
347	A1	—	Multflag	*[CGOVX] Double/Multiple Systems flag	(H59)
349	A1	—	Source	*[PFILS] Astrometrical source flag	(H60)
351	A1	—	Qual	*[ABCDS] Solution quality	(H61)
353-354	A2	—	m_Hip	Components identifiers	(H62)
356-358	I3	deg	theta	? Position angle between components	(H63)
360-366	F7.3	arcsec	rho	? Angular separation between components	(H64)
368-372	F5.3	arcsec	e_rho	? Standard error on rho	(H65)
374-378	F5.2	mag	dHp	Magnitude difference of components	(H66)
380-383	F4.2	mag	e_dHp	? Standard error on dHp	(H67)
385	A1	—	Survey	[S] Flag indicating a Survey Star	(H68)
387	A1	—	Chart	*[DG] Identification Chart	(H69)
389	A1	—	Notes	*[DGPWXYZ] Existence of notes	(H70)
391-396	I6	—	HD	[1/359083]? HD number <III/135>	(H71)
398-407	A10	—	BD	Bonner DM <I/119>, <I/122>	(H72)
409-418	A10	—	CoD	Cordoba Durchmusterung (DM) <I/114>	(H73)
420-429	A10	—	CPD	Cape Photographic DM <I/108>	(H74)
431-434	F4.2	mag	(V - I)red	V - I used for reductions	(H75)
436-447	A12	—	SpType	Spectral Type	(H76)
449	A1	—	r_SpType	*[1234GKSX]? Source spectral type	(H77)

Сокращённое описание каталога даётся в [20] — **Приложение A**, из которого взят и приведённый выше краткий справочник по форматам полей. В приложениях **D** и **E** [20] приводятся исходные тексты программ по работе с каталогом на языках программирования СИ и ФОРТРАН-90 соответственно.

Исполнимый файл получается из главной программы **d32.c**, файла **hipmain.c** с функциями открытия **OpenHipparcosMain**, закрытия **CloseHipparcosMain** и чтения **ReadHipparcosMain** файла-каталога **hipmain.dat**, и заголовочного фай-

ла **hipmain.h** с прототипами указанных функций и с определением типа данных **THipparcos**, в поля которого расфасовываются данные о текущей звезде.

На первый взгляд кажется, что ввиду краткости фрагментов открытия и закрытия файла-каталога, их можно было внедрить в тело главной программы, не оформляя функциями **OpenHipparcosMain** и **CloseHipparcosMain**. Тем не менее использованный подход более практичен и надёжен, так как в принципе не позволит случайно что-либо испортить в исходном тексте главной программы, если, например, потребуется уточнить что-то, касающееся открытия и/или закрытия каталога.

4.2 C-версия программы

```
// Листинг D 3.1.  Модуль чтения каталога Hipparcos. Заголовочный файл.
// Расположение полной версии каталога Hipparcos
#define HipparcosName "/home/aw/lecture/semestr3/Hipparcos/hipmain.dat"
#define HipNumOfStars 118218
                                //                      Файл hipmain.h
typedef struct
 { long HIP;                    // Номер звезды по Hipparocs
                                // Астрометрическая информация:
  double RAdeg,DEdeg;           //   экваториальные координаты в градусах
  double Plx;                   //   тригонометрический параллакс в mas
  double pmRA,pmDE;             //   собств. движ. ma*cos(d) и md mas/год
  char   AstroRef;              //   флаг для кратных систем.
                                // Фотометрическая инфомация:
  float VMag;                   //   звёздная вел. по шкале Джонсона,
  float B_V ;                   //   показатель цвета B-V по шкале Джонсона
  struct                        // Ошибки соответствующих величин
  { double RAdeg,DEdeg;         //         имена
    double Plx;                 //               полей
    double pmRA,pmDE;           //                     структуры
  } sigma;                      // с именем sigma
  char Sp[10+1];                // Развернутый спектральный класс
  struct                        // Битовая структура отстствия информации:
   { unsigned NoRaDe:1;         //    Нет данных о точных координатах
     unsigned NoPlx :1;         //    Нет данных о параллаксе
     unsigned NoPM  :1;         //    Нет данных о собственных движениях
     unsigned NoVMag:1;         //    Нет данных о звездной величине
     unsigned NoB_V :1;         //    Нет данных о показателе цвета
   }  Info;                     // с именем Info (отсутствия информации)
} THipparcos;
                                  // Прототипы функций:
  int OpenHipparcosMain();        //         открытия файла каталога,
  void CloseHipparcosMain();      //         закрытия файла каталога,
  int ReadHipparcosMain(THipparcos *s); // чтения одной строки каталога
                                  // на место указываемое s.
 // При достижении признака окончания  файла возвращает false, т.е 0.
```

```
// Функции чтения каталога Hipparcos.                    Файл  hipmain.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "hipmain.h"
#define  HipRecSize 451                 // длина строки каталога, включая CR,LF

int IsBlank(char *s)                    // Возвращает true, если s состоит
{ while (*s) if (*s++ != ' ') return 0; //  только из пробелов
   return 1;
}
FILE  *f;                               // Файловая переменная каталога
int OpenHipparcosMain()                 // Открытие файла каталога
 {                                      // в режиме "только чтение"
   f=fopen(HipparcosName,"rt");
   if (f) return 1; else return 0;
 }
void CloseHipparcosMain() { fclose(f); }; // Закрытие файла каталога
int ReadHipparcosMain(THipparcos *s)      // Чтение одной записи
 {
  char hs[HipRecSize];                    // Строка каталога
  char st[16];                            // Вспомогательный буфер.
  if (feof(f)) return 0;                  //Если конец файла, то False
  fread(hs,HipRecSize,1,f);               // Чтение одной строки
  s->Info.NoRaDe=0;                       // Обнуление всех битов флага
  s->Info.NoPlx =0;  s->Info.NoPM =0;
  s->Info.NoVMag=0;  s->Info.NoB_V =0;

  // Мы не используем форматный ввод оператором scanf, с тем,
  // чтобы полностью контролировать процесс чтения полей (в том
  // числе и пустых). Функции atoi и atof возвращают нулевые
  // значения в случае ошибки преобразования, поэтому надо
  // проверять поля на пробелы отдельно.

  // Интерпретация 12 байт,  начиная с 3-го - это номер HIP.
  s->HIP=atoi(strncpy(st,hs+3-1,12));

  // Чтение координат: по 12 байт с 52 и с 65 позиции.
  strncpy(st,hs+52-1,12); st[12]=0; s->RAdeg=atof(st);
  if (IsBlank(st))  s->Info.NoRaDe=1;
  strncpy(st,hs+65-1,12); st[12]=0; s->DEdeg=atof(st);
  if (IsBlank(st))  s->Info.NoRaDe=1;

  // Чтение параллакса - 7 байт с 80-й позиции
  strncpy(st,hs+80-1,7); st[7]=0; s->Plx=atof(st);
  if (IsBlank(st))  s->Info.NoPlx=1;

  // Чтение собственных движений: по 8 байт с 88 и с 97 позиции
  strncpy(st,hs+88-1,7); st[7]=0; s->pmRA=atof(st);
  if (IsBlank(st))  s->Info.NoPM=1;
  strncpy(st,hs+97-1,7); st[7]=0; s->pmDE=atof(st);
  if (IsBlank(st))  s->Info.NoPM=1;

  s->AstroRef=hs[78-1]; // Флаг кратной звезды
```

```
  // Чтение зв.величины и показателя цвета B-V по Джонсону
  strncpy(st,hs+42-1,5); st[5]=0; s->VMag=atof(st);
  if (IsBlank(st))  s->Info.NoVMag=1;
  strncpy(st,hs+246-1,6); st[6]=0; s->B_V=atof(st);
  if (IsBlank(st))  s->Info.NoB_V=1;
  if (!s->Info.NoRaDe)
    {  // Данные об ошибках всегда присутствуют,
       // если присутствуют и сами координаты.
     strncpy(st,hs+106-1,6); st[6]=0; s->sigma.RAdeg=atof(st);
     strncpy(st,hs+113-1,6); st[6]=0; s->sigma.DEdeg=atof(st);
    }
  if (!s->Info.NoPlx)
    { strncpy(st,hs+120-1,6); st[6]=0; s->sigma.Plx=atof(st);}
  if (!s->Info.NoPM)
   { strncpy(st,hs+127-1,6); st[6]=0; s->sigma.pmRA=atof(st);
     strncpy(st,hs+134-1,6); st[6]=0; s->sigma.pmDE=atof(st);
   }
  strncpy(s->Sp,hs+436-1,10); st[10]=0; // Чтение данных о спектр. классе
 return 1;
}

// Листинг D 3.2.                                    Файл      d32.c
// Подсчет звезд без данных о координатах,
// собственных движениях и параллаксах
#include <stdio.h>
#include "hipmain.h"
main()
{
 THipparcos s;                 // Структура данных о звезде.
 long NoCoord = 0;             // Счетчик звезд без точных координат
 long NoProp  = 0;             // Счетчик звезд без собств. движений
 long NoPar   = 0;             // Счетчик звезд без параллаксов
 int k;                        // Битовая маска
 OpenHipparcosMain();          // Открытие каталога
 while (ReadHipparcosMain(&s)) // Цикл чтения каталога
 {
  if (s.Info.NoRaDe) NoCoord++; // Сравнение битов
  if (s.Info.NoPM)   NoProp++;  //        в  маске
  if (s.Info.NoPlx)  NoPar++;   //           с константами
 }
 CloseHipparcosMain();          // Закрытие каталога

 printf("Звезд без точных координат    %ld\n",NoCoord); // Вывод
 printf("Звезд без собственных движений %ld\n",NoProp); // резуль-
 printf("Звезд без параллаксов %ld\n", NoPar);          // татов.
}
```

Результат работы С-программы

```
Звезд без точных координат    263
Звезд без собственных движений 263
Звезд без параллаксов 263
```

4.3 ФОРТРАН-версия программы

```
module hipmain ! Модуль структуры каталога Hipparcos.  Файл hipmain.f95
implicit none  !                                       ================
               ! Путь к расположению полной версии каталога Hipparcos:
character(*), parameter :: &
&       HipparcosName ='/home/aw/lecture/semestr3/Hipparcos/hipmain.dat'
integer, parameter ::  HipNumOfStars = 118218  ! Число звезд в Hipparcos
integer, parameter :: u = 10                   ! Номер файла
TYPE  THipparcos
  SEQUENCE
  INTEGER(4) :: HIP           ! Номер звезды по Hipparocs
                              ! Астрометрическая информация:
  REAL(8)      :: RAdeg,DEdeg !    экваториальные координаты в градусах
  REAL(8)      :: Plx         !    тригонометрический параллакс в mas
  REAL(8)      :: pmRa,pmDE   !    собственные движения ma*cos(d) и md в mas/год
  CHARACTER(1) :: AstroRef    ! Флаг для кратных систем
                              ! Фотометрическая инфомация:
  REAL(4)  :: VMag            !    звездная вел. по шкале Джонсона
  REAL(4)  :: B_V             !    показатель цвета B-V по шкале Джонсона
  REAL(8) :: sigma_RAdeg      ! Ошибки соответствующих величин
  REAL(8) :: sigma_DEdeg, sigma_Plx
  REAL(8) :: sigma_pmRa,sigma_pmDE
  CHARACTER(10) Sp       ! Развернутый спектральный класс
  LOGICAL NoRaDe         ! Нет данных о точных координатах
  LOGICAL NoPlx          ! Нет данных о параллаксе
  LOGICAL Nopm           ! Нет данных о собственных двжиениях
  LOGICAL NoVMag         ! Нет данных о звездной величине
  LOGICAL NoB_V          ! Нет данных о показателе цвета
END TYPE THipparcos
CONTAINS
  subroutine OpenHipparcosMain    ! Подпрограмма открытия файла каталога
  integer ier                     ! Индикатор успешности открытия.
  open(u,file=HipparcosName,iostat=ier)       ! Открытие каталога.
  if (ier/=0) then
     print *,'Не могу найти каталог Hipparcos'! Реакция на невозможность
     stop 111                                 !               открытия.
  endif
  end subroutine OpenHipparcosMain
  subroutine CloseHipparcosMain
  close(u)
  end subroutine CloseHipparcosMain
  logical function ReadHipparcosMain(s)  ! Функция чтения текущей строки
  implicit none                          ! каталога и заполнения полей
                                         ! структуры текущей звезды.
  character(450) hs                      ! Переменная для строки каталога
  type(THipparcos), intent(out) :: s
  integer ier
  read(u,'(A450)',iostat=ier) hs         ! 4итаем текущую строку
  ReadHipparcosMain=(ier==0)
  if (ier==-1) then
    print *, 'достигнут маркер окончания файла'
    return
  endif
```

```
  if (ier>0 ) then
    print *, 'в данном обнаружена ошибка  '; stop 222
  endif
  s%NoRaDe =.false.           ! Сбрасываем флаги событий
  s%NoPlx  =.false.
  s%Nopm   =.false.
  s%NoVMag =.false.
  s%NoB_V  =.false.
  read(hs(3:14),*) s%hip      ! 12 байт, начиная с 3-го - это номер HIP
  if (len(trim(hs(52:63))) == 0)  then ! Координаты: по 12 байт
    s%NoRaDe=.true.; s%RADeg=0.0       ! с 52 и с 65 позиции
    else
      read(hs(52:63),*) s%RAdeg
  endif
  if (len(trim(hs(65:76))) == 0)  then
    s%NoRaDe=.true.; s%DEDeg=0.0       ! Нуль пишем на всякий случай
    else
      read(hs(65:76),*) s%DEdeg
  endif
  if (len(trim(hs(80:86))) == 0) then  ! Параллакс: 7 байт
    s%NoPlx=.true.; s%Plx=0.0          !    с 80-й позиции
    else
      read(hs(80:86),*) s%Plx
  endif
  if (len(trim(hs(88:95))) == 0) then ! Собств. движения:
    s%NoPM=.true.; s%pmRA=0.0         ! по 8 байт с 88 и с 97
    else
      read(hs(80:86),*) s%pmRA
  endif
  if (len(trim(hs(88:95))) == 0) then
    s%NoPM=.true. ;s%pmDE=0.0
    else;  read(hs(97:104),*) s%pmDE
  endif
  s%AstroRef=hs(78:78)                 ! Флаг кратной звезды
  if (len(trim(hs(42:46))) == 0) then  ! Зв.в. и показ. цвета
    s%NoVMag=.true.; s%VMag=0.0        ! B-V по шкале Джонсона
    else;  read(hs(42:46),*) s%VMag
  endif
  if (len(trim(hs(246:251))) == 0) then
    s%NoB_V=.true. ; s%B_V=0.0; else; read(hs(246:251),*) s%B_V
  endif
  if (.not. s%NoRADE) then
    read(hs(106:111),*) s%sigma_RAdeg  ! Данные об ошибках
    read(hs(113:118),*) s%sigma_DEdeg  ! всегда присутствуют,
  endif                                ! если присутствуют и
  if (.not. s%NoPlx) then              ! сами координаты
    read(hs(120:125),*) s%sigma_Plx
  endif
  if (.not. s%Nopm) then; read(hs(127:132),*) s%sigma_pmRA
                          read(hs(134:139),*) s%sigma_pmDE
  endif
  s%Sp=hs(436:445)         ! Чтение данных о спектральном классе
 end function ReadHipparcosMain
end module hipmain
```

```
PROGRAM test2   ! Подсчет числа всех звезд в каталоге.      Файл test2.f95
use hipmain     ! Подключение модуля HipMain
implicit none
integer(4) nstar       ! Счетчик числа звезд
character(450)    hs   ! Текущая читаемая строка каталога Hipparcos.
TYPE(THipparcos)   s   ! Описание переменной s типа THipparcos.
logical b
call OpenHipparcosMain
nstar=0                              ! Число звезд до чтения
do while (ReadHipparcosMain(s))      ! Бесконечный цикл:
   nstar=nstar+1                     ! Учет в счётчике очередной звезы
enddo
call CloseHipparcosMain
print *,'4исло звезд nstar=',nstar,' = ', HipNumOfStars
stop 0000
end program test2
```

Замечание:

- Компоновка исполнимого файла ФОРТРАН -программы а При чтении строки каталога ФОРТРАН-функцией **ReadHipparcosMain** в [20] использована функция **eof**, которая нацелена на проверку достижения признака окончания файла. В предлагаемой здесь программе она не используется по причине её отсутствия в доступной версии компилятора **gfortran**.

Результат работы ФОРТРАН-программы

```
 достигнут маркер окончания файла
 4исло звезд nstar=      118218  =       118218
```

4.4 ФОРТРАН-расчёт распределения звёзд по абсолютной величине

Программа выбирает только те звёзды, для которых относительная погрешность определения параллаксов выше 50%. Программа по сути дела просто скопирована из [20]. Отличаются детали. Например, в [20] диапазон абсолютных величин выбран **[-12:12]**; здесь же он положен **[-13:15]** так как было обнаружено, что общее число звёзд с указанной погрешностью определения параллакса оказалось бóльшим количества звёзд с диапазоном **[-12:12]**. Кроме того, демонстрируется насколько удобно иногда включать в текст программ обработки и процедуру вывода соответствующего gnuplot-скрипта. Так процедура **plot(ia,Mmin,Mmax,nplt)** выводит в файл **dist.plt 28** строк, содержащих количества звёзд, попадающих в соответствующий столбец гистограммы, разворачивая запись чисел, хранящихся в векторе **ia**, по вертикали.

```
program distrib ! Вычисление распределения звезд       Файл distrib.f95
use hipmain     ! Hipparcos по абсолютной величине.
implicit none
integer, parameter :: nplt=10
integer, parameter :: Mmin=-13
integer, parameter :: Mmax= 15
integer(4) nstar, nstar1 ! Счетчик числа звезд
character(450)    hs     ! Текущая читаемая строка каталога Hipparcos.
TYPE(THipparcos)   s     ! Описание переменной s типа THipparcos.
integer ia(Mmin:Mmax)    ! Статистика
integer round            ! Тип функции округления
integer  i               ! Вспомогательная переменная
real(8)  r               ! Расстояние
real(8)  M               ! Абсолютная звёздная величина
ia=0                     ! Обнуление статистики
call OpenHipparcosMain
nstar=0
do while (ReadHipparcosMain(s))        ! Пока не прочли весь каталог
  if ( s%NoPlx    ) cycle              ! Нет данных о параллаксе
  if ( s%Plx <=0d0) cycle              ! Неположительный параллакс
  if (s%sigma_Plx/s%Plx>0.5d0)  cycle  ! Точность параллакса хуже 50%
  r=1000d0/s%Plx                       ! Расчёт расстояния в парсеках
  M=s%Vmag-5d0*dlog10(r)+5d0           ! Расчёт абс.зв.величины
  i=round(M)
  if (i>=Mmin .and. i<=Mmax) ia(i)=ia(i)+1
  nstar=nstar+1
enddo
call CloseHipparcosMain

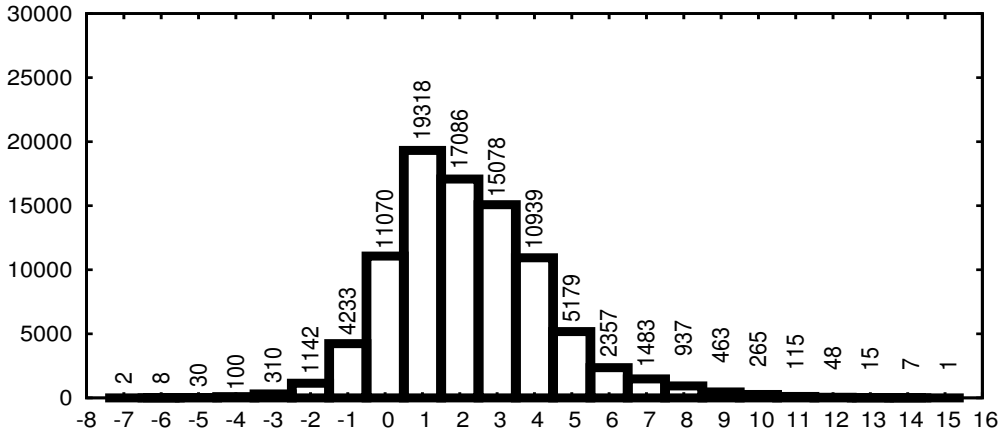
print '(i3,3x,i7)', (i,ia(i),i=Mmin,Mmax)
nstar1=0
do i=Mmin, Mmax
  nstar1=nstar1+ia(i)
enddo
print *, ' #  nstar=',nstar
print *, ' # nstar1=', nstar1, sum(ia)
open (nplt,file='dist.plt',status='replace')
call plot(ia,Mmin, Mmax,nplt)
close(nplt)
stop 0000
end program distrib

function round(x)
implicit none
integer round
real(8) x; if (x<0d0) then; round=idint(x-0.5d0)
                      else; round=idint(x+0.5d0)
           endif
end function round
```

```
subroutine plot( ia,Mmin,Mmax,nplt)
implicit none
integer Mmin, Mmax
integer ia(Mmin:Mmax)
character(1) z; character(3) sx
character(5) str, sy, sp; character(128) txt(38)
integer s1, s2, ii, i, nplt
txt=' #'
txt(1)='set terminal postscript eps 32'
txt(2)='set output ''distrib.eps'' '
txt(3)='set nokey'
txt(4)='set border 15 lw 3'
txt(5)='set size 1.5, 1'
txt(6)='set xtics -8, 1, 16'
do i=Mmin, Mmax
 z=' '; if (i<0) z='-';  ii=iabs(i)
 s1=ii/10;  s2=mod(ii,10)
 sx=z //achar(s1+iachar('0')) //achar(s2+iachar('0'));  sy=str(ia(i))
 sp=str(ia(i)+iabs(5000-ii*100))
 txt(20+i)='set label "'//sy//'" at '//sx//' , '//sp//' center rotate by 90'
enddo
txt(36)='plot [-8:16] [0:30000] ''distrib.dat'' u 1:2 w boxes lt -1 lw 10'
txt(37)='unset label'
txt(38)='reset'
do i=1,38
  write(nplt,'(a)') txt(i)
enddo
end

function str(nn)
implicit none
character(*) str
character(10) w
character(1) sf
integer nn, n, f
n=nn
str=''
do while (n/=0)
  f=mod(n,10)
  sf=achar(f+iachar('0'))
  w=str
  str=sf//w
  n=n/10
enddo
end
```

```
set terminal postscript eps 24
set output 'distrib.eps'
set nokey
set border 15 lw 3
set size 1.5, 1
set xtics -8, 1, 16
set label "     " at -13 , 4000  center rotate by 90
set label "     " at -12 , 3800  center rotate by 90
set label "     " at -11 , 3900  center rotate by 90
set label "     " at -10 , 4000  center rotate by 90
set label "     " at -9 , 4100  center rotate by 90
set label "     " at -8 , 4200  center rotate by 90
set label "2    " at -7 , 2500  center rotate by 90
set label "8    " at -6 , 2500  center rotate by 90
set label "30   " at -5 ,2500  center rotate by 90
set label "100  " at -4 , 2500  center rotate by 90
set label "310  " at -3 , 3000  center rotate by 90
set label "1142 " at -2 , 4000  center rotate by 90
set label "4233 " at -1 , 7000  center rotate by 90
set label "11070" at  0 , 14000 center rotate by 90
set label "19318" at  1 , 22500 center rotate by 90
set label "17086" at  2 , 20200 center rotate by 90
set label "15078" at  3 , 18000 center rotate by 90
set label "10939" at  4 , 14000 center rotate by 90
set label "5179 " at  5 , 8000  center rotate by 90
set label "2357 " at  6 , 5300  center rotate by 90
set label "1483 " at  7 , 4500  center rotate by 90
set label "937  " at  8 , 5137  center rotate by 90
set label "463  " at  9 , 4563  center rotate by 90
set label "265  " at  10 , 4265  center rotate by 90
set label "115  " at  11 , 4015  center rotate by 90
set label "48   " at  12 , 3848  center rotate by 90
set label "15   " at  13 , 3715  center rotate by 90
set label "7    " at  14.1 , 3607  center rotate by 90
set label "1    " at  15.1 , 3501  center rotate by 90
plot [-8:16] [0:30000] 'distrib.dat' u 1:2 w boxes lt -1 lw 10
unset label
reset
```



4.5 Четвёртое домашнее задание (III-семестр).

1. Активировать программу расчёта распределения звёзд по абсолютной звёздной величине, приведённую в лекции.

2. Написать программу, которая выводит список звёзд заданного спектрального класса.

3. Написать (по желанию) программу, которая вводит координаты некоторых звёзд (из имеющихся в каталоге **Hipparcos**), например, любимого (или фиктивного) созвездия, и в приемлемой координатной сетке перевычисляет их на дискретном отрезке времени. Написать GNUPLOT-скрипт, который используя полученный результат, демонстрирует изменение контура созвездия.

5 GNU-совмещение языков программирования.

ФОРТРАН издавна используется для решения задач расчётного характера. Широко востребован для решения аналогичных задач и язык СИ. Иногда возникает необходимость обратиться из С-программы к ФОРТРАН-процедуре. Реальна и обратная задача — из ФОРТРАН-программы вызвать полезную С-функцию. Посредством **GNU**-компилятора компиляторов **gcc** это делается просто (см. [11] глава 10; [27]).

5.1 Простые примеры.

1. C++ вызовы ФОРТРАН-функций и -подпрограмм.
2. ФОРТРАН-вызовы C++-функций.
3. C-вызовы ФОРТРАН-функций и -подпрограмм.
4. ФОРТРАН-вызовы C-функций.
5. C-вызовы C++-функций.
6. C++-вызовы C-функций.
7. О чем узнали из параграфа 5.7? (кратко)

5.1.1 C++ вызовы ФОРТРАН-функций и -подпрограмм.

1. Рассмотрим слегка модифицированный пример из [6]. Дан ФОРТРАН-текст функции **cube(n)**, которая возводит аргумент **n** целого типа в куб, и подпрограммы **kvad(n, m)**, которая возводит первый аргумент того же типа в квадрат, помещая результат во второй аргумент **m**.

```
function cube (n)    ! Здесь не обсуждается: выгодно или нет оформить
implicit none        ! возведение в квадрат подпрограммой.
integer cube, n      ! Нам важна формальная сторона дела:
cube= n*n*n          !
end                  !
subroutine kvad(n,m) ! Научиться правильно подключать к программе,
implicit none        ! написанной на C++, объектные коды простейших
integer n, m         ! независимых программных единиц ФОРТРАНа и
m=n*n                ! корректно вызывать их.
end                  !
```

Вызов их из программы, написанной на C++, выполняется так:

```
#include <iostream>
using namespace std;
extern "C" void kvad_(int *, int *);
extern "C" int  cube_(int *);
int main()
{ int n, n2, n3;
  cout << " Введи целое n"<< endl;  cin >> n; cout << " n=" << n << endl;
  kvad_(&n, &n2);  cout << n <<"**2 =" << n2 << endl;
  n3=cube_(&n);    cout << n <<"**3 =" << n3 << endl; return 0; }
```

Здесь (при описании прототипов используемых функций) указано не только **extern"C"** – их *иностранное — имеется ввиду* (по отношению к C++) происхождение, но и типы формальных аргументов оформлены указателями: **int ***. Вспоминаем (см. **3.3.7**), что ФОРТРАН всегда передает свои аргументы **по адресу** в то время как C – всегда **по значению**. В переводе на язык C – это означает, что формальные C-аргументы должны быть указателями, а фактические C-аргументы – адресами. Поэтому в C-программе перед соответствующими именами помещен значок **&** (операция получения адреса).

В приведенном тексте (в отличие от соответствующего текста [6]) справа к именам программных единиц ФОРТРАНа, вызываемых C-обращениями, добавлен **знак подчеркивания**. Наш пример, нацелен на использование **GNU**-компилятора **gfortran**, который добавляет этот символ к имени соответствующего объектного кода, чтобы отличать упомянутые имена от имен одноименных C-функций, что связано с различием соглашений о способе передачи параметров (см. [6]).

make-файл проекта пропуска задачи может, например, быть таким:

```
comp1:=c++
comp2:=gfortran
main       : testcfor.o subfun.o
[  TAB ]    $(comp1) testcfor.o subfun.o -o main
testcfor.o : testcfor.cpp
[  TAB ]    $(comp1) -c testcfor.cpp
  subfun.o : subfun.f
[  TAB ]    $(comp2) -c subfun.f
 clear     :
[  TAB ]    rm *.o
 result    :  main
[  TAB ]    ./main
```

В нем переменные **comp1** и **comp2** хранят имена используемых компиляторов. Результаты пропуска таковы:

```
$ make result
./main
 Введи целое n
9
 n=9
9**2 =81
9**3 =729
```

Если в исходном ФОРТРАН-тексте изменим тип **integer** на **real**, а в исходном C-тексте тип **int** на **float**, то и результат получим соответствующий одинарной точности.

2. Следуя разобранному примеру, обратимся из C++-программы

```
#include <cmath>       // для использования exp
#include <stdio.h>     // для использования fopen, printf, scanf

extern "C" { float  w0_(float *); float  w1_(float *); }
int main()
{  float t0, tn, t, ht, x, r0, r1;  int n, i; FILE *ninp, *nres;
  ninp=fopen("input","r");
  if (ninp==NULL) { printf("Неудача открытия файла input.\n"); return 1;}
  nres=fopen("result","w");
  if (nres==NULL){ printf("Неудача открытия файла result.\n"); return 2;}
   fscanf(ninp,"%e %e %d", &t0, &tn, &n);
   fprintf(nres," #  t0=%e  tn=%e  n=%i\n", t0, tn, n);
   ht=(tn-t0)/n;
   fprintf(nres," # %2s %10s %15s %15s\n","i","t","r0","r1");
   for (i=1; i<=n;i++)
     { t=t0+ht*(i-1);  x=exp(-t);
       r0=w0_(&x);
       r1=w1_(&x);
       fprintf(nres,"%5i %15.7e %15.7e %15.7e\n", i, t, r0, r1);
     }
   fclose(ninp); fclose(nres);
   return 0;
}
```

решающей задачу из контрольной N 1 (см. главу **7**), к ФОРТРАН-функциям **w0** и **w1**:

```
      function w0(x) result(w); implicit none         !    Файл w0.for
      real x, x2, w, a, b
      x2=x*x; a=cos(x) - 1 + x2* 0.5*(1 - x2/12 * ( 1 - x2 / 30))
              b=sin(x)/x - 1+x2/6 * (1-x2/20*(1-x2/42))
      w=a/b
      end

      function w1(x) result(w);                        !    Файл w1.for
      implicit none; real w, x, x2, sa1, sa, sb, a, b
      integer k
      x2=x*x; sa1=1.0; sa=0; sb=0; a=1/40320.0; b=a/9; k=0
      do while (abs(sa).ne.sa1)
        sa1=abs(sa)
        sa=sa+a; a=-a*x2/(2*k+ 9)/(2*k+10)
        sb=sb+b; b=-b*x2/(2*k+10)/(2*k+11); k=k+1
      enddo
      w=sa/sb
      end
```

Соответствующий **make**-файл может иметь вид:

```
comp1:=c++
comp2:=gfortran
main : tmixcf30.o w0.o w1.o
  $(comp1) tmixcf30.o w0.o w1.o -o $@
tmixcf30.o : tmixcf30.cpp
  $(comp1) -c tmixcf30.cpp
     w0.o : w0.for
  $(comp2) -c w0.for
     w1.o : w1.for
  $(comp2) -c w1.for
 clear  :
  rm -f *.o main
 result : input main
  ./main
 restab : result main
  cat result
 resplt : result main
  gnuplot 'view.gnu'
```

```
 #  t0=1.000000e+00  tn=2.000000e+00  n=10
 #  i          t              r0              r1
    1   1.0000000e+00   0.0000000e+00   8.9975405e+00
    2   1.1000000e+00   3.5000000e+00   8.9979858e+00
    3   1.2000000e+00   1.0434783e+00   8.9983501e+00
    4   1.3000000e+00   1.7142858e-01   8.9986496e+00
    5   1.4000000e+00   3.0000000e+00   8.9988937e+00
    6   1.5000000e+00   5.2631581e-01   8.9990950e+00
    7   1.6000000e+00  -3.1460676e-01   8.9992599e+00
    8   1.7000000e+00  -5.7142860e-01   8.9993935e+00
    9   1.8000000e+00  -1.1739130e+00   8.9995041e+00
   10   1.9000001e+00   5.7971016e-02   8.9995947e+00

 #  t0=1.000000e+00  tn=2.000000e+00  n=10            // Старый результат,
 #  i          t              r0              r1      // полученный
    1   1.0000000e+00   5.7829318e+00   8.9975405e+00 // после отработки
    2   1.1000000e+00   6.6944509e+00   8.9979858e+00 // исполнимого файла,
    3   1.2000000e+00   6.4263763e+00   8.9983501e+00 // собранного из
    4   1.3000000e+00   7.2002325e+00   8.9986496e+00 // объектных файлов
    5   1.4000000e+00   3.8472526e+00   8.9988937e+00 // w0.o и w1.o,
    6   1.5000000e+00   1.7917061e+00   8.9990950e+00 // полученных C++
    7   1.6000000e+00   3.4897115e+00   8.9992599e+00 // компилятором
    8   1.7000000e+00   3.1662014e+00   8.9993935e+00 // при компиляции
    9   1.8000000e+00   3.1587539e+00   8.9995041e+00 // файлов w0.cpp и
   10   1.9000000e+00   2.9607465e+00   8.9995947e+00 //        w1.cpp
```

Очевидно, что в обоих результатах значения **r1** соответственно совпадают при разительном отличии значений **r0** от **r1**, которые в идеале должны быть одинаковы.

5.1.2 ФОРТРАН-вызовы C++-функций

1. Исходные **C++**-тексты функций **ckvard** и **ccube** первого из примеров предыдущего пункта могут иметь вид:

```
extern "C"  void ckvad_( int *, int *);
extern "C"  int  ccube_( int *);
void ckvad_(int *n, int *m)
{ int k;  k= *n; *m= k*k; return;}
int ccube_(int *n)
{ int k; k= *n; return k*k*k;}
```

Главная программа на ФОРТРАНе:

```
      program testc                          ! Главная ФОРТРАН-программа
      implicit none                          ! вызывает функции, имена
      integer ccube, n, i, n2, n3            ! которых пишутся, как
      write(*,*) 'введи целое n'             ! ckvad и ccube.
      read (*,*) n;write(*,*) ' n=',n        ! Однако, при  написании их
      call ckvad(n,n2); write(*,1001)  n, n2 ! исходных текстов и
      n3=ccube(n);      write(*,1002)  n, n3 ! прототипов на C++ эти
 1001 format(1x,' n=',i5,'**2=',i10)         ! имена следует писать как
 1002 format(1x,' n=',i5,'**3=',i10)         !       ckvad_ и ccube_.
      end
```

Соответствующий проекту простейший **make**-файл:

```
comp1:=gfortran
comp2:=c++
main       : forc.o cfun.o
[  TAB ]      $(comp1) forc.o cfun.o -o main
 forc.o    : forc.f
[  TAB ]      $(comp1) -c forc.f
  subfun.o : subfun.f
[  TAB ]      $(comp2) -c cfun.cpp
 clear     :
[  TAB ]       rm *.o
 result    :  main
[  TAB ]      ./main
```

И, наконец, результат пропуска:

```
 введи целое n
9
  n=           9
  n=    9**2=        81
  n=    9**3=       729
```

2. Теперь обратимся из ФОРТРАН-программы, решающей задачу контрольной, к функциям **w0** и **w1**, объектные коды которых получены компиляцией исходных **C++**-текстов. Аналогично предыдущему примеру дописываем значок подчёркивания в имя каждой из функций и добавляем соответствующую строку с указанием **extern**:

```
#include <math.h>
extern "C" double wd0_(double *);
double wd0_(double *y)                              //    Файл w0.cpp
{double x2, a, b, x;
      x=*y; x2=x*x;
      a=cos(x) - 1 + x2* 0.5l*(1l - x2/12l * ( 1l - x2 / 30l));
      b=sin(x)/x - 1+x2/6l * (1l-x2/20l*(1l-x2/42l));
      return(a/b);
}
```

```
#include <cmath>
extern "C" double wd1_(double *);
double wd1_(double *y)                               //    Файл w1.cpp
{double x,x2, sa1, sa, sb, a, b;
 int k, k2;
 x=*y; x2=x*x; sa1=1.0; sa=0; sb=0; a=1/40320.0; b=a/9; k=k2=0;
 while (fabs(sa)!=sa1)
   { sa1=fabs(sa); sa=sa+a; a=-a*x2/(k2+ 9)/(k2+10);
                   sb=sb+b; b=-b*x2/(k2+10)/(k2+11); k=k+1; k2=2*k;
   }
 return (sa/sb);
}
```

В исходном тексте ФОРТРАН-программы в местах вызова функций используем обычные ФОРТРАН-имена **wd0** и **wd1**

```
      program tmixfc30;  implicit none             !    Файл  tmixfc30.f
      real(8) wd0, wd1, t0, tn, t, ht, x, r0, r1
      integer, parameter :: ninp=5, nres=6
      integer n, i
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,'(d15.7)')  t0, tn
      read(ninp,'(i15)') n
      write(nres,'(" # t0=",d23.15,2x,"tn=",d23.15,2x,"n=",i3)') t0,tn,n
      ht=(tn-t0)/n
      write(nres,'(" #",2x,"i",12x,"t",21x,"w0",21x,"w1")')
      do i=0,n
          t=(t0+i*ht); x=dexp(-t); r0=wd0(x); r1=wd1(x)
          write(nres,1001) i, t,  r0, r1
      enddo
      close(nres)
 1001 format(1x,i4,d23.15,d23.15,d23.15)
      end
```

Результаты работы исполнимого файла, полученный линковкой объектного кода главной ФОРТРАН-программы и объектных кодов C++ функций:

```
# t0=  0.100000000000000D+01  tn=  0.200000000000000D+01  n= 10
#  i            t                      w0                     w1
   0  0.100000000000000D+01  0.899754087712832D+01  0.899754060051501D+01
   1  0.110000000000000D+01  0.899798635357056D+01  0.899798622967396D+01
   2  0.120000000000000D+01  0.899835355376421D+01  0.899835114074282D+01
   3  0.130000000000000D+01  0.899866240707475D+01  0.899864994539330D+01
   4  0.140000000000000D+01  0.899891650505822D+01  0.899889461325594D+01
   5  0.150000000000000D+01  0.899909488985519D+01  0.899909494866309D+01
   6  0.160000000000000D+01  0.899925226373302D+01  0.899925898169114D+01
   7  0.170000000000000D+01  0.899923934343499D+01  0.899939328880004D+01
   8  0.180000000000000D+01  0.899916818488433D+01  0.899950325567335D+01
   9  0.190000000000000D+01  0.900004287170312D+01  0.899959329262975D+01
  10  0.200000000000000D+01  0.900343180712275D+01  0.899966701113196D+01
```

Старый результат, полученный исключительно на базе компилятора **C++**:

```
#  t0=1.000000e+00  tn=2.000000e+00  n=10
#  i         t                r0                r1          aer      rer
   1 1.0000000000e+00 8.9975408830e+00 8.9975406005e+00 2.82e-07 3.14e-08
   2 1.1000000000e+00 8.9979863484e+00 8.9979862297e+00 1.19e-07 1.32e-08
   3 1.2000000000e+00 8.9983535469e+00 8.9983511407e+00 2.41e-06 2.67e-07
   4 1.3000000000e+00 8.9986624089e+00 8.9986499454e+00 1.25e-05 1.39e-06
   5 1.4000000000e+00 8.9989164630e+00 8.9988946133e+00 2.18e-05 2.43e-06
   6 1.5000000000e+00 8.9990948232e+00 8.9990949487e+00 1.25e-07 1.39e-08
   7 1.6000000000e+00 8.9992521780e+00 8.9992589817e+00 6.80e-06 7.56e-07
   8 1.7000000000e+00 8.9992391060e+00 8.9993932888e+00 1.54e-04 1.71e-05
   9 1.8000000000e+00 8.9991686096e+00 8.9995032557e+00 3.35e-04 3.72e-05
  10 1.9000000000e+00 9.0000427391e+00 8.9995932926e+00 4.49e-04 4.99e-05
```

Совпадение результатов по обеим формулам в обоих вариантах расчёта не совсем полное, несмотря на то что расчёт вёлся на типах **real(8)** и **double**. Для сравнения полезно взглянуть на результаты работы арифметики одинарной точности (см. предыдущий пример). Там вообще не было никакого совпадения, поскольку в исходной формуле, и в числителе, и в знаменателе происходила катастрофическая потеря точности из-за вычитания почти равных с точностью до 10^{-8} при $x = exp(-2.0) \approx 0.13$.

В случае же работы на арифметике **real(8)** (шестнадцатизначная в десятичной системе счисления мантисса) восемь-десять из шестнадцати цифр ещё удаётся сохранить, что и подтверждается, например, выводом оценок абсолютной и относительной погрешностей.

5.1.3 С-вызовы ФОРТРАН-функций

Для наглядности приведём исходные тексты ФОРТРАН-функций **w0** и **w1**:

```
      function w0(x) result(w); implicit none          !    Файл w0.for
      real x, x2, w, a, b
      x2=x*x
      a=cos(x) - 1 + x2* 0.5*(1 - x2/12 * ( 1 - x2 / 30))
      b=sin(x)/x - 1+x2/6 * (1-x2/20*(1-x2/42))
      w=a/b
      end function w0

      function w1(x) result(w); implicit none          !    Файл w1.for
      real w, x, x2, sa1, sa, sb, a, b
      integer k
      x2=x*x; sa1=1.0; sa=0; sb=0; a=1/40320.0; b=a/9
      k=0
      do while (abs(sa).ne.sa1)
        sa1=abs(sa)
        sa=sa+a; a=-a*x2/(2*k+ 9)/(2*k+10)
        sb=sb+b; b=-b*x2/(2*k+10)/(2*k+11); k=k+1
      enddo
      w=sa/sb
      end function w1
```

При вызове из СИ-программы их имена завершаем значком подчёркивания

```
#include <stdio.h>
#include <math.h>
float  w0_(float *);
float  w1_(float *);
int main()
{  float t0, tn, t, ht, x, r0, r1; int n, i; FILE *ninp, *nres;
  ninp=fopen("input","r");
  if (ninp==NULL) { printf("Неудача открытия файла  input!\n"); return 1;}
  nres=fopen("result","w");
  if (nres==NULL) { printf("Неудача открытия файла result!\n"); return 2;}
   fscanf(ninp,"%e %e %d", &t0, &tn, &n);
   fprintf(nres," #  t0=%e  tn=%e  n=%i\n", t0, tn, n);
   ht=(tn-t0)/n;
   fprintf(nres," # %2s %10s %15s %15s\n","i","t","r0","r1");
   for (i=1; i<=n+1;i++)
     { t=t0+ht*(i-1);  x=exp(-t);
       r0=w0_(&x);
       r1=w1_(&x);
       fprintf(nres,"%5i %15.7e %15.7e %15.7e\n", i, t, r0, r1);
     }
   fclose(ninp); fclose(nres);
   return 0;
}
```

Забавно, что в СИ-результат **r0**, полученный ФОРТРАН-функцией, отличен от результатов ФОРТРАНа и C++, хотя и те, и другие результаты абсолютно неверны. Результат, полученный вызовом из **C** ФОРТРАН-функций:

```
#  t0=1.000000e+00  tn=2.000000e+00  n=10
#  i          t              r0             r1
   1   1.0000000e+00   0.0000000e+00   8.9975405e+00
   2   1.1000000e+00   3.5000000e+00   8.9979858e+00
   3   1.2000000e+00   1.0434783e+00   8.9983501e+00
   4   1.3000000e+00   1.7142858e-01   8.9986496e+00
   5   1.4000000e+00   3.0000000e+00   8.9988937e+00
   6   1.5000000e+00   5.2631581e-01   8.9990950e+00
   7   1.6000000e+00  -3.1460676e-01   8.9992599e+00
   8   1.7000000e+00  -5.7142860e-01   8.9993935e+00
   9   1.8000000e+00  -1.1739130e+00   8.9995041e+00
  10   1.9000001e+00   5.7971016e-02   8.9995947e+00
  11   2.0000000e+00   1.0109891e+00   8.9996672e+00
```

Результат, полученный вызовом из **C**++ ФОРТРАН-функций:

```
#  t0=1.000000e+00  tn=2.000000e+00  n=10
#  i          t              r0             r1
   1   1.0000000e+00   0.0000000e+00   8.9975405e+00
   2   1.1000000e+00   3.5000000e+00   8.9979858e+00
   3   1.2000000e+00   1.0434783e+00   8.9983501e+00
   4   1.3000000e+00   1.7142858e-01   8.9986496e+00
   5   1.4000000e+00   3.0000000e+00   8.9988937e+00
   6   1.5000000e+00   5.2631581e-01   8.9990950e+00
   7   1.6000000e+00  -3.1460676e-01   8.9992599e+00
   8   1.7000000e+00  -5.7142860e-01   8.9993935e+00
   9   1.8000000e+00  -1.1739130e+00   8.9995041e+00
  10   1.9000001e+00   5.7971016e-02   8.9995947e+00
```

Результат, полученный компилятором **C**++

```
#  t0=1.000000e+00  tn=2.000000e+00  n=10            // Старый результат,
#  i          t              r0             r1      // полученный
   1   1.0000000e+00   5.7829318e+00   8.9975405e+00 // после отработки
   2   1.1000000e+00   6.6944509e+00   8.9979858e+00 // исполнимого файла,
   3   1.2000000e+00   6.4263763e+00   8.9983501e+00 // собранного из
   4   1.3000000e+00   7.2002325e+00   8.9986496e+00 // объектных файлов
   5   1.4000000e+00   3.8472526e+00   8.9988937e+00 // w0.o и w1.o,
   6   1.5000000e+00   1.7917061e+00   8.9990950e+00 // полученных C++
   7   1.6000000e+00   3.4897115e+00   8.9992599e+00 // компилятором
   8   1.7000000e+00   3.1662014e+00   8.9993935e+00 // при компиляции
   9   1.8000000e+00   3.1587539e+00   8.9995041e+00 // файлов w0.cpp и
  10   1.9000000e+00   2.9607465e+00   8.9995947e+00 //        w1.cpp
```

Заметим, что при вызове из СИ-программы ФОРТРАН-функций указание

```
extern "C" { float w0_(float *); float w1_(float *); }
```

приводит к синтаксической ошибке.

5.1.4 ФОРТРАН-вызовы С-функций

```
#include <math.h>
double wd0_(double *y)                                    //    Файл w0.c
{double x2, a, b, x;
      x=*y; x2=x*x;
      a=cos(x) - 1 + x2* 0.5l*(1l - x2/12l * ( 1l - x2 / 30l));
      b=sin(x)/x - 1+x2/6l * (1l-x2/20l*(1l-x2/42l));
      return(a/b);
}


#include <math.h>
double wd1_(double *y)                                     //    Файл w1.c
{double x,x2, sa1, sa, sb, a, b;
 int k, k2;
 x=*y; x2=x*x; sa1=1.0; sa=0; sb=0; a=1/40320.0; b=a/9; k=k2=0;
 while (fabs(sa)!=sa1)
   { sa1=fabs(sa); sa=sa+a; a=-a*x2/(k2+ 9)/(k2+10);
                   sb=sb+b; b=-b*x2/(k2+10)/(k2+11); k=k+1; k2=2*k;
   }
 return (sa/sb);
}


      program tmixfc30                        !    Файл  tmixfc30.f
      implicit none
      real*8 wd0, wd1, t0, tn, t, ht, x, r0, r1
      integer ninp, nres
      integer n, i
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100)  t0, tn
      read(ninp,101) n
      write(nres, 1099) t0, tn, n
      ht=(tn-t0)/n
      write(nres,1100)
      do i=0,n
          t=(t0+i*ht); x=dexp(-t); r0=wd0(x); r1=wd1(x)
          write(nres,1001) i, t,  r0, r1
      enddo
      close(nres)
  100 format(d15.7)
  101 format(i15)
 1099 format(1x,' #   t0=',d23.15,2x,'tn=',d23.15,2x,'n=',i3)
 1100 format(1x,' #',2x,'i',14x,'t',21x,'w0',21x,'w1')
 1001 format(1x,i5,2x,d23.15,d23.15,d23.15)
      end
```

5.1.5 C-вызовы C++-функций

```
#include <stdio.h>
float w0(float); float w1(float);       //  Файл  tsfs2p30.c
int main()
{ float t0, tn, t, ht, x, r0, r1; int n, i; FILE *ninp, *nres;
  ninp=fopen("input","r");
  if (ninp==NULL) { printf("Неудача при открытии файла input !!!\n"); return 1;}
  nres=fopen("result","w");
  if (nres==NULL){ printf("Неудача при открытии файла result !!!\n"); return 2;}
  fscanf(ninp,"%e %e %d", &t0, &tn, &n);
  fprintf(nres," #  t0=%e  tn=%e  n=%i\n", t0, tn, n);
  ht=(tn-t0)/n;
  fprintf(nres," # %2s %10s %15s %15s\n","i","t","r0","r1");
  for (i=1; i<=n;i++)
    { t=t0+ht*(i-1); x=exp(-t); r0=w0(x); r1=w1(x);
      fprintf(nres,"%5i %15.7e %15.7e %15.7e\n", i, t, r0, r1);
    }
   fclose(ninp); fclose(nres); return 0;
}
#include <cmath>
extern "C" float w0(float);
float w0(float x)                                    //    Файл w0.cpp
{float x2, a, b;
  x2=x*x; a=cos(x) - 1 + x2* 0.5*(1 - x2/12 * ( 1 - x2 / 30));
          b=sin(x)/x - 1+x2/6 * (1-x2/20*(1-x2/42));
  return(a/b);
}
#include <cmath>
extern "C" float w1(float);
float w1(float x)                                     //    Файл w1.cpp
{float x2, sa1, sa, sb, a, b;  int k, k2;
 x2=x*x; sa1=1.0; sa=0; sb=0; a=1/40320.0; b=a/9;  k=k2=0;
 while (fabs(sa)!=sa1)
   { sa1=fabs(sa);  sa=sa+a; a=-a*x2/(k2+ 9)/(k2+10);
                    sb=sb+b; b=-b*x2/(k2+10)/(k2+11); k=k+1; k2=2*k;
   } return (sa/sb);
}
```

```
$ gcc -c tsfs2p30.c
$ c++ -c w0.cpp
$ c++ -c w1.cpp
$ gcc tsfs2p30.o w0.o w1.o -lstdc++
$ ./a.out
 #  t0=1.000000e+00  tn=2.000000e+00  n=10
 #  i          t              r0              r1
    1   1.0000000e+00   5.7829318e+00   8.9975405e+00
    2   1.1000000e+00   6.6944509e+00   8.9979858e+00
    3   1.2000000e+00   6.4263763e+00   8.9983501e+00
    4   1.3000000e+00   7.2002325e+00   8.9986496e+00
    5   1.4000000e+00   3.8472526e+00   8.9988937e+00
    6   1.5000000e+00   1.7917061e+00   8.9990950e+00
    7   1.6000000e+00   3.4897115e+00   8.9992599e+00
    8   1.7000000e+00   3.1662014e+00   8.9993935e+00
    9   1.8000000e+00   3.1587539e+00   8.9995041e+00
   10   1.9000000e+00   2.9607465e+00   8.9995947e+00
```

5.1.6 C++-вызовы C-функций

Обратное обращение из главной программы программы на **C++**

```
#include <iostream>  // Программа демонстрирует применение некоторых
#include <iomanip>   // манипуляторов C++ из класса для форматирования
#include <fstream>   // данных, выводимых в файл.
#include <cmath>
using namespace std;
extern "C" { float w0(float); float w1(float); }
int main()
{ float t0, tn, t, ht, x, r0, r1;
  int n, i;
  ifstream ninp("input");
  if (!ninp) { printf("Неудача при открытии файла input  !!!\n"); return 1;}
  ofstream nres("result");
  if (!nres) { printf("Неудача при открытии файла result !!!\n"); return 2;}
  ninp >> t0 >> tn >> n;
  nres << scientific;
  nres <<" #  t0="<< t0 <<"  tn=" << tn << "  n="<< n << endl; ht=(tn-t0)/n;
  nres << setw( 5) << "i" ; nres << setw(10)<<"t";
  nres << setw(15) <<"r0"; nres << setw(15)<< "r1\n";
 for (i=1; i<=n;i++)
   { t=t0+ht*(i-1); x=exp(-t); r0=w0(x); r1=w1(x);
     nres << setw( 5)<< i;
     nres << setw(15)<< t;
     nres << setw(15)<<  r0;
     nres << setw(15)<< r1 <<endl;
    }
   ninp.close(); nres.close(); return 0;
}
```

к функциям **w0** и **w1**

```
float w0(float x)                                    //    Файл w0.c
{float x2, a, b;
      x2=x*x;
      a=cos(x) - 1 + x2* 0.5*(1 - x2/12 * ( 1 - x2 / 30));
      b=sin(x)/x - 1+x2/6 * (1-x2/20*(1-x2/42));
      return(a/b);
}

float w1(float x)                                     //    Файл w1.c
{ float x2, sa1, sa, sb, a, b; int k, k2;
 x2=x*x; sa1=1.0; sa=0; sb=0; a=1/40320.0; b=a/9; k=k2=0;
 while (fabs(sa)!=sa1)
   { sa1=fabs(sa); sa=sa+a; a=-a*x2/(k2+ 9)/(k2+10);
                   sb=sb+b; b=-b*x2/(k2+10)/(k2+11);
     k=k+1; k2=2*k;
   }
 return (sa/sb);
}
```

написанным на **C**, дает, очевидно, тот же результат

```
$ c++ -c tsfs2p30.cpp
$ gcc -c w0.c
$ gcc -c w1.c
$ c++ tsfs2p30.o w0.o w1.o -lstdc++ -lm
$ ./a.out
$
 #  t0=1.000000e+00  tn=2.000000e+00  n=10
    i         t             r0            r1
    1   1.000000e+00   5.782932e+00   8.997540e+00
    2   1.100000e+00   6.694451e+00   8.997986e+00
    3   1.200000e+00   6.426376e+00   8.998350e+00
    4   1.300000e+00   7.200233e+00   8.998650e+00
    5   1.400000e+00   3.847253e+00   8.998894e+00
    6   1.500000e+00   1.791706e+00   8.999095e+00
    7   1.600000e+00   3.489712e+00   8.999260e+00
    8   1.700000e+00   3.166201e+00   8.999393e+00
    9   1.800000e+00   3.158754e+00   8.999504e+00
   10   1.900000e+00   2.960747e+00   8.999595e+00
```

5.2 Уяснение ситуации

Все приведённые выше примеры демонстрируют организацию совмещённого программирования применительно к простым случаям, когда на **gcc**-компиляторах можно обойтись С-типами **int**, **float**, **double** и ФОРТРАН-типами **integer**, **real(4)** и **real(8)**.

Современные ФОРТРАН и С/С++ имеют много разновидостей целого типа, которые и обозначаются по разному, и, более того, количество этих разновидностей различно. Поэтому, если при **gfortran**-сборке исполнимого файла участвуют объектные файлы функций, написанных на современных версиях С/С++ (т.е. полученные современными **gcc**/**g++**-компиляторами), то в исходном ФОРТРАН-тексте важно правильно указать требуемый тип разновидности аргумента СИ-функции.

Современный ФОРТРАН использует для этого встроенный модуль

ISO_C_BINDING

(*International Standard Organization* — привязка ISO C), в котором определены некоторые функции (в частности, и **C_sizeof**) и установлено соответствие ФОРТРАН- и СИ-типов посредством соответствующих именованных констант.

Подробнее о **ISO_C_BINDING**, о совмещённом (смешанном или *интероперабельном*) программировании узнаем в четвёртом семестре

newpage

5.3 О чем узнали из параграфа 5? (кратко)

1. Совмещенное (или смешанное) программирование – это генерация исполнимого (загрузочного) кода посредством компоновки объектных кодов, полученных компиляцией исходников, написанных на разных языках программирования.

2. Задача совмещенного программирования возникает, когда необходимо в программе, написанной на одном языке обратится к ряду нужных подпрограмм, исходные коды которых написаны на другом.

3. Из-за возможных различных принципов распределения памяти, схем передачи параметров, неких внутренних условностей и правил умолчаний разных компиляторов для получения соответствующих друг другу объектных кодов требуется определенная модификация исходных.

4. Эта модификация в рамках возможностей **GNU**-компилятора компиляторов **gcc** не представляет затруднений, хотя и требует активного владения некоторыми базовыми понятиями совмещаемых языков.

5. В частности, узнали, что компиляторы **g77** и **gfortran** при создании имен объектных кодов функций и подпрограмм добавляют к их исходным именам в качестве окончания **знак подчеркивания**.

6. Поэтому, если **ФОРТРАН**-программа *вызывает* объектные коды функций **C** или **C++** функций, то в их исходных **C**- и **C++**-текстах имена функций необходимо завершать знаком подчеркивания.

7. Соответственно имя каждой ФОРТРАН-функции в операторах **вызова** из **C**- или **C++** программы также должно завершаться значком подчеркивания.

8. При описании **C++** функций, объектные коды которых подключаются к C- или ФОРТРАН-программе, необходимо указать что взаимодействие с вызывающей программой должно проводиться в соответствии с порядком связывания языка **C**. Именно на это и указывает заключенная в двойные кавычки литера **C** после слова **extern**.

9. Не забываем, что в ФОРТРАНе аргументы функций и подпрограмм всегда передаются по адресу, а в C – всегда по значению.

10. Поэтому, когда из C вызывается ФОРТРАН-функция, то (помимо вышеупомянутой модификации имени) при описании ее прототипа все формальные аргументы ее необходимо оформить **указателями** на соответствующий тип данных, а все фактические предварить значком операции получения адреса.

11. Организуя совмещённое (или, как иногда говорят, интероперабельное) программирование на современных версиях **GNU**-ФОРТРАНа используем встроенный модуль **ISO_C_BINDING**.

6 Приложение I: Лабораторная работа

6.1 Введение.

Лабораторная работа нацелена на закрепление навыка анализа и (при необходимости) соответствующего преобразования формул, предложенных для программирования. Часто формула, определяющая функцию, получаемую в процессе исследовательской работы, не всегда применима для табулирования функции по всей области определения.

Например, функцию $\dfrac{\sin(x) - x + x^3/6}{x^5}$ по приведенной явной формуле невозможно верно вычислить на ЭВМ в области $x \leq 10^{-3}$ из-за катастрофической потери точности, возникающей вследствие вычитания фактически равных (даже на восьмибайтовой разрядной сетке) слагаемых $\sin x$ и $x - x^3/6$.

В указанной области расчёт выгодно вести по разложению для $\sin x$, начиная суммирование с его третьего слагаемого. Таким образом, аргументация, что при наличии современных пакетов вычислительной математики не нужно иметь предствления об алгоритмах расчёта элементарных функций – безосновательна.

В то же время полезно понимать почему при $x \gg 1$ условие малости последнего учтенного слагаемого сходящегося знакопеременного ряда не всегда обеспечивает точность расчёта, гарантированную известной теоремой матанализа.

Причина заключается в том, что оценки математического анализа делаются в предположении об идельном (бесконечнозначном) арифметическом устройстве, ведущим расчёт, в то время как, арифметическое устройство реальной ЭВМ объективно имеет конечноразрядную сетку (7-8 десятичных цифр при одинарной точности и 15-16 при удвоенной).

Так что, если среди слагаемых ряда встречаются такие, которые для записи целой части своего значения требуют, например, 15 десятичных цифр, то на дробные разряды мантиссы места не остаётся. А тогда результат суммирования (допустим меньший единицы) в принципе невозможно получить верным, поскольку его разряды отсутствовали в ячейках, хранящих упомянутые большие значения. Другими словами, погрешность ограничения суммирования (т.е. малость последнего учтённого слагаемого) достигнута, в то время как вычислительная погрешность результата на много порядков больше, что полностью обесценивает временные затраты на проделанное суммирование.

6.2 Условия задач

6.2.1 Вариант 1. Тема: $\sin(x) - 2\sin(x/2)\ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \Big(\sin \mathbf{x} - \mathbf{2} * \sin \frac{\mathbf{x}}{\mathbf{2}} + \frac{\mathbf{x^3}}{\mathbf{8}}\Big(\mathbf{1} - \frac{\mathbf{x^2}}{\mathbf{16}}\Big)\Big)\frac{\mathbf{5120}}{\mathbf{x^7}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	10^{-3}	$2 \cdot 10^{-3}$	10
3)	50	150	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.001$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Поиск $\lim\limits_{x\to 0} f(x)$, разложения, и проведение расчёта по исходной формуле на базе соответствующего **maxima**-скрипта, выводящего результат в файл.
6. Вывод рекуррентной формулы расчёта его очередного слагаемого.
7. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
8. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
9. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
10. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
11. Заключение (о чём узнали, решая данную задачу?)

Примечание:

$$\sin x = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

6.2.2 Вариант 2. Тема: $\sin(x) - 3\sin(x/3) + \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \Big(\sin \mathbf{x} - \mathbf{3}\sin\frac{\mathbf{x}}{\mathbf{3}} + \frac{\mathbf{2x^3}}{\mathbf{27}}\Big(\mathbf{1} - \frac{\mathbf{x^2}}{\mathbf{9}}\Big)\Big)\frac{\mathbf{65610}}{\mathbf{13x^7}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	0.01	0.011	10
3)	50	150	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.01$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Поиск $\lim\limits_{x\to 0} f(x)$, разложения, и проведение расчёта по исходной формуле на базе соответствующего **maxima**-скрипта, выводящего результат в файл.
6. Вывод рекуррентной формулы расчёта его очередного слагаемого.
7. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
8. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
9. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
10. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
11. Заключение (о чём узнали, решая данную задачу?)

Примечание:

$$\sin x = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

6.2.3 Вариант 3. Тема: $\cos(x) - \cos(x/2) + \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \Big(\cos \mathbf{x} - \cos \frac{\mathbf{x}}{\mathbf{2}} + \frac{\mathbf{x^2}}{\mathbf{8}}\Big(\mathbf{3} - \frac{\mathbf{5x^2}}{\mathbf{16}}\Big)\Big)\frac{\mathbf{5120}}{\mathbf{7x^6}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	0.001	0.002	10
3)	50	150	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.001$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Поиск $\lim\limits_{x\to 0} f(x)$, разложения, и проведение расчёта по исходной формуле на базе соответствующего **maxima**-скрипта, выводящего результат в файл.
6. Вывод рекуррентной формулы расчёта его очередного слагаемого.
7. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
8. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
9. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
10. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
11. Заключение (о чём узнали, решая данную задачу?)

Примечание:

$$\cos x = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!}$$

6.2.4 Вариант 4. Тема: $\exp(-0.5x^2) - \exp(-x^2) - \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x) = \left(e^{-\frac{x^2}{2}} - e^{-x^2} - \frac{x^2}{2}\left(1 - \frac{x^2}{4}\left(3 - \frac{7x^2}{6}\right)\right)\right)\frac{128}{5x^8}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	0.001	0.002	10
3)	10	20	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.001$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Поиск $\lim\limits_{x\to 0} f(x)$, разложения, и проведение расчёта по исходной формуле на базе соответствующего **maxima**-скрипта, выводящего результат в файл.
6. Вывод рекуррентной формулы расчёта его очередного слагаемого.
7. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
8. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
9. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
10. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
11. Заключение (о чём узнали, решая данную задачу?)

Примечание:

$$\exp(-x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^k}{k!}$$

6.2.5 Вариант 5. Тема: $\exp(-x^2) - \exp(-0.25x^2) + \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \Big(\mathbf{e^{-x^2}} - \mathbf{e^{-\frac{x^2}{4}}} + \frac{\mathbf{3x^2}}{\mathbf{4}}\Big(\mathbf{1} - \frac{\mathbf{5x^2}}{\mathbf{8}}\Big)\Big)\frac{\mathbf{128}}{\mathbf{21x^6}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	0.0005	0.0015	10
3)	7	7.5	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.0005$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Поиск $\lim\limits_{x\to 0} f(x)$, разложения, и проведение расчёта по исходной формуле на базе соответствующего **maxima**-скрипта, выводящего результат в файл.
6. Вывод рекуррентной формулы расчёта его очередного слагаемого.
7. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
8. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
9. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
10. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
11. Заключение (о чём узнали, решая данную задачу?)

Примечание:

$$\exp(-x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^k}{k!}$$

6.2.6 Вариант 6. Тема: $e^{-x^2} - \cos x + \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \Big(\mathbf{e^{-x^2}} - \cos\mathbf{x} + \frac{\mathbf{x^2}}{\mathbf{2}}\Big(\mathbf{1} - \frac{\mathbf{11x^2}}{\mathbf{12}}\Big)\Big)\frac{\mathbf{720}}{\mathbf{119x^6}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	0.0005	0.0007	10
3)	7	9	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.0005$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Поиск $\lim\limits_{x\to 0} f(x)$, разложения, и проведение расчёта по исходной формуле на базе соответствующего **maxima**-скрипта, выводящего результат в файл.
6. Вывод рекуррентной формулы расчёта его очередного слагаемого.
7. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
8. Ответ на вопрос: “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
9. Ответ на вопрос: “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
10. Верные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
11. Заключение (о чём узнали, решая данную задачу?)

Примечание:

$$\exp(-x) = \sum_{k=0}^{\infty}(-1)^k\frac{x^k}{k!}$$

$$\cos x = \sum_{k=0}^{\infty}(-1)^k\frac{x^{2k}}{(2k)!}$$

6.2.7 Вариант 7. Тема: $e^{-x^4} - e^{-x^4/9} + \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \left(\mathbf{e^{-x^4} - e^{-x^4/9} + \frac{8x^4}{9}\left(1 - \frac{5x^4}{9}\right)}\right)\frac{\mathbf{2187}}{\mathbf{364x^{12}}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	0.025	0.035	10
3)	3	4	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.025$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Поиск $\lim\limits_{x\to 0} f(x)$, разложения, и проведение расчёта по исходной формуле на базе соответствующего **maxima**-скрипта, выводящего результат в файл.
6. Вывод рекуррентной формулы расчёта его очередного слагаемого.
7. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
8. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
9. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
10. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
11. Заключение (о чём узнали, решая данную задачу?)

Примечание:

$$\exp(-x) = \sum_{k=0}^{\infty}(-1)^k\frac{x^k}{k!}$$

6.2.8 Вариант 8. Тема: $\cos(x) - \frac{\sin(x)}{x} + \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \Big(\cos \mathbf{x} - \frac{\sin \mathbf{x}}{\mathbf{x}} + \frac{\mathbf{x^2}}{\mathbf{3}}\Big(\mathbf{1} - \frac{\mathbf{x^2}}{\mathbf{10}}\Big)\Big)\frac{\mathbf{840}}{\mathbf{x^6}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	0.001	0.002	10
3)	50	60	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.001$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Поиск $\lim\limits_{x\to 0} f(x)$, разложения, и проведение расчёта по исходной формуле на базе соответствующего **maxima**-скрипта, выводящего результат в файл.
6. Вывод рекуррентной формулы расчёта его очередного слагаемого.
7. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
8. Ответ на вопрос: “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
9. Ответ на вопрос: “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
10. Верные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
11. Заключение (о чём узнали, решая данную задачу?)

Примечание:

$$\cos x = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!}$$

$$\sin x = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

6.2.9 Вариант 9. Тема: $sin^2(x) - x^2 + \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \Big(\sin^{\mathbf{2}}\mathbf{x} - \mathbf{x^2}\Big(\mathbf{1} - \frac{\mathbf{x^2}}{\mathbf{3}}\Big)\Big)\frac{\mathbf{45}}{\mathbf{2x^6}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	10^{-4}	$2 \cdot 10^{-4}$	10
3)	25	30	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.0001$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Поиск $\lim\limits_{x\to 0} f(x)$, разложения, и проведение расчёта по исходной формуле на базе соответствующего **maxima**-скрипта, выводящего результат в файл.
6. Вывод рекуррентной формулы расчёта его очередного слагаемого.
7. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
8. Ответ на вопрос:

   "Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?"
9. Ответ на вопрос:

   "Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?"
10. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
11. Заключение (о чём узнали, решая данную задачу?)

Примечание:

$$\sin^2 x = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{2^{2k-1} x^{2k}}{(2k)!}$$

6.2.10 Вариант 10. Тема: $\sin^3(x) - x^3 + \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \Big(\sin^{\mathbf{3}}\mathbf{x} - \mathbf{x^3}\Big(\mathbf{1} - \frac{\mathbf{x^2}}{\mathbf{2}}\Big)\Big)\frac{\mathbf{120}}{\mathbf{13x^7}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	10^{-4}	$2 \cdot 10^{-4}$	10
3)	15	20	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.0001$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Поиск $\lim\limits_{x\to 0} f(x)$, разложения, и проведение расчёта по исходной формуле на базе соответствующего **maxima**-скрипта, выводящего результат в файл.
6. Вывод рекуррентной формулы расчёта его очередного слагаемого.
7. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
8. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
9. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
10. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
11. Заключение (о чём узнали, решая данную задачу?)

Примечание:

$$\sin^3 x = \frac{1}{4}\sum_{k=1}^{\infty}(-1)^{k+1}\frac{3^{2k+1}-3}{(2k+1)!}x^{2k+1}$$

6.2.11 Вариант 11. Тема: $x \cdot \exp(-x^2) - \sin(x) + \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \Big(\mathbf{x{\cdot}e^{-x^2}} - \sin\mathbf{x} + \frac{\mathbf{5x^3}}{\mathbf{6}}\Big(\mathbf{1 - 0.59x^2}\Big)\Big)\frac{\mathbf{5040}}{\mathbf{839x^7}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	0.0001	0.0011	10
3)	7	7.5	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.0001$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Поиск $\lim\limits_{x\to 0} f(x)$, разложения, и проведение расчёта по исходной формуле на базе соответствующего **maxima**-скрипта, выводящего результат в файл.
6. Вывод рекуррентной формулы расчёта его очередного слагаемого.
7. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
8. Ответ на вопрос: “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
9. Ответ на вопрос: “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
10. Верные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
11. Заключение (о чём узнали, решая данную задачу?)

Примечание:

$$\exp(-x) = \sum_{k=0}^{\infty}(-1)^k\frac{x^k}{k!}$$

$$\sin x = \sum_{k=0}^{\infty}(-1)^k\frac{x^{2k+1}}{(2k+1)!}$$

6.2.12 Вариант 12. Тема: $\frac{0.5\sqrt{\pi}\cdot erf(x)-\sin x}{x}+\ \cdots$

Оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)}=\left(\frac{\frac{\sqrt{\pi}}{\mathbf{2}}\mathbf{erf(x)}-\sin\mathbf{x}}{\mathbf{x}}+\frac{\mathbf{x^2}}{\mathbf{6}}\Big(\mathbf{1}-\frac{\mathbf{11x^2}}{\mathbf{20}}\Big)\right)\frac{\mathbf{720}}{\mathbf{17x^6}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	0.002	0.003	10
3)	7	7.5	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$ и оценивание погрешности расчёта по исходной формуле при $x = 0.002$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
3. Получение разложения $f(x)$ в ряд Маклорена.
4. Поиск $\lim\limits_{x\to 0} f(x)$, разложения, и проведение расчёта по исходной формуле на базе соответствующего **maxima**-скрипта, выводящего результат в файл.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос: “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
8. Ответ на вопрос: “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
9. Верные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

Примечание:

$$\frac{\sqrt{\pi}}{2}erf(x)=\int_0^x e^{-t^2}dt=\sum_{k=0}^{\infty}(-1)^k\frac{x^{2k+1}}{k!(2k+1)}$$

$$\sin x=\sum_{k=0}^{\infty}(-1)^k\frac{x^{2k+1}}{(2k+1)!}$$

6.2.13 Вариант 13. Тема: $0.5\sqrt{\pi}\cdot\dfrac{erf(x)-2erf(x/2)}{x}+\ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)}=\left(\frac{\sqrt{\pi}}{\mathbf{2}}\cdot\frac{\mathbf{erf(x)-2erf\left(\frac{x}{2}\right)}}{\mathbf{x}}+\frac{\mathbf{x^2}}{\mathbf{4}}\Big(\mathbf{1}-\frac{\mathbf{3x^2}}{\mathbf{8}}\Big)\right)\frac{\mathbf{128}}{\mathbf{3x^6}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	0.0025	0.0035	10
3)	7	7.5	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.0025$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Поиск $\lim\limits_{x\to 0} f(x)$, разложения, и проведение расчёта по исходной формуле на базе соответствующего **maxima**-скрипта, выводящего результат в файл.
6. Вывод рекуррентной формулы расчёта его очередного слагаемого.
7. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
8. Ответ на вопрос: “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
9. Ответ на вопрос: “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
10. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
11. Заключение (о чём узнали, решая данную задачу?)

Примечание:

$$\frac{\sqrt{\pi}}{2}erf(x)=\int_0^x e^{-t^2}dt=\sum_{k=0}^{\infty}(-1)^k\frac{x^{2k+1}}{k!(2k+1)}$$

6.2.14 Вариант 14. Тема: $0.5\sqrt{\pi}erf(x) - x \cdot \cos x + \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \left(\frac{\sqrt{\pi}}{\mathbf{2}} \cdot \mathbf{erf(x)} - \mathbf{x} \cdot \cos \mathbf{x} - \frac{\mathbf{x^3}}{\mathbf{6}} \left(\mathbf{1} + \frac{\mathbf{7x^2}}{\mathbf{20}} \right) \right) \frac{\mathbf{5040}}{\mathbf{113x^7}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	0.0010	0.0050	10
3)	7	7.5	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x \to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.001$ относительно найденного $\lim\limits_{x \to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
8. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
9. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

Примечание:

$$\frac{\sqrt{\pi}}{2} erf(x) = \int_0^x e^{-t^2} dt = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{k!(2k+1)}$$

$$\cos x = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!}$$

6.2.15 Вариант 15. Тема: $E_1(x) - E_1(0.5x) + \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x) = \left(E_1(x) - E_1\left(\frac{x}{2}\right) + \ln(2) - \frac{x}{2}\left(1 - \frac{3x^2}{8}\right)\right) \frac{144}{7x^3}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	0.0001	0.0012	10
3)	50	60	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.0001$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
8. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
9. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

Примечание: $E_1(x) = \int_1^\infty e^{-xt}\frac{dt}{t}$ представима в виде

$$E_1(x) = -\gamma - ln(x) + \sum_{k=1}^{\infty}(-1)^{k+1}\frac{x^k}{k!k} \quad ,$$

где $\gamma = 0.57721566490153286061$ — постоянная Эйлера (EulerGamma).

$E_1(x)$ — первая интнгрально-показательная функция, часто встречающаяся в модельных задачах теории переноса излучения.

Высокоточный расчёт выражения $E_1(x) + \gamma + ln(x)$ при $0 \geq x \leq 8$ можно выполнить, например, функцией **e141a(x)**, а при $x \geq 5$ — функцией **e141c(x)** (см. раздел 5).

6.2.16 Вариант 16. Тема: $1 - e^{-x} - E_1(x) - \gamma - ln(x) + \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \left(\mathbf{1 - e^{-x} - E_1(x) - \gamma - ln(x) + \frac{x^2}{4}\left(1 - \frac{4x}{9}\right)}\right)\frac{\mathbf{32}}{\mathbf{x^4}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	0.0010	0.0020	10
3)	49	50	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.001$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
8. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
9. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

Примечание: $E_1(x) = \int_1^\infty e^{-xt}\frac{dt}{t}$ представима в виде

$$E_1(x) = -\gamma - ln(x) + \sum_{k=1}^{\infty}(-1)^{k+1}\frac{x^k}{k!k} \quad ,$$

где $\gamma = 0.57721566490153286061$ — постоянная Эйлера (EulerGamma).

$E_1(x)$ — первая интнгрально-показательная функция, часто встречающаяся в модельных задачах теории переноса излучения.

Высокоточный расчёт выражения $E_1(x) + \gamma + ln(x)$ при $0 \geq x \leq 8$ можно выполнить, например, функцией **e141a(x)**, а при $x \geq 5$ — функцией **e141c(x)** (см. раздел 5).

6.2.17 Вариант 17. Тема: $C(x)\sqrt{\frac{\pi}{2x}} - \cos x \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \left(\mathbf{C(x)}\sqrt{\frac{\pi}{\mathbf{2x}}} - \mathbf{cos(x)} - \mathbf{x^2}\left(\mathbf{0.4} - \frac{\mathbf{x^2}}{\mathbf{27}}\right)\right)\frac{\mathbf{780}}{\mathbf{x^6}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	0.005	0.006	10
3)	50	60	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.005$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
8. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
9. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

Примечание: $C(x) = \dfrac{1}{\sqrt{2\pi}} \cdot \int_0^x \cos t \dfrac{dt}{\sqrt{t}}$ представим в виде

$$C(x) = \sqrt{\frac{2x}{\pi}} \cdot \sum_{k=0}^{\infty} (-1)^k \frac{x^2 k}{(2k)! \cdot (4k+1)}$$

$C(x)$ — косинус-интеграл Френеля.

Для высокоточного расчёта $C(x)$ можно использовать либо имеющиеся в наличии в соответствующих библиотеках функции, либо при отсутствии последних приводимые в разделе 5 функцию **cfren1(x)** и подпрограмму csfren6(x,dre,dim,nout). Первая находит 2C(x) при $0 \geq x \leq 8$, а вторая при $x \geq 5$ вещественную (dre) и мнимую (dim) части интеграла

$$\int_x^\infty \frac{e^{-it}}{\sqrt{t}} dt \quad ,$$

зная которые можно вычислить для указанной полуоси косинус- $(C(x))$ и $(S(x))$ синус-интегралы Френеля.

6.2.18 Вариант 18. Тема: $\frac{1}{4}\sin x - \frac{3}{4}S(x)\sqrt{\frac{\pi}{2x}} + \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \left(\frac{\mathbf{1}}{\mathbf{4}}\sin\mathbf{x} - \frac{\mathbf{3}}{\mathbf{4}}\mathbf{S(x)}\sqrt{\frac{\pi}{\mathbf{2x}}} + \frac{\mathbf{x^3}}{\mathbf{42}}\Big(\mathbf{1} - \frac{\mathbf{7x^2}}{\mathbf{110}}\Big)\right)\frac{\mathbf{25200}}{\mathbf{x^7}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	0.001	0.002	10
3)	50	60	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.001$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
8. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
9. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

Примечание: $S(x) = \dfrac{1}{\sqrt{2\pi}} \cdot \displaystyle\int_0^x \sin t \frac{dt}{\sqrt{t}}$ представим в виде

$$S(x) = \sqrt{\frac{2x}{\pi}} \cdot \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)! \cdot (4k+3)}$$

$S(x)$ — синус-интеграл Френеля.

Для высокоточного расчёта $S(x)$ можно использовать либо имеющиеся в наличии в соответствующих библиотеках функции, либо при отсутствии последних приводимые в разделе 5 функцию **sfren1(x)** и подпрограмму csfren6(x,dre,dim,nout). Первая находит 2S(x) при $0 \geq x \leq 8$, а вторая при $x \geq 5$ вещественную (dre) и мнимую (dim) части интеграла

$$\int_x^\infty \frac{e^{-it}}{\sqrt{t}} dt \quad ,$$

зная которые можно вычислить для указанной полуоси косинус- $(C(x))$ и $(S(x))$ синус-интегралы Френеля.

6.2.19 Вариант 19. Тема: $C(x)\sqrt{\frac{\pi}{2x}} - S(x)\sqrt{\frac{\pi}{2x^3}}\ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \left(\mathbf{C(x)}\sqrt{\frac{\pi}{\mathbf{2x}}} - \mathbf{S(x)}\sqrt{\frac{\pi}{\mathbf{2x^3}}} - \frac{\mathbf{2}}{\mathbf{3}} + \frac{\mathbf{8x^2}}{\mathbf{105}}\right)\frac{\mathbf{5940}}{\mathbf{23x^4}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	2.0	10
2)	0.0005	0.0010	10
3)	50	60	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.0005$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
8. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
9. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

Примечание: $C(x) = \dfrac{1}{\sqrt{2\pi}} \cdot \int_0^x \cos t \dfrac{dt}{\sqrt{t}}$ представим в виде

$$C(x) = \sqrt{\frac{2x}{\pi}} \cdot \sum_{k=0}^{\infty} (-1)^k \frac{x^2 k}{(2k)! \cdot (4k+1)}$$

$S(x) = \frac{1}{\sqrt{2\pi}} \cdot \int_0^x \sin t \frac{dt}{\sqrt{t}}$ представим в виде

$$S(x) = \sqrt{\frac{2x}{\pi}} \cdot \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)! \cdot (4k+3)}$$

$C(x)$ — косинус-интеграл Френеля. $S(x)$ — синус-интеграл Френеля.

Для высокоточного расчёта $C(x)$ и $S(x)$ можно использовать либо имеющиеся в наличии в соответствующих библиотеках функции, либо при отсутствии последних приводимые в разделе 5 функцию **cfren1(x)** и **sfren1(x)**, а также подпрограмму csfren6(x,dre,dim,nout). Первые две находят соответственно 2C(x) и 2S(x) при $0 \geq x \leq 8$, а вторая при $x \geq 5$ вещественную (dre) и мнимую (dim) части интеграла

$$\int_x^{\infty} \frac{e^{-it}}{\sqrt{t}} dt \quad ,$$

зная которые можно вычислить для указанной полуоси косинус- $(C(x))$ и $(S(x))$ синус-интегралы Френеля.

6.2.20 Вариант 20. Тема: $C(x)\sqrt{\frac{\pi}{2x}} - C(\frac{x}{2})\sqrt{\frac{\pi}{x}}\ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \left(\mathbf{C(x)}\sqrt{\frac{\pi}{\mathbf{2x}}} - \mathbf{C}(\frac{\mathbf{x}}{\mathbf{2}})\sqrt{\frac{\pi}{\mathbf{x}}} + \frac{\mathbf{x^2}}{\mathbf{8}}\left(\frac{\mathbf{3}}{\mathbf{5}} - \frac{\mathbf{5x^2}}{\mathbf{144}}\right)\right)\frac{\mathbf{66560}}{\mathbf{7x^6}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	6.0	10
2)	0.010	0.015	10
3)	55	65	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.01$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
8. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
9. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

Примечание: $C(x) = \dfrac{1}{\sqrt{2\pi}} \cdot \int_0^x \cos t \dfrac{dt}{\sqrt{t}}$ представим в виде

$$C(x) = \sqrt{\frac{2x}{\pi}} \cdot \sum_{k=0}^{\infty} (-1)^k \frac{x^2 k}{(2k)! \cdot (4k+1)}$$

$C(x)$ — косинус-интеграл Френеля.

Для высокоточного расчёта $C(x)$ можно использовать либо имеющиеся в наличии в соответствующих библиотеках функции, либо при отсутствии последних приводимую в разделе 5 функцию **cfren1(x)**, а также подпрограмму csfren6(x,dre,dim,nout). Первая находит 2C(x) при $0 \geq x \leq 8$, а вторая при $x \geq 5$ вещественную (dre) и мнимую (dim) части интеграла

$$\int_x^\infty \frac{e^{-it}}{\sqrt{t}} dt \quad ,$$

зная которые можно вычислить для указанной полуоси косинус- $(C(x))$ и $(S(x))$ синус-интегралы Френеля.

6.2.21 Вариант 21. Тема: $S(x)\sqrt{\frac{\pi}{2x}} - S(\frac{x}{2})\sqrt{\frac{\pi}{x}}\ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \left(\mathbf{S(x)}\sqrt{\frac{\pi}{\mathbf{2x}}} - \mathbf{S}(\frac{\mathbf{x}}{\mathbf{2}})\sqrt{\frac{\pi}{\mathbf{x}}} - \frac{\mathbf{x}}{\mathbf{6}}\left(\mathbf{1} - \frac{\mathbf{x^2}}{\mathbf{8}}\right)\right)\frac{\mathbf{42240}}{\mathbf{31x^5}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	6.0	10
2)	0.0005	0.00075	10
3)	50	60	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.01$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
8. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
9. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

Примечание: $S(x) = \dfrac{1}{\sqrt{2\pi}} \cdot \int_0^x \sin t \dfrac{dt}{\sqrt{t}}$ представим в виде

$$S(x) = \sqrt{\frac{2x}{\pi}} \cdot \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)! \cdot (4k+3)}$$

$C(x)$ — косинус-интеграл Френеля.

Для высокоточного расчёта $S(x)$ можно использовать либо имеющиеся в наличии в соответствующих библиотеках функции, либо при отсутствии последних приводимую в разделе 5 функцию **sfren1(x)**, а также подпрограмму csfren6(x,dre,dim,nout). Первая находит 2S(x) при $0 \geq x \leq 8$, а вторая при $x \geq 5$ вещественную (dre) и мнимую (dim) части интеграла

$$\int_x^\infty \frac{e^{-it}}{\sqrt{t}} dt \quad ,$$

зная которые можно вычислить для указанной полуоси косинус- $(C(x))$ и $(S(x))$ синус-интегралы Френеля.

6.2.22 Вариант 22. Тема: $ci(x) - \gamma - ln(x)\ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \left(\mathbf{ci(x)} - \gamma - \mathbf{ln(x)} + \frac{\mathbf{x^2}}{\mathbf{4}}\left(\mathbf{1} - \frac{\mathbf{x^2}}{\mathbf{24}}\right)\right)\frac{\mathbf{4320}}{\mathbf{x^6}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	5.0	10.0	10
2)	0.01	0.02	10
3)	55	65	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.01$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
8. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
9. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

Примечание: $ci(x) = \gamma + ln(x) - \int_0^x \frac{1-\cos t}{t}dt$ представима в виде

$$ci(x) = \gamma + ln(x) + \sum_{k=1}^{\infty}(-1)^k \frac{x^{2k}}{(2k)!2k}\ ,$$

где $\gamma = 0.57721566490153286061$ — постоянная Эйлера (EulerGamma).

$ci(x)$ — интегральный косинус.

Высокоточный расчёт $ci(x)$ можно выполнить (если нужно), например, функцией **ci(x)** (см. раздел 5).

6.2.23 Вариант 23. Тема: $Si(x) - x + \frac{x^3}{18} \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x) = \left(Si(x) - x + \frac{x^3}{18}\left(1 - \frac{3x^2}{100}\right)\right) \frac{35280}{x^7}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	6.0	11	10
2)	0.01	0.02	10
3)	65	70	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.01$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
8. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
9. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

Примечание: $Si(x) = \int_0^x \frac{\sin t}{t} dt$ представима в виде

$$Si(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!(2k+1)}$$

$Si(x)$ — интегральный косинус.

Высокоточный расчёт $Si(x)$ можно выполнить (если нужно), например, функцией **si(x)** (см. раздел 5).

6.2.24 Вариант 24. Тема: $Si(x) - \sin x \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \left(\mathbf{Si(x)} - \sin \mathbf{x} - \frac{\mathbf{x^3}}{\mathbf{3}}\left(\frac{\mathbf{1}}{\mathbf{3}} - \frac{\mathbf{x^2}}{\mathbf{50}}\right)\right) \frac{\mathbf{5880}}{\mathbf{x^7}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	5.0	6.0	10
2)	0.01	0.02	10
3)	57	67	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x \to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.01$ относительно найденного $\lim\limits_{x \to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
8. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
9. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

Примечание: $Si(x) = \int_0^x \frac{\sin t}{t} dt$ представима в виде

$$Si(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!(2k+1)}$$

$$\sin x = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

$Si(x)$ — интегральный косинус.

Высокоточный расчёт $Si(x)$ можно выполнить (если нужно), например, функцией **si(x)** (см. раздел 5).

6.2.25 Вариант 25. Тема: $ci(x) - \cos x + 1 \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \left(\mathbf{ci(x)} - \cos\mathbf{x} + \mathbf{1} - \gamma - \mathbf{ln(x)} - \frac{\mathbf{x^2}}{\mathbf{4}}\left(\mathbf{1} - \frac{\mathbf{x^2}}{\mathbf{8}}\left(\mathbf{1} - \frac{\mathbf{x^2}}{\mathbf{27}}\right)\right)\right)\frac{\mathbf{46080}}{\mathbf{x^8}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	5.0	6.0	10
2)	0.05	0.06	10
3)	55	65	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.01$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
8. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
9. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

Примечание: $ci(x) = \gamma + ln(x) - \int_0^x \frac{1-\cos t}{t}dt$ представима в виде

$$ci(x) = \gamma + ln(x) + \sum_{k=1}^{\infty}(-1)^k \frac{x^{2k}}{(2k)!2k} \quad ,$$

где $\gamma = 0.57721566490153286061$ — постоянная Эйлера (EulerGamma).

$ci(x)$ — интегральный косинус.

Высокоточный расчёт $ci(x)$ можно выполнить (если нужно), например, функцией **ci(x)** (см. раздел 5).

6.2.26 Вариант 26. Тема: $ci(x) - \gamma - ln(x) + 0.25xSi(x)\ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \left(\mathbf{ci(x)} - \gamma - \mathbf{ln(x)} - \mathbf{0.25x}\left(\mathbf{Si(x)} + \frac{\mathbf{x^3}}{\mathbf{72}}\right)\right)\frac{\mathbf{5400}}{\mathbf{x^6}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	5.0	6.0	10
2)	0.01	0.02	10
3)	50	60	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.01$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
8. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
9. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

Примечание: $ci(x) = \gamma + ln(x) - \int_0^x \frac{1-\cos t}{t}dt$ представима в виде

$$ci(x) = \gamma + ln(x) + \sum_{k=1}^{\infty}(-1)^k \frac{x^{2k}}{(2k)!2k}\ ,$$

где $\gamma = 0.57721566490153286061$ — постоянная Эйлера (EulerGamma).

$Si(x) = \int_0^x \frac{\sin t}{t} dt$ представима в виде

$$Si(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!(2k+1)}$$

$ci(x)$ — интегральный косинус.

Высокоточный расчёт $ci(x)$ и $si(x)$ можно выполнить (если нужно), например, функциями **ci(x)** и b **si(x)** соответственно (см. раздел 5).

6.2.27 Вариант 27. Тема: $J_0(x) - \cos(x/\sqrt{2}) + \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \left(\mathbf{J_0(x)} - \cos\frac{\mathbf{x}}{\sqrt{\mathbf{2}}} - \frac{\mathbf{x^4}}{\mathbf{192}}\Big(\mathbf{1} - \frac{\mathbf{x^2}}{\mathbf{20}}\Big)\right)\frac{\mathbf{573440}}{\mathbf{3x^8}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	5.0	6.0	10
2)	0.05	0.06	10
3)	55	65	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.05$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос:

   “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
8. Ответ на вопрос:

   “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
9. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

Примечание: $J_0(x)$ — функция Бесселя, для которой справедливо разложение

$$J_0(x) = \sum_{k=0}^{\infty}(-1)^k\frac{(0.5x)^{2k}}{(k!)^2} \quad ,$$

$$\cos x = \sum_{k=0}^{\infty}(-1)^k\frac{x^{2k}}{(2k)!}$$

Высокоточный расчёт $J_0(x)$ можно выполнить (если нужно), например, функциями **bessj05(x)** и **bessj08(x)** (см. раздел 5).

6.2.28 Вариант 28. Тема: $J_1(x) - \sin(x/2) + \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x) = \left(J_1(x) - \sin\frac{x}{2} + \frac{x^3}{24}\left(1 - \frac{9x^2}{160}\right)\right) \frac{322560}{17x^7}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	1.0	11.0	10
2)	0.01	0.02	10
3)	55	65	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.01$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос: “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
8. Ответ на вопрос: “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
9. Верные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

В исходной формуле $\mathbf{J_1(x)}$ — функция Бесселя. Её расчёт возможен посредством вызова встроенной ФОРТРАН-функций **besj1**. Верны разложения:

$$\mathbf{J_1(x)} = \sum_{\mathbf{k=0}}^{\infty} (-\mathbf{1})^{\mathbf{k}} \frac{(\mathbf{0.5x})^{\mathbf{2k+1}}}{\mathbf{k!(k+1)!}} \quad ,$$

$$\mathbf{\sin(x)} = \sum_{\mathbf{k=0}}^{\infty} (-\mathbf{1})^{\mathbf{k}} \frac{\mathbf{x}^{\mathbf{2k+1}}}{\mathbf{(2k+1)!}}$$

6.2.29 Вариант 29. Тема: $J_0(x) - J_0(x/2) + \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x) = \left(J_0(x) - J_0(x/2) + \frac{3x^2}{16}\left(1 - \frac{5x^2}{64}\right)\right) \frac{16384}{7x^6}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	4.0	16.0	12
2)	0.01	0.02	10
3)	52	62	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.01$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос:

   "Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?"
8. Ответ на вопрос:

   "Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?"
9. Правильные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

В исходной формуле $\mathbf{J_0(x)}$ — функция Бесселя. Её расчёт возможен посредством вызова встроенной ФОРТРАН-функций **besj0**. Верно разложение:

$$\mathbf{J_0(x) = \sum_{k=0}^{\infty} (-1)^k \frac{(0.5x)^{2k}}{(k!)^2}} \quad ,$$

6.2.30 Вариант 30. Тема: $J_1(x) - 0.5xJ_0(x/2) + \ \cdots$

Разработать и оформить функциями алгоритмы расчёта с удвоенной точностью выражения:

$$\mathbf{f(x)} = \left(\mathbf{J_1(x) - 0.5x \cdot J_0(x/2) - \frac{x^3}{16}\left(1 - \frac{x^2}{12}\right)}\right)\frac{\mathbf{6144}}{\mathbf{x^7}}$$

Тестирующая программа должна табулировать **f(x)** для **n** значений аргумента, равномерно распределенных по промежутку **[a,b]** (**a**, **b** и **n** вводятся из файла).

Продемонстрировать работу программы для трёх наборов вводимых данных

N	a	b	n
1)	6.0	7.0	12
2)	0.010	0.012	10
3)	54	64	10

План письменного отчёта:

1. В чём достоинство и недостаток исходной формулы **f(x)** для расчёта на ЭВМ?
2. Аналитическое нахождение $\lim\limits_{x\to 0} f(x)$.
3. Оценивание погрешности расчёта по исходной формуле при $x = 0.01$ относительно найденного $\lim\limits_{x\to 0} f(x)$.
4. Получение разложения $f(x)$ в ряд Маклорена.
5. Вывод рекуррентной формулы расчёта его очередного слагаемого.
6. Сравнение результатов расчёта по исходной формуле $f(x)$ и разложению.
7. Ответ на вопрос: “Почему при $x \gg 1$ и достижении требуемой погрешности метода результат, полученный по разложению, *сомнителен*?”
8. Ответ на вопрос: “Почему при $x \ll 1$ результат, полученный исходной формулой, *сомнителен*?”
9. Верные значения $f(x)$ в конечных точках каждого из требуемых диапазонов.
10. Заключение (о чём узнали, решая данную задачу?)

В исходной формуле $\mathbf{J_0(x)}$ и $\mathbf{J_1(x)}$ — функции Бесселя. Их расчёт возможен посредством вызова встроенных ФОРТРАН-функций **besj0** и **besj1**. Верны разложения:

$$\mathbf{J_0(x)} = \sum_{\mathbf{k=0}}^{\infty} (\mathbf{-1})^{\mathbf{k}} \frac{\mathbf{(0.5x)^{2k}}}{\mathbf{(k!)^2}} \ ,$$

$$\mathbf{J_1(x)} = \sum_{\mathbf{k=0}}^{\infty} (\mathbf{-1})^{\mathbf{k}} \frac{\mathbf{(0.5x)^{2k+1}}}{\mathbf{k!(k+1)!}} \ ,$$

6.3 Контрольные результаты

6.3.1 Числа

Вариант 1.	
x	f(x)
1.0	-0.98607 30906 08608 73766
2.0	-0.93084 73418 88938 33598
0.0010	-0.99999 99859 45767 32391
0.0020	-0.99999 99437 83070 83071
50	-0.01589 75999 84738 98235
150	-0.00177 65135 80065 05839

Вариант 2.	
x	f(x)
1.0	-0.98621 97800 42801 47451
2.0	-0.94634 86563 60191 82721
0.0100	-0.99999 86094 16540 42036
0.0110	-0.99999 83173 94335 17678
50	-0.01649 57396 79452 68307
150	-0.00184 46769 22863 51321

Вариант 3.	
x	f(x)
1.0	-0.98213 01191 18993 34608
2.0	-0.93084 73418 88938 33598
0.0010	-0.99999 99819 30272 31021
0.0020	-0.99999 99277 21091 65722
50	-0.01138 46869 42467 02502
150	-0.00126 92994 85186 93941

Вариант 4.	
x	f(x)
1.0	-0.82386 21386 78841 12192
2.0	-0.52163 13688 98545 48217
0.0010	-0.99999 97933 33336 83333
0.0020	-0.99999 91733 33893 33330
10	-0.03638 61333 33333 33333
20	-0.00927 35333 33333 33333

Вариант 5.	
x	f(x)
1.0	-0.79037 77030 09295 52243
2.0	-0.46186 32192 65019 82298
0.0005	-0.99999 99367 55955 55245
0.0015	-0.99999 94308 03828 32022
7	-0.05640 50695 64266 84305
7.5	-0.04934 88536 42272 32451

Вариант 6.	
x	f(x)
1.0	-0.79112 99376 60690 12588
2.0	-0.46312 85474 90305 78671
0.0005	-0.99999 99370 12308 07312
0.0007	-0.99999 98765 44129 75254
7	-0.05537 28584 63773 00341
9	-0.03376 44536 46181 21964

Вариант 7.	
x	f(x)
1.0	-0.79247 59561 29347 08703
2.0	-0.16482 55279 03582 32060
0.0250	-0.99999 99022 24519 23926
0.0350	-0.99999 96243 85796 50718
3	-0.03581 60372 11253 60746
4	-0.01150 84805 33138 73642

Вариант 8.	
x	f(x)
1.0	-0.98169 03093 95702 97135
2.0	-0.92919 15932 24779 95538
0.0010	-0.99999 99814 81481 69192
0.0020	-0.99999 99259 25929 29293
50	-0.01115 51478 41320 86556
60	-0.00775 61898 95395 12288

Вариант 9.	
x	f(x)
1.0	0.93165 19111 55351 85372
2.0	0.75942 95427 29931 78187
0.0001	0.99999 99992 85714 28603
0.0002	0.99999 99971 42857 14793
25	0.01194 24016 14365 40708
30	0.00830 55856 85385 50023

Вариант 10.	
x	f(x)
1.0	0.88452 21639 16512 9951?
2.0	0.63114 13662 02090 8226?
0.0001	0.99999 99987 48473 7497?
0.0002	0.99999 99949 93895 01028
15	0.02033 04991 86966 28468
20	0.01148 07747 18112 45184

Вариант 11.	
x	f(x)
1.0	-0.79249 27055 15767 69092
2.0	-0.46646 15370 90313 94524
0.0001	-0.99999 99974 97185 80821
0.0011	-0.99999 96971 59555 47980
7.0	-0.05819 56831 06132 39480
7.5	-0.00126 92994 85186 93941

Вариант 12.	
x	f(x)
1.0	-0.83210 19668 66942 73542
2.0	-0.53841 70707 56225 74039
0.0020	-0.99999 92161 53641 26955
0.0030	-0.99999 82363 47136 65365
7.0	-0.07627 99518 78276 26147
7.5	-0.06679 03137 24856 66296

Вариант 13.	
x	f(x)
1.0	-0.83148 28805 61402 54344
2.0	-0.53718 89582 87477 45694
0.0025	-0.99999 87702 55891 02174
0.0035	-0.99999 75897 03919 83392
7.0	-0.07723 59738 66520 70539
7.5	-0.06776 82449 64889 97001

Вариант 14.	
x	f(x)
1.0	-0.82415 92230 15858 12113
2.0	-0.51766 79810 67629 94137
0.0010	-0.99999 97946 16552 95097
0.0050	-0.99999 48654 34089 85693
7.0	-0.05643 12145 82236 35143
7.5	-0.04866 03364 26341 13402

Вариант 15.	
x	f(x)
1.0	0.82815 47008 31412 43915
2.0	0.70113 53748 14964 39936
0.00010	0.99997 99110 46423 88398
0.00012	-0.99997 58933 35420 4715?
50	0.07314 26436 50286 3486?
60	0.06149 45854 45767 61355

Вариант 16.	
x	f(x)
1.0	-0.81888 48505 47410
2.0	-0.68697 50565 90081
0.0010	-0.99978 67036 98262
0.0020	-0.99957 34814 37955
49	-0.06924 06696 25434
50	-0.06792 89760 13103

Вариант 17.	
x	f(x)
1.0	0.98199 58739 52132 86649
2.0	0.93034 83794 34133 85531
0.0050	0.99999 95448 18055 11554
0.0060	0.99999 93445 38080 43210
50	0.01150 55916 72747 92555
60	0.00800 06345 17653 5738?

Вариант 18.	
x	f(x)
1.0	-0.98551 64639 73014 08149
2.0	-0.94365 98815 75143 77478
0.0010	-0.99999 99853 80117 09631
0.0020	-0.99999 99415 20470 03213
50	-0.01517 67312 99737 2711?
60	-0.01055 97655 95669 2440?

Вариант 19.	
x	f(x)
1.0	0.97616 04113 98303 9412?
2.0	0.90853 20134 82726 7015?
0.0005	0.99999 99939 56043 9772?
0.0010	0.99999 99758 24176 1635?
50	0.00784 67440 14111 9748?
60	0.00545 40868 42007 9092?

Вариант 20.	
x	f(x)
6.0	-0.63263 05990 30309 079??
11.0	-0.????? ????? ????? ?????
0.0100	-0.99999 86181 98525 457??
0.0150	-0.99999 68909 50188 185??
55	-0.01356 50085 35692 75???
65	-0.00972 80641 38525 541??

Вариант 21.	
x	f(x)
4.0	0.75879 11627 94456 62916
6.0	0.55442 98695 63305 13544
0.00050	0.99999 99955 29313 88842
0.00075	0.99999 99899 40956 28361
50	0.01131 83873 24301 14736
60	0.00786 77410 04361 78069

Вариант 22.	
x	f(x)
5	-0.72910 54062 46824 77643
10	-0.35463 71110 64688 14652
0.0100	-0.99999 86607 15476 18972
0.0200	-0.99999 46428 76190 42809
55	-0.01475 87266 59964 35280
65	-0.01059 06569 26996 18573

Вариант 23.	
x	f(x)
6	-0.69760 88219 83681 01787
11	-0.36913 70151 08604 25748
0.0100	-0.99999 89197 53889 90876
0.0200	-0.99999 56790 25201 48216
65	-0.01380 78161 08163 76623
70	-0.01191 86605 32228 88434

Вариант 24.	
x	f(x)
5	0.71149 31684 94429 07466
6	0.62056 80081 55217 84466
0.0100	0.99999 85596 72121 04119
0.0200	0.99999 42387 04553 90129
57	0.01200 33620 86340 16663
67	0.00870 00377 92267 70590

Вариант 25.	
x	f(x)
5.0	-0.76247 56596 41044 29825
6.0	-0.68305 08275 63050 78212
0.05	-0.99997 14291 22567 28698
0.06	-0.99995 88582 85691 38651
55	-0.01747 39058 64519 38651
65	-0.01254 27579 23598 08559

Вариант 26.	
x	f(x)
5.0	0.59818 85400 07568 25676
6.0	0.48611 09894 55778 33519
0.01	0.99999 78475 79176 11276
0.02	0.99999 13903 48450 35466
50	0.00750 51495 59698 35969
60	0.00521 05468 53098 77293

Вариант 27.	
x	f(x)
5.0	0.76317 18871 24999 01755
6.0	0.68319 91026 05146 70855
0.05	0.99997 17083 42581 78693
0.06	0.99995 92603 45097 21604
55	0.01634 66662 58925 53710
65	0.01172 59513 32947 50468

Вариант 28.	
x	f(x)
1	-0.98734 08403 50874 12921
11	-0.31301 24482 72991 54153
0.01	-0.99999 87234 48782 40058
0.02	-0.99999 48938 07969 35905
55	-0.01461 46334 09714 46769
65	-0.01048 12969 51881 57886

Вариант 29.	
x	f(x)
4	-0.78345 17651 45761 73734
16	-0.12728 04896 59754 62933
0.01	-0.99999 84189 00395 27419
0.02	-0.99999 36756 20610 04852
52	-0.01261 96169 70168 97235
62	-0.00888 95828 26280 01088

Вариант 30	
x	f(x)
6.0	0.57660 09409 07971 32723
7.0	0.48525 73868 02233 69472
0.010	0.99999 83333 35069 44320
0.012	0.99999 76000 03599 99630
54	0.01092 87899 55400 56859
64	0.00778 96077 30104 34365

6.3.2 Разложения

6.3.3 Рекуррентные формулы

6.4 Расчёт некоторых специальных функций

В приводимых далее результатах тестирующих ФОРТРАН-программ могут встречаться и результаты расчёта требуемых функций на основе разложения последних в ряд Маклорена при отсутствии соответствующих исходных ФОРТРАН-текстов. Предполагается, что человек, решающий задачу, при необходимости может разработать и написать их самостоятельно.

При выполнении лабораторной работы на языках C и C++ можно использовать функции, доступные через подключение **#include <math.h>** или, если хотим, **#include <cmath>** в случае C++. В частности, к таким функциям относятся интеграл ошибок **erf(x)**, его дополнение **erfc(x)**, функции Бесселя **j0** и **j1**.

На ФОРТРАНе упомянутые функции относятся к встроенным функциям языка. Их имена соответственно **erf(x)**, **erfc(x)**, **besj0(x)** и **besj1(x)**. При этом помним, что на ФОРТРАНе-77 не реализован механизм перегрузки функций. Поэтому указанные ФОРТРАН-функции выдают результат с одинарной точностью. Для получения их значений с удвоенной точностью следует пользоваться функциями **derf(x)**, **derfc(x)**, **dbesj0(x)** и **dbesj1(x)**.

Работающим на СИ полезно знать, что имеется обширная библиотека, называемая **GSL** (GNU Scientific Library). Здесь дан пример её использования для расчёта функций Бесселя **J0(x)**, **J1(x)**, интеграла ошибок **erf(x)** и его дополнения **erfc(x)**, а также интегральных экспоненты **E1(x)**, синуса **Si(x)** и косинуса **Ci(x)**. Последних трёх нет в **math.h**.

```
#include <stdio.h>
#include <gsl/gsl_sf.h> // Из раздела sf доступна любая специальная функция.
#include <math.h>       // <gsl/gsl_sf_J0.h> -"- лишь gsl_sf_bessel_J0(x).
int main()
{ double x, j0g, j0m, j1g, j1m, erfg, erfm, erfcg, erfcm;
  x=5.0;
  j0g=gsl_sf_bessel_J0(x); j0m=j0 (x);
  j1g=gsl_sf_bessel_J1(x); j1m=j1 (x);
  erfg = gsl_sf_erf(x);    erfm= erf(x);
  erfcg=gsl_sf_erfc(x);    erfcm=erfc(x);
  printf("GSL:   J0(%g)=%+.16e  math.h: =%+.16e\n", x,j0g,j0m);
  printf("GSL:   J1(%g)=%+.16e  math.h: =%+.16e\n", x,j1g,j1m);
  printf("GSL:  erf(%g)=%+.16e  math.h: =%+.16e\n", x, erfg, erfm);
  printf("GSL: erfc(%g)=%+.16e  math.h: =%+.16e\n", x,erfcg,erfcm);
  printf("GSL:   E1(%g)=%+.16e\n",x,gsl_sf_expint_E1(x));
  printf("GSL:   Si(%g)=%+.16e\n",x,gsl_sf_Si(x));
  printf("GSL:   Ci(%g)=%+.16e\n",x,gsl_sf_Ci(x));
  return 0;
}
```

```
$ g++ -lgsl -lgslcblas -lm tgslj0.c
$ ./a.out > result
```

```
GSL:   J0(5)=-1.7759677131433829e-01  math.h: =-1.7759677131433829e-01
GSL:   J1(5)=-3.2757913759146523e-01  math.h: =-3.2757913759146517e-01
GSL:  erf(5)=+9.9999999999846256e-01  math.h: =+9.9999999999846256e-01
GSL: erfc(5)=+1.5374597944280355e-12  math.h: =+1.5374597944280349e-12
GSL:   E1(5)=+1.1482955912753257e-03
```

6.4.1 Расчёт erf(x) (варианты 12, 13, 14).

Расчёт интеграла вероятностей

$$\mathbf{erf(x)} = \frac{\mathbf{2}}{\sqrt{\pi}} \int_{\mathbf{0}}^{\mathbf{x}} \mathbf{e^{-t^2} dt}$$

и его дополнения

$$\mathbf{erfc(x)} = \frac{\mathbf{2}}{\sqrt{\pi}} \int_{\mathbf{x}}^{\infty} \mathbf{e^{-t^2} dt} = \mathbf{1 - erf(x)}$$

можно осуществить, используя встроенные функции **erf(x)** и **erfc(x)** современного ФОРТРАНа (некоторые версии старых ФОРТРАН-компиляторов не имели **erf(x)** и **erfc(x)** среди встроенных). Кроме того, на практике на самом деле часто требуется не сама **erfc(x)**, значения которой при $\mathbf{x} \gg \mathbf{1}$ могут оказаться даже меньше машинного нуля, а произведение $\mathbf{erfc(x) * x * e^{x^2}}$, представляющее собой медленно меняющуюся конечную функцию. Ясно, что вычислять такое произведение на ЭВМ, используя встроенную **erfc(x)** — бессмысленно так, как её значение будет нулевым и искомое произведение окажется равным нулю, а не конечным. Поэтому полезно иметь алгоритм расчёта указанного произведения, который не требует прямого вычисления **erfc(x)**.

Ниже приводятся ФОРТРАН-функции, вычисляющие разными способами **erf(x)**, **erfc(x)**, упомянутое произведение $\mathbf{erfc(x) * x * e^{x^2}}$ и программы, тестирующие их расчёт.

1. **ddrfc(n, x, eps)** использует для расчёта при $\mathbf{x} \leq \mathbf{1}$ и $\mathbf{x} \geq \mathbf{1}$ соответствующее представление искомого произведения в виде цепной дроби.

   eps – требуемая относительная погрешность двух последовательных приближений (относительно последнего приближения).

   n – целочисленный ключ из диапазона $[\mathbf{1}, \mathbf{5}]$, задающий функцию, значение которой нужно получить, именно

n	Смысловая нагрузка ключа: расчёт
1	$\mathbf{erf(x)}$
2	$\mathbf{erfc(x)}$
3	$\mathbf{erf(x) * exp(x^2)}$
4	$\mathbf{erfc(x) * exp(x^2)}$
5	$\mathbf{erf(x) * exp(x^2)/x}$ при $\mathbf{\lvert x \rvert < 1}$
	$\mathbf{erfc(x) * exp(x^2) * x}$ при $\mathbf{\lvert x \rvert \geq 1}$

```
c============================================================
c  ddrfc (n,x,eps) вычисляет медленно меняющуюся компоненту
c функции erf(x) или erfc(x) через её разложения в цепную дробь.
c eps - требуемая относительная погрешность очередного приближения.
c Конкретный вид функции задается ее номером n согласно таблице:
c  n    Функция   :  n         Функция
c  1    erf ( x ) :  3    erf ( x ) * exp( x**2 )
c  2    erfc( x ) :  4    erfc( x ) * exp( x**2 )
c                     5    erf(x)*exp(x^2)/x,        при |x| <  1.
c                     5    erfc(x)*exp(x^2)*x        при |x| >= 1.
c ............................................................
      function ddrfc(n,x,eps)
      implicit none
      real*8 x, eps, ddrfc, c0, c05, c1, c2, c23, spi2, spi
      parameter (c0=0d0, c05=0.5d0, c1=1d0, c2=2d0, c23=2d0/3d0,
     > spi2=1.128379167095512573896d0, spi=0.5641895835477562869d0)
      real*8 g, q, u, v, w, f, di, dj, dl, z, dg, dg1
      integer n, i, j, l, k
      g=x*x; q=dabs(x)
      if (q.le.c1) then; u = c1; v = c23 * g; w=v; f=-g; i=2; j=3
                   do; i=i+2; l=j; j=j+2; di=i; dl=l; dj=j; f=-f;
                     z=u*f*di/(dl*dj); u=c1/(c1+z); v=-v*u*z; w=w+v
                     if (.not.( dabs(v).gt.eps*w) ) exit
     enddo; w=spi2/(c1-w); k=1
                  else
                    f=c05/g; u=c1; v=f; w=f; i=1
                    do; i=i+1; di=i; z=di*f*u
                      u=c1/(c1+z); v=-v*(u*z); w=w+v
                      if ( .not.(dabs(v).ge.eps*w) ) exit
                    enddo; w=spi/(c1+w); k=2
      endif
      ddrfc=w
      select case (n)
      case(5);
      case(4); dg=dexp(g); if (k.eq.1) then; ddrfc=dg-w*x
                                       else; ddrfc=w/x; endif
      case(3); if (k.eq.1) then; ddrfc=w*x
                           else; dg=dexp(g); ddrfc=dg-w/x; endif
      case(2); dg1=dexp(-g)
               if (k.eq.1) then; w=w*dg1*q; if(x.lt.c0) then;ddrfc=c1+w
                                                        else;ddrfc=c1-w
                                            endif
                           else; w=w*dg1/q; if(x.lt.c0) then;ddrfc=c2-w
                                                        else; ddrfc=w
               endif;                       endif
      case(1); dg1=dexp(-g)
               if (k.eq.1) then; w=w*dg1*q; if (x.lt.c0) w=-w; ddrfc=w
                           else; w=w*dg1/q
                                 if (x.lt.c0) then; ddrfc=w-c1
                                              else; ddrfc=c1-W
                endif;           endif
      endselect
      end
```

2. **derfl3**(**x**) — при $\mathbf{x} \leq \mathbf{3}$ вычисляет **erf**(**x**) на основе разложения последней по полиномам Чебышева нечётного порядка.

```
 function derfl3(x)  ! Подпрограмма-функция derfl3(x) вычисляет для
 implicit none       ! x<=3 значение функции erf(x) по её разложению
 real*8 x, derfl3    ! в ряд по полиномам Чебышева нечётного порядка.
 real*8 a(0:25)      ! Коэффициенты разложения взяты из книги Ю.Люка
 real*8 w, z, spi2   ! Специальные математические функции и их
 real*8 b0, b1, b2   ! аппроксимации. Издательство "МИР". Москва,1980
 integer i           !                     стр. 131-132 (таблица 4.5:
 data spi2/0.11283 79167 09551 30D01/   !            коэффициенты c)
 data a   / 1.09547 12997 77623 19604d0,
1          -0.28917 54011 26989 01480d0,
2           0.11045 63986 33795 06164d0,
3          -0.04125 31882 27856 54783d0,
4           0.01408 28380 70651 63996d0,
5          -0.00432 92954 47431 43677d0,
6           0.00119 82719 01592 28759d0,
7          -0.00029 99729 62353 24930d0,
8           0.00006 83258 60378 87479d0,
9          -0.00001 42469 88454 86775d0,
A           0.00000 27354 08772 83989d0,
1          -0.00000 04861 91287 19754d0,
2           0.00000 00803 87276 21172d0,
3          -0.00000 00124 18418 31213d0,
4           0.00000 00017 99532 58879d0,
5          -0.00000 00002 45479 48775d0,
6           0.00000 00000 31625 08603d0,
7          -0.00000 00000 03859 02200d0,
8           0.00000 00000 00447 20291d0,
9          -0.00000 00000 00049 33613d0,
B           0.00000 00000 00005 19303d0,
1          -0.00000 00000 00000 52258d0,
2           0.00000 00000 00000 05037d0,
3          -0.00000 00000 00000 00466d0,
4           0.00000 00000 00000 00041d0,
5          -0.00000 00000 00000 00004d0 /
 w=x/3d0
 z=4d0*w*w-2d0
 b0=0d0
 b1=0d0
 do i=25,0,-1
   b2=b1
   b1=b0
   b0=z*b1-b2+a(i)
 enddo
 derfl3=spi2*w*(b0-b1)
 end
```

3. **derfcge**(**x**) — при $\mathbf{x} \geq \mathbf{3}$ вычисляет $\mathbf{erfc}(\mathbf{x}) * \mathbf{exp}(\mathbf{x^2}) * \mathbf{x}$ на основе разложения последней по полиномам Чебышева чётного порядка.

```
      function derfcge(x) ! Подпрограмма-функция derfcge(x) вычисляет для
      implicit none       ! x>=3 значение функции:  2x*erfc(x)*exp(x**2)
      real*8 x, derfcge   ! по её разложению в ряд по полиномам Чебышева
      real*8 c(0:22)      ! чётного порядка. Коэффициенты разложения взяты
      real*8 w, z, spi2   ! из книги Ю.Люка  Специальные математические
      real*8 b0, b1, b2   ! функции и их аппроксимации. Издательство "МИР".
      integer i           ! Москва, 1980.   стр. 131-132 (таблица 4.5:
      data spi2/0.11283 79167 09551 30D01/   !            коэффициенты c)
      data c   / 0.97508 34237 08555 92854D0,
     1           -0.02404 93938 50414 60496D0,
     2            0.00082 04522 40880 43199D0,
     3           -0.00004 34293 08130 34276D0,
     4            0.00000 30184 47034 03493D0,
     5           -0.00000 02544 73319 25082D0,
     6            0.00000 00248 58353 02051D0,
     7           -0.00000 00027 31720 13238D0,
     8            0.00000 00003 30847 22797D0,
     9           -0.00000 00000 43505 49080D0,
     A            0.00000 00000 06141 21457D0,
     1           -0.00000 00000 00922 36928D0,
     2            0.00000 00000 00146 35665D0,
     3           -0.00000 00000 00024 39278D0,
     4            0.00000 00000 00004 24976D0,
     5           -0.00000 00000 00000 77084D0,
     6            0.14507D-15,-0.2824D-16,0.567D-17,
     9           -0.117D-17,0.25D-18,-0.5D-19,0.1D-19/
      w=3D0/X
      z=4d0*w*w-2d0
      b0=0d0
      b1=0d0
      do i=22,0,-1
        b2=b1
        b1=b0
        b0=z*b1-b2+c(i)
      enddo
      derfcge=spi2*(b0-b1*z/2d0)
      end
```

4. Программа, тестирующая расчёт **erf(x)** :

```
c Програма awtsterf тестирует функции расчёта erf(x) через:
c      1. derfl3(x) - разложение erf(x) по полиномам Чебышева
c                                  НЕЧЕТНОГО порядка при x<=3);
c      2. ddrfc(1,x,eps) - разложение erf(x) в цепную дробь;
c      3. derfmac(x,eps) - разложение erf(x) в ряд Маклорена
c Результат - таблица, каждая строка которой содержит:
c 1) аргумент  x;
c 2) erf(x), вычисленная встроенной функцией defr(x)
c 3) относительная ошибка derfl3(x);
c 4) относительная ошибка ddrfc(1,x,1d-17);
c 5) относительная ошибка derfmac(x,1d-17);
      program tsterf
      implicit none
      real*8  pi, spi2, x0, hx, xn, eps, x
      real*8 erf00, erf01, erf02, erf03
      real*8        rel01, rel02, rel03
      real*8 derfl3, ddrfc, derfmac, derfcge
      integer num, i, nx, nres / 6 /
      data pi /3.141592 653589 7932385d0/
      spi2=2d0/dsqrt(pi)
      x0 = 0d0; hx = 0.5d0; xn = 3.0d0
      eps=1d-17
      num=1
      nx=idint((xn-x0)/hx+1.5D0)
      write(nres,1000)
      do i=1,nx
        x=x0+(i-1)*hx
        erf00 = derf(x); erf01 = derfl3(x)
       erf02 = ddrfc(1,x,eps)
       erf03 = x*derfmac(x,eps)
        rel01 = 1000;        rel02 = 1000;        rel03 =1000
        if (erf00.ne.0d0) then
           rel01=(erf01-erf00)/erf00;
           rel02=(erf02-erf00)/erf00;
           rel03=(erf03-erf00)/erf00
        endif
        write(nres,1001) x,erf00,rel01,rel02,rel03
      enddo
      stop 0000
 1000 format(3x,'#  x ',14x,'derf(x)',7x,'  derf(Чеб)',
     >          '  derf(ddr)','  derf(mac)' )
 1001 format(1x,d10.3,d25.16,d11.3,d11.3,d11.3)
      end
```

При $\mathbf{x} \geq \mathbf{6}$ в пределах удвоенной точности **erf(x)=1**. Для тестирования в диапазоне $\mathbf{x} \in [\mathbf{3}, \mathbf{6}]$ расчёт **erf(x)** выгодно вести по формуле **1-erfc(x)**, используя для расчёта **erfc(x)** выражение **derfcge(x)*exp(-x*x)/2/x**. Видно, что при $\mathbf{x} \gg \mathbf{1}$ саму **erfc(x)** вычислять нецелесообразно так, как результат из-за малости экспоненты оказывается равен нулю. Другими словами, если по задаче требуется вычислять выражения вида **x*erfc(x)*exp(x*x)**, то используем для этого НЕ встроенную функцию **erfc(x)**, а **derfcge(x)**.

5. Программа, тестирующая расчёт **erfc(x)** и $\mathbf{erfc(x) * exp(x^2) * x}$:

```
c Програма tsterfc тестирует функции расчёта erfc(x) через:
c      1. derfcge(x) - разложение функции 2x*exp(x**2)*erfc(x)
c                      по полиномам Чебышева ЧЕТНОГО порядка при x>=3);
c      2. ddrfc(2,x,eps) - разложение erfc(x) в цепную дробь;
c      3. ddrfc(5,x,eps) - разложение x*exp(x**2)*erfc(x) в цепную дробь;
c Результат - две таблицы.
c   i-я строка первой содержит:
c      1) аргумент  x=3+(i-1);
c      2) erfc(x), вычисленную встроенной функцией defrc(x);
c      3) ( derfge(x)*dexp(-x*x)/2/x- derfc(x)) / derfc(x);
c      4) ( ddrfc(2,x,1d-17) - derfc(x)       ) / derfc(x);
c      5) ( ddrfc(5,x,1d-17) - derfge(x)/2    ) / ddrfc(5,x,1d-17);
c   i-я строка второй для аргумента x=10d0**i содержит значение
c          x * exp(x**2) erfc(x), вычисленное ddrfc(5,x,1d-17), и
c    погрешность работы derfcge(x) относительно ddrfc(5,x,1d-17).
      program tsterfc
      implicit none
      real*8  pi, spi2, x0, hx, xn, eps, x, ddrfc, derfcge
      real*8 erfc00, erfc01, erfc02, erfc03
      real*8    rel01, rel02, rel03
      integer num, i, nx, nres / 6 /
      data pi /3.141592 653589 7932385d0/ ; spi2=2d0/dsqrt(pi)
      x0 = 3; hx = 1d0; xn = 10d0; eps=1d-17
      nx=idint((xn-x0)/hx+1.5d0) ; write(nres,1000)
      do i=1,nx
        x=x0+(i-1)*hx; erfc00 = derfc(x);       erfc01 = derfcge(x)/2
                      erfc02 = ddrfc(2,x,eps); erfc03 = ddrfc(5,x,eps)
        rel01=(erfc01*dexp(-x*x)/x-erfc00)/erfc00;
        rel02=(erfc02-erfc00)/erfc00;
        rel03=(erfc03-erfc01)/erfc03
        write(nres,1001) x,erfc00,rel01,rel02,rel03
      enddo
      x=1; write(*,1010)
      do i=1,10; x=x*10;
        erfc01 = derfcge(x)/2; erfc03 = ddrfc(5,x,eps)
        rel03=(erfc03-erfc01)/erfc03
        write(nres,'(d10.3,d25.16,d15.2)') x,erfc03, rel03
      enddo
 1000 format(3x,'#  x ',14x,'derfc(x)',7x,'  rel01(Чеб)',
     >          '  rel01(ddr)',3x,'erfc01-erfc03'/
     >       3x,'#',60x,'-------------'/3x,'#',64x,'erfc03')
 1001 format(1x,d10.3,d25.16,d12.2,d12.2,d15.2)
 1010 format(//3x,'#  x ',14x,'erfc03',7x,3x,'erfc01-erfc03'/
     >       3x,'#',34x,'-------------'/3x,'#',38x,'erfc03')
      end
```

6. Результаты работы **tsterf** :

```
      x              derf(x)          derf(Че6)  derf(ddr)  derf(mac)
  0.000D+00   0.0000000000000000D+00  0.100D+04  0.100D+04  0.100D+04
  0.500D+00   0.5204998778130465D+00  0.427D-15  0.000D+00  0.427D-15
  0.100D+01   0.8427007929497149D+00  0.395D-15  0.000D+00  0.263D-15
  0.150D+01   0.9661051464753108D+00  0.345D-15  0.000D+00  0.690D-15
  0.200D+01   0.9953222650189527D+00  0.446D-15  0.000D+00  0.335D-15
  0.250D+01   0.9995930479825550D+00  0.444D-15  0.000D+00  0.167D-14
  0.300D+01   0.9999779095030014D+00  0.444D-15  0.000D+00  0.655D-14
```

7. Результаты работы **tsterfc** :

```
   #  x              derfc(x)          rel01(Че6)  rel01(ddr)   erfc01-erfc03
   #                                                            -------------
   #                                                                erfc03
  0.300D+01   0.2209049699858544D-04    0.60D-15    0.15D-15      -0.41D-15
  0.400D+01   0.1541725790028002D-07    0.30D-15   -0.21D-15      -0.41D-15
  0.500D+01   0.1537459794428035D-11    0.33D-15   -0.13D-15      -0.40D-15
  0.600D+01   0.2151973671249892D-16    0.15D-15   -0.29D-15      -0.40D-15
  0.700D+01   0.4183825607779414D-22    0.47D-15    0.14D-15      -0.40D-15
  0.800D+01   0.1122429717298293D-28    0.45D-15    0.00D+00      -0.40D-15
  0.900D+01   0.4137031746513810D-36    0.54D-15    0.20D-15      -0.40D-15
  0.100D+02   0.2088487583762545D-44    0.31D-15   -0.15D-15      -0.40D-15
```

```
   #  x              erfc03           erfc01-erfc03
   #                                  -------------
   #                                      erfc03
 0.100D+02   0.5614099274382258D+00      -0.40D-15
 0.100D+03   0.5641613782989433D+00      -0.39D-15
 0.100D+04   0.5641893014533876D+00      -0.39D-15
 0.100D+05   0.5641895807268084D+00      -0.39D-15
 0.100D+06   0.5641895835195468D+00      -0.39D-15
 0.100D+07   0.5641895835474742D+00      -0.39D-15
 0.100D+08   0.5641895835477535D+00      -0.39D-15
 0.100D+09   0.5641895835477563D+00      -0.39D-15
 0.100D+10   0.5641895835477563D+00      -0.39D-15
 0.100D+11   0.5641895835477563D+00      -0.39D-15
```

6.4.2 Расчёт $E_1(x)$ (варианты 15, 16).

Расчёт первой интегрально-показательной функции

$$\mathbf{E_1(x) = \int_1^{\infty} e^{-xt} \frac{dt}{t}}$$

возможен, например, посредством подпрограмм-функций **e141a(x)** и **e141c(x)**.

1.
```
      function e141a(x)   ! Функция e141a(x) вычисляет для заданного аргумента
      implicit none       ! ( 0 <= x <= 8 ) значение E1(x)+gamma+ln(x) через
      real*8 x, e141a     ! разложение в ряд по смещенным полиномам Чебышева.
      real*8 a(0:24)      ! Коэффициенты разложения взяты из книги  Ю.Люка
      real*8 w,z          ! Специальные математические функции и их
      real*8 b0, b1, b2   ! аппроксимации. Издательство "МИР". Москва, 1980.
      integer  i          !                           стр. 112 (таблица 4.1).
      data a / 1.67391 43547 42720 57853d0, !
     1         1.22849 44785 47155 95018d0, ! Здесь
     2        -0.31786 98982 98373 61369d0, !
     3         0.09274 60391 97292 19112d0, !     E1(x) - первая интегрально-
     4        -0.02602 73631 69001 19930d0, !             показательная
     5         0.00674 81530 94642 28057d0, !             функция;
     6        -0.00159 89706 89342 25285d0, ! gamma=0.57721 56649 01532 86061
     7         0.00034 59445 38800 95403d0, !
     8        -0.00006 85365 26820 11094d0, !   (постоянная Эйлера)
     9         0.00001 24859 32391 73701d0,
     A        -0.00000 21013 44638 71704d0,
     1         0.00000 03281 84479 81081d0,
     2        -0.00000 00477 68786 45154d0,
     3         0.00000 00065 05816 88192d0,
     4        -0.00000 00008 32101 93692d0,
     5         0.00000 00001 00281 01125d0,
     6        -0.00000 00000 11422 29304d0,
     7         0.00000 00000 01233 07945d0,
     8        -0.00000 00000 00126 48523d0,
     9         0.00000 00000 00012 35706d0,
     B        -0.00000 00000 00001 15226d0,
     1         0.00000 00000 00000 10276d0,
     2        -0.00000 00000 00000 00878d0,
     3         0.00000 00000 00000 00072d0,
     4        -0.00000 00000 00000 00006d0 /
      w=0.125d0*x
      z=4d0*w-2d0
      b0=0d0
      b1=0d0
      do i=24,0,-1
        b2=b1
        b1=b0
        b0=z*b1-b2+a(i)
      enddo
      e141a=b0-b1*z/2d0
      end
```

2.
```
      function e141c(x)  ! Функция e141c(x) вычисляет для заданного аргумента
      implicit none      ! ( x >= 5 ) значение выражения E1(x)*exp(x)*x через
      real*8 x, e141c    ! его разложение в ряд по смещённым полиномам Чебышева.
      real*8 c(0:30)     ! Коэффициенты разложения взяты из книги
      real*8 w,z         ! Ю.Люка  Специальные математиче! ские функции и их
      real*8 b0,b1,b2    !  аппроксимации. Издательство "МИР". Москва, 1980.
      integer i          ! стр. 112-113 (таблица 4.1 (c)).
      data c /
     >  0.92078 51444 53893 91645d0,
     1 -0.07343 41178 31621 28775d0,
     2  0.00520 98119 67272 32977d0,
     3 -0.00050 21407 19895 99012d0,
     4  0.00005 92079 69379 26337d0,
     5 -0.00000 80856 45130 97880d0,
     6  0.00000 12372 03856 47926d0,
     7 -0.00000 02075 01768 60703d0,
     8  0.00000 00375 60054 74022d0,
     9 -0.00000 00072 54109 76004d0,
     A  0.00000 00014 81816 01182d0,
     1 -0.00000 00003 17956 82863d0,
     2  0.00000 00000 71269 35828d0,
     3 -0.00000 00000 16612 31319d0,
     4  0.00000 00000 04011 55069d0,
     5 -0.00000 00000 01000 38792d0,
     6  0.00000 00000 00256 93341d0,
     7 -0.00000 00000 00067 80374d0,
     8  0.00000 00000 00018 34785d0,
     9 -0.00000 00000 00005 08209d0,
     B  0.00000 00000 00001 43861d0,
     1 -0.00000 00000 00000 41560d0,
     2  0.00000 00000 00000 12238d0,
     3 -0.00000 00000 00000 03669d0,
     4  0.00000 00000 00000 01119d0,
     5 -0.00000 00000 00000 00347d0,
     6  0.00000 00000 00000 00109d0,
     7 -0.00000 00000 00000 00035d0,
     8  0.00000 00000 00000 00011d0,
     9 -0.00000 00000 00000 00004d0,
     C  0.00000 00000 00000 00001d0 /
      w=5d0/x
      z=4d0*w-2d0
      b0=0d0
      b1=0d0
      do i=30,0,-1
        b2=b1
        b1=b0
        b0=z*b1-b2+c(i)
      enddo
      e141c=b0-b1*z/2d0
      end
```

3.

```
      program teste1              ! Программа тестирует алгоритмы расчета
      implicit none               ! первой интегрально-показательной функции
      integer  nres               ! E1(x). Первый -- функция e1mac(x) -- ведет
      real*8 rel                  ! расчет через разложение в ряд Маклорена.
      real*8 x,r1,r2,a,b,h,gamma  ! выражения
      real*8 e1mac, e141a, e141c  !                E1(x)+gamma+ln(x)
      integer n, i                !
                                  ! Второй -- функция e141a(x) -- через
      gamma=.5772156649015328606d0!
      a=0.1d0; b=1.1d0; n=10      ! разложение по смещённым полиномам Чебышева
      h=(b-a)/n                   ! при рабочем диапазоне аргумента:
      write(*,1200)               !
      do i=0,n                    !            0 <= x <= 8.
        x=a+i*h; r1=e1mac(x)*x    !
                 r2=e141a(x)      ! Третья -- функция e141c(x) -- через
        rel=dabs(r1-r2)/r2        ! разложение по смещённым полиномам
        write(*,1201) x,r1,r2, rel ! Чебышева функции E1(x)*x*exp(x) при
      enddo                       ! рабочем диапазоне аргумента: x >= 5.
      a=1; b=9; n=8
      h=(b-a)/n; write(*,'(/)')
      write(*,1200)
      do i=0,n
        x=a+i*h; r1=e1mac(x)*x
                 r2=e141a(x)
        rel=dabs(r1-r2)/r2
        write(*,1201) x,r1, r2, rel
      enddo
      a=5d0; b=8d0; n=3           ! Сравнение работы e141a и e141c в
      h=(b-a)/n; write(*,1300)    ! области перекрытия рабочих диапазонов:
      do i=0,n                    !            5 <=  x <= 8.
        x=a+i*h; r1=e141a(x)
        r2=e141c(x)*dexp(-x)/x + gamma+dlog(x)
        rel=dabs(r1-r2)/r2
        write(*,1201) x, r1, r2, rel
      enddo
 1200 format(1x,' #  x',12x,'e1mac(x)*x     = ',4x,
     >                        'e141a=E1+g+ln(x)',6x,'relerr')
 1201 format(1x,2x,f5.2, d25.17, d25.17, d10.2)
 1300 format(//1x,' #  x',7x,'e141a=E1(x)+g+ln(x) = ',
     >                     'e141c(x)exp(-x)/x+g+ln(x)',2x,'reller')
      end
```

4. Результаты работы **testE1**:

```
#  x            e1mac(x)*x     =     e141a=E1+g+ln(x)      relerr
  0.10  0.97554530326877859D-01  0.9755453032687810 8D-01  0.26D-14
  0.20  0.19042829665132555D+00  0.1904282966513262 1D+00  0.35D-14
  0.30  0.27891951225144357D+00  0.2789195122514434 0D+00  0.60D-15
  0.40  0.36330505189304030D+00  0.3633050518930403 0D+00  0.00D+00
  0.50  0.44384207911774837D+00  0.4438420791177486 4D+00  0.63D-15
  0.60  0.52076954432494427D+00  0.5207695443249447 1D+00  0.85D-15
  0.70  0.59430956419630954D+00  0.5943095641963098 7D+00  0.56D-15
  0.80  0.66466869213286617D+00  0.6646686921328661 7D+00  0.00D+00
  0.90  0.73203908856970623D+00  0.7320390885697063 4D+00  0.15D-15
  1.00  0.79659959929705337D+00  0.7965995992970536 0D+00  0.28D-15
  1.10  0.85851674924189803D+00  0.8585167492418979 1D+00  0.13D-15


#  x            e1mac(x)*x     =     e141a=E1+g+ln(x)      relerr
  1.00  0.79659959929705337D+00  0.7965995992970536 0D+00  0.28D-15
  2.00  0.13192633561695390D+01  0.1319263356169539 3D+01  0.17D-15
  3.00  0.16888763346638391D+01  0.1688876334663839 6D+01  0.26D-15
  4.00  0.19672893784312719D+01  0.1967289378431272 4D+01  0.23D-15
  5.00  0.21878018729269084D+01  0.2187801872926908 9D+01  0.20D-15
  6.00  0.23693352165817498D+01  0.2369335216581750 3D+01  0.19D-15
  7.00  0.25232412956884582D+01  0.2523241295688456 5D+01  0.70D-15
  8.00  0.26566948722042070D+01  0.2656694872204212 7D+01  0.22D-14
  9.00  0.27744526895919290D+01  0.2774452689591843 8D+01  0.31D-13


#  x       e141a=E1(x)+g+ln(x) = e141c(x)exp(-x)/x+g+ln(x)  reller
  5.00  0.21878018729269089D+01  0.2187801872926908 4D+01  0.20D-15
  6.00  0.23693352165817503D+01  0.2369335216581750 3D+01  0.00D+00
  7.00  0.25232412956884565D+01  0.2523241295688456 5D+01  0.00D+00
  8.00  0.26566948722042127D+01  0.2656694872204212 7D+01  0.00D+00
```

5. Расчёт значений только **E1(x)** можно оформить функцией **e141(x)**:

```
function e141(x)                    ! Функция e141(x) вычисляет для
implicit none                       ! аргумента (x) значение первой
real*8 x,e141,e141a,e141c,w,g       ! интегрально-показательной
data g /.5772156649015328606 1d0 /  ! функции E1(x).
if (x.ge.5d0) then                  ! Расчёт ведется на основе известных
       w=e141c(x)/x*dexp(-x)        ! разложений E1(x) в ряд по полиномам
  else                              ! Чебышева (см. книгу Люка:
    if (x.gt.0) then                ! Специальные математические
      w=e141a(x)-g-dlog(x)          !  функции и их аппроксимации.
     else                           ! Издательство "МИР". Москва, 1980.
       w=1d303                      ! стр. 112-113 (таблица 4.1).
    endif
endif
e141=w
end
```

6.4.3 Расчёт Ci(x) и Si(x) (варианты 22, 23, 24, 25, 26).

```
      function ci(x)
      implicit none
      real*8 x, ci, w, cosint1, g
      parameter ( g=0.5772156649015328606 1d0 )
      real*8 dre, dim
      integer ier
      if (x.lt.5) then
                    w=-cosint1(x)+g+dlog(x)
        else
          call csintegr(x, dre, dim, ier)
          if (ier.eq.0) w=-(dcos(x) * dim - dsin(x) *dre) / x
      endif
      ci=w
      end

      function cosint1(x) ! Функция cosint1(x) вычисляет для аргумента
      implicit none       ! ( -8 <=x<= 8 ) интегральный косинус через его
      real*8 x, cosint1   ! разложение в ряд по полиномам Чебышева четного
      real*8 a(0:17)      ! порядка. Коэффициенты разложения взяты из книги
      real*8 w,z          ! Ю.Люка  Специальные математические функции и их
      real*8 b0, b1, b2   ! аппроксимации. Издательство "МИР". Москва, 1980.
      integer i                                  ! стр. 123-124 (таблица 4.4).
      data a / 1.94054 91464 83554 93374d0,
     1 0.94134 09132 86521 34390d0, -0.57984 50342 92992 76547d0,
     3 0.30915 72011 15927 13017d0, -0.09161 01792 20771 33969d0,
     5 0.01644 37407 51546 24963d0, -0.00197 13091 95216 41024d0,
     7 0.00016 92538 85083 49925d0, -0.00001 09393 29573 10627d0,
     9 0.00000 05522 38574 83778d0, -0.00000 00223 99493 31410d0,
     1 0.00000 00007 46533 25345d0, -0.00000 00000 20818 33157d0,
     3 0.00000 00000 00493 12353d0, -0.00000 00000 00010 04784d0,
     5 0.00000 00000 00000 17803d0, -0.00000 00000 00000 00277d0,
     7 0.00000 00000 00000 00004d0 /
       w=0.125d0*x
       z=4d0*w*w-2d0
       b0=0d0
       b1=0d0
       do i=17,0,-1
          b2=b1
          b1=b0
          b0=z*b1-b2+a(i)
       enddo
       cosint1=b0-b1*z/2d0
       end
```

```
 function si(x)
 implicit none
 real*8 x, si, w, sinint1, g, pi2
 parameter (  g=0.57721566490153286061d0,
>           pi2=1.57079632679489661923d0)
 real*8 dre, dim
 integer ier
 if (x.lt.5) then
                w=sinint1(x)
   else
     call csintegr(x, dre, dim, ier)
     if (ier.eq.0) w=pi2 -(dcos(x)*dre+dsin(x)*dim)/x
 endif
 si=w
 end

 function sinint1(x) ! Функция sinint1(x) вычисляет для аргумента
 implicit none       ! ( -8 <=x<= 8 ) интегральный синус через его
 real*8 x, sinint1   ! разложение в ряд по полиномам Чебышева нечетного
 real*8 b(0:16)      ! порядка. Коэффициенты разложения взяты из книги
 real*8 w,z          ! Ю.Люка  Специальные математические функции и их
 real*8 b0, b1, b2   ! аппроксимации. Издательство "МИР". Москва, 1980.
 integer i                                ! стр. 123-124 (таблица 4.4).
 data b / 1.95222 09759 53071 08224d0,
1 -0.68840 42321 25715 44408d0,  0.45518 55132 25584 84126d0,
3 -0.18045 71236 83877 85342d0,  0.04104 22133 75859 23964d0,
5 -0.00595 86169 55588 85229d0,  0.00060 01427 41414 43021d0,
7 -0.00004 44708 32910 74925d0,  0.00000 25300 78230 75133d0,
9 -0.00000 01141 30759 30294d0,  0.00000 00041 85783 94210d0,
1 -0.00000 00001 27347 05516d0,  0.00000 00000 03267 36126d0,
3 -0.00000 00000 00071 67679d0,  0.00000 00000 00001 36020d0,
5 -0.00000 00000 00000 02255d0,  0.00000 00000 00000 00033d0 /
 w=0.125d0*x
 z=4d0*w*w-2d0
 b0=0d0
 b1=0d0
 do i=16,0,-1
   b2=b1
  b1=b0
   b0=z*b1-b2+b(i)
 enddo
 sinint1=w*(b0-b1)
 end
```

```
 subroutine csintegr(x,dre,dim, ier)     ! Подпрограмма csintegr(x)
 implicit none                           ! вычисляет для аргумента
 real*8 x, dre, dim, w,dcheb0            ! (x>=5) значения dre и dim
 real*8 cr(0:36), ci(0:34)               ! таких, что     cos(t)
 integer  ier                            !   x * [x,...) ------ dt =
 data cr / 0.97615 52711 28712 28562d0,  !                  t
1         -0.03046 56658 03069 59120d0,  ! cos(x) * dim - sin(x) * dre  и
2         -0.00578 07368 31483 85631d0,  !
3          0.00083 86432 56650 89313d0,  !               sin(t)
4         -0.00002 15746 20728 12156d0,  !  x * [x,...) ------ dt =
5         -0.00001 56456 41351 02321d0,  !                 t
6          0.00000 40400 10138 43204d0,  !cos(x) * dre + sin(x) * dim
7         -0.00000 04349 85305 97434d0,  !
8         -0.00000 00534 30218 60611d0,  ! через  разложения в ряд
9          0.00000 00385 02885 51259d0,  ! по смещённым полиномам
A         -0.00000 00100 73535 82172d0,  ! Чебышева.
1          0.00000 00012 80496 19406d0,  ! Коэффициенты разложения
2          0.00000 00001 86917 28895d0,  ! взяты из книги Ю.Люка
3         -0.00000 00001 70673 48371d0,  ! Специальные математические
4          0.00000 00000 58800 44115d0,  ! функции и их аппроксимации.
5         -0.00000 00000 12157 23809d0,  ! Издательство "МИР". Москва,
6          0.00000 00000 00474 81418d0,  ! стр. 124-125 (табл. 4.4).
7  0.00000 00000 00905 90381d0, -0.00000 00000 00500 96832d0,
9  0.00000 00000 00166 16291d0, -0.00000 00000 00034 84536d0,
1  0.00000 00000 00000 57400d0,  0.00000 00000 00003 68837d0,
3 -0.00000 00000 00002 17822d0,  0.00000 00000 00000 81978d0,
5 -0.00000 00000 00000 21304d0,  0.00000 00000 00000 02270d0,
7  0.00000 00000 00000 01445d0, -0.00000 00000 00000 01220d0,
9  0.00000 00000 00000 00577d0, -0.00000 00000 00000 00199d0,
1  0.00000 00000 00000 00046d0, -0.00000 00000 00000 00001d0,
3 -0.00000 00000 00000 00006d0,  0.00000 00000 00000 00005d0,
5 -0.00000 00000 00000 00002d0,  0.00000 00000 00000 00001d0 /
 data ci / 0.08968 45854 91642 30208d0,
1  0.08508 92472 92294 52754d0, -0.00507 18267 77756 90802d0,
3 -0.00033 42234 15981 73821d0,  0.00012 85606 50086 06518d0,
5 -0.00001 52025 51359 72619d0, -0.00000 05958 96122 75216d0,
7  0.00000 07134 72533 53084d0, -0.00000 01760 03581 15661d0,
9  0.00000 00192 57654 44417d0,  0.00000 00033 63591 94377d0,
1 -0.00000 00024 25468 70827d0,  0.00000 00007 13431 29834d0,
3 -0.00000 00001 14604 07035d0, -0.00000 00000 06784 17843d0,
5  0.00000 00000 12656 12487d0, -0.00000 00000 05323 09477d0,
7  0.00000 00000 01400 46450d0, -0.00000 00000 00180 45804d0,
9 -0.00000 00000 00050 26164d0,  0.00000 00000 00046 00566d0,
1 -0.00000 00000 00019 53107d0,  0.00000 00000 00005 62862d0,
3 -0.00000 00000 00000 89561d0, -0.00000 00000 00000 16803d0,
5  0.00000 00000 00000 21351d0, -0.00000 00000 00000 10783d0,
7  0.00000 00000 00000 03813d0, -0.00000 00000 00000 00919d0,
9  0.00000 00000 00000 00052d0,  0.00000 00000 00000 00099d0,
1 -0.00000 00000 00000 00073d0,  0.00000 00000 00000 00034d0,
3 -0.00000 00000 00000 00012d0,  0.00000 00000 00000 00003d0 /
 w=5/x
 dre=dcheb0(cr,w,36,3,ier)
 if (ier.eq.0) dim=dcheb0(ci,w,34,3,ier)
 end
```

```
C Подпрограмма-функция dcheb0(a,x,n,key,ier) вычисляет,
c используя суммирующий алгоритм Кленшоу,
C для заданных в первых N+1 элементах вектора A
C коэффициентов разложения по полиномам Чебышева и
C аргумента X сумму: F(N,X)=[K=0,N] A(K) T(L(KEY),X),
c
C где       L       KEY
C         --------------
C           N        0    просто по полиномам Чебышева
C         2N+1       1    по полиномам Чебышева нечетного порядка
C          2N        2    по полиномам Чебышева четного порядка
C          N СМЕЩ    3,   по смещённым полиномам Чебышева
C
c  a  T(L(key),x) полином Чебышева L(key)-го порядка
cz
C ier - код завершения работы функции.
C Лит.-ра: Ю.Люк Специальные математические функции и их
C аппроксимации. "МИР", МОСКВА, 1980, 608 С. СТР. 511.
C
      function dcheb0(a,x,n,key,ier)
      implicit none
      integer ier, n, n1, key, i, k
      real*8 a(n+1), x, z, b0, b1, b2, dcheb0
      ier=0
      if (n.le.0) ier=1                    ! Контроль числа слагаемого (n):
      if ((key.lt.0).or.(key.gt.3)) ier=2 ! Контроль ключа (key):
      if (ier.ne.0) return
      n1=n+1
      if (key.eq.0) z=2d0*x
      if (key.eq.1) z=4d0*X*X-2d0
      if (key.eq.2) z=4d0*X*X-2d0
      if (key.eq.3) z=4d0*X-2d0
      b0=0d0
      b1=0d0
      do i=1,n1
        k=n1-i
        b2=b1
        b1=b0
        b0=z*b1-b2+a(k+1)
      enddo
      if (key.eq.0) dcheb0=b0-x*b1
      if (key.eq.1) dcheb0=x*(b0-b1)
      if (key.eq.2) dcheb0=b0-b1*z/2d0
      if (key.eq.3) dcheb0=b0-b1*z/2d0
      return
      end
```

Исходные тексты подпрограмм **csintegr** и **dcheb0** нацелены не на экономию машинного времени, а на простоту применения **dcheb0** для расчёта значений различных функций по известным коэффициентам их разложений по полиномам Чебышёва на основе схемы суммирования, изложенной в книге [15].

```
      program tstcosin                    ! Программа тестирует алгоритмы
      implicit none                       ! расчета интегрального косинуса.
      integer nres                        ! Первый -- функция cosin0(x) --
      real*8 g /0.5772156649015328606 1d0/ ! ведет расчет
      real*8 x,r1,r2,a,b,h, pi, si,ci     !              ci-gamma-ln(x)
      real*8 cosint0, cosint1, dre        !              --------------
      real*8 sinint0, sinint1, dim        !                     x^2
      integer n, i, ier                   ! через разложение в ряд Маклорена.
      pi=4*datan(1d0)
      n=8                                 ! Второй -- функция cosin1(x) --
      a=-8d0                              ! через разложение числителя
      b=8d0                               ! указанной дроби по полиномам
      h=(b-a)/n                           ! Чебышева четного порядка
      write(*,1200)
      do i=0,n                            !
        x=a+i*h                           ! Сравнение cosint0(x) с
        r1=cosint0(x)*x*x                 ! cosint1(x) - ln(gamma) - ln(x).
        r2=-cosint1(x)
        write(*,1201) i, x, r2, dabs(r1-r2)
      enddo
      write(*,1300)
      do i=1,n
        x=i*h
        r1=cosint0(x)*x*x+g+dlog(x)
        r2=-cosint1(x)+g+dlog(x)
        write(*,1201) i, x, r2, dabs(r1-r2)
      enddo
      write(*,1500)                    !
      do i=0,n                         !   Сравнение sinint0(x) с sinint1(x)
        x=a+i*h                        !
        r1=sinint0(x)*x                !
        r2=sinint1(x)
        write(*,1201) i, x, r2, dabs(r1-r2)
      enddo
      write(*,1600)                    !
      do i=5,8                         !   csinteg6(x,dre, dim)
        x=i                            !
        call csintegr(x, dre, dim,ier)
        r1=-(dcos(x) * dim - dsin(x) *dre) / x
        r2=pi/2 -(dcos(x)*dre+dsin(x)*dim)/x
        write(*,1301) i, x, r2, dabs((r1-r2)/r2)
      enddo
      write(*,1600)
      do i=10,100,10;  x=i; r1=si(x); r2=ci(x)
         write(*,1301) i, x, r1, r2
      enddo
 1100 format(1x,' #  a=',d15.7,'  b=',d15.7,'  n=',i5)
 1200 format(1x,' #  i',4x,'x',19x,'r1',23x,'r2')
 1201 format(1x,2x,i3,d11.3, d25.16, d18.3)
 1301 format(1x,2x,i3,d11.3, d25.16, d25.16)
 1300 format(//1x,' #  i',4x,'x',16x,'Ci Cheb',19x,'Ci Mac')
 1500 format(//1x,' #  i',4x,'x',16x,'Si Cheb',19x,'Si Mac')
 1600 format(//1x,' #  i',4x,'x',16x,'Si ',25x,'Ci ')
      end
```

#	i	x	r1	r2
	0	-0.800D+01	-0.2534223324049359D+01	0.266D-14
	1	-0.600D+01	-0.2437032378022835D+01	0.444D-15
	2	-0.400D+01	-0.2104491723908354D+01	0.888D-15
	3	-0.200D+01	-0.8473820166866131D+00	0.333D-15
	4	0.000D+00	0.0000000000000000D+00	0.000D+00
	5	0.200D+01	-0.8473820166866131D+00	0.333D-15
	6	0.400D+01	-0.2104491723908354D+01	0.888D-15
	7	0.600D+01	-0.2437032378022835D+01	0.444D-15
	8	0.800D+01	-0.2534223324049359D+01	0.266D-14

#	i	x	Ci Cheb	Ci Mac
	1	0.200D+01	0.4229808287748651D+00	0.333D-15
	2	0.400D+01	-0.1409816978869307D+00	0.888D-15
	3	0.600D+01	-0.6805724389324741D-01	0.444D-15
	4	0.800D+01	0.1224338825320093D+00	0.266D-14
	5	0.100D+02	-0.4545643303446578D-01	0.300D-10
	6	0.120D+02	-0.4978039066667606D-01	0.384D-06
	7	0.140D+02	0.6900605879152000D-01	0.390D-03
	8	0.160D+02	-0.1181826763436713D+00	0.104D+00

#	i	x	Si Cheb	Si Mac
	0	-0.800D+01	-0.1574186821706942D+01	0.222D-15
	1	-0.600D+01	-0.1424687551280507D+01	0.444D-15
	2	-0.400D+01	-0.1758203138949053D+01	0.000D+00
	3	-0.200D+01	-0.1605412976802695D+01	0.444D-15
	4	0.000D+00	0.0000000000000000D+00	0.000D+00
	5	0.200D+01	0.1605412976802695D+01	0.444D-15
	6	0.400D+01	0.1758203138949053D+01	0.000D+00
	7	0.600D+01	0.1424687551280507D+01	0.444D-15
	8	0.800D+01	0.1574186821706942D+01	0.222D-15

#	i	x	Si	Ci
	5	0.500D+01	0.1549931244944674D+01	0.1122605277025322D+01
	6	0.600D+01	0.1424687551280506D+01	0.1047769943544518D+01
	7	0.700D+01	0.1454596614248094D+01	0.9472738505432108D+00
	8	0.800D+01	0.1574186821706942D+01	0.9222240455556282D+00

#	i	x	Si	Ci
	10	0.100D+02	0.1658347594218874D+01	-0.4545643300445537D-01
	20	0.200D+02	0.1548241701043440D+01	0.4441982084535331D-01
	30	0.300D+02	0.1566756540030351D+01	-0.3303241728207115D-01
	40	0.400D+02	0.1586985119354784D+01	0.1902000789620877D-01
	50	0.500D+02	0.1551617072485936D+01	-0.5628386324116305D-02
	60	0.600D+02	0.1586745616259947D+01	-0.4813243377443215D-02
	70	0.700D+02	0.1561594849178006D+01	0.1092198847346498D-01
	80	0.800D+02	0.1572330886912487D+01	-0.1240250115507096D-01
	90	0.900D+02	0.1575663406657456D+01	0.9986124071643133D-02
	100	0.100D+03	0.1562225466889056D+01	-0.5148825142610492D-02

6.4.4 Расчёт C(x) и S(x) (варианты 17, 18, 19, 20, 21).

```
      function cfren1(x)  ! Функция cfren1(x) вычисляет для аргумента
      implicit none       ! ( 0 <= x<= 8 ) косинус-интеграл Френеля через его
      real*8 x, cfren1    ! разложению в ряд по полиномам Чебышева четного
      real*8 a(0:17)      ! порядка. Коэффициенты разложения взяты из книги
      real*8 w,z          ! Ю.Люка  Специальные математические функции и их
      real*8 b0, b1, b2   ! аппроксимации. Издательство "МИР". Москва, 1980.
      integer n, i        !                    стр. 146-147 (таблица 4.8).
      data a / 0.76435 13866 41860 00189d0,
     1 -0.43135 54754 76601 79313d0, 0.43288 19997 97266 53054d0,
     3 -0.26973 31033 83871 11029d0, 0.08416 04532 08769 35378d0,
     5 -0.01546 52448 44613 81958d0, 0.00187 85542 34398 22018d0,
     7 -0.00016 26497 76188 87547d0, 0.00001 05739 76563 83260d0,
     9 -0.00000 05360 93398 89243d0, 0.00000 00218 16584 54933d0,
     1 -0.00000 00007 29016 21186d0, 0.00000 00000 20373 32548d0,
     3 -0.00000 00000 00483 44033d0, 0.00000 00000 00009 86533d0,
     5 -0.00000 00000 00000 17502d0, 0.00000 00000 00000 00272d0,
     7 -0.00000 00000 00000 00004d0 /
      w=0.125d0*x; z=4d0*w*w-2d0
      b0=0d0; b1=0d0
      do i=17,0,-1; b2=b1; b1=b0; b0=z*b1-b2+a(i)
      enddo
      cfren1=b0-b1*z/2d0
      end

      function sfren1(x)  ! Функция sfren1(x) вычисляет для аргумента
      implicit none       ! (0<=x<=8) синус-интеграл Френеля через его
      real*8 x, sfren1    ! разложение в ряд по полиномам Чебышева
      real*8 a(0:16)      ! нечетного порядка. Коэффициенты разложения
      real*8 w,z          ! взяты из книги Ю.Люка: Специальные
      real*8 b0, b1, b2   ! математические функции и их  аппроксимации.
      integer i           ! Издательство "МИР". Москва, 1980.
      data a / 0.63041 40431 45705 39241d0, ! стр. 146-147 (таблица 4.8)
     1 -0.42344 51140 57053 33544d0, 0.37617 17264 33436 56625d0,
     3 -0.16249 48915 45095 67415d0, 0.03822 25577 86330 08694d0,
     5 -0.00564 56347 71321 90899d0, 0.00057 45495 19768 97367d0,
     7 -0.00004 28707 15321 02004d0, 0.00000 24512 07499 23299d0,
     9 -0.00000 01109 88418 40868d0, 0.00000 00040 82497 31696d0,
     1 -0.00000 00001 24498 30219d0, 0.00000 00000 03200 48425d0,
     3 -0.00000 00000 00070 32416d0, 0.00000 00000 00001 33638d0,
     5 -0.00000 00000 00000 02219d0, 0.00000 00000 00000 00032d0 /
      w=0.125d0*x; z=4d0*w*w-2d0
      b0=0d0; b1=0d0
      do i=16,0,-1; b2=b1; b1=b0; b0=z*b1-b2+a(i)
      enddo
      sfren1=w*(b0-b1)
      end
```

Точнее **cfren1(x)** и **sfren1(x)** соответственно вычисляют $\sqrt{\frac{2\pi}{x}} \cdot \int_0^x \frac{\cos t}{\sqrt{t}} dt$ и $\sqrt{\frac{2\pi}{x}} \cdot \int_0^x \frac{\sin t}{\sqrt{t}} dt$ так, что

$$C(x) = \sqrt{\frac{x}{2\pi}} \cdot cfren1(x) \quad ; \quad S(x) = \sqrt{\frac{x}{2\pi}} \cdot sfren1(x)$$

```
 subroutine csfren6(x,dr,di,ier) ! Подпрограмма csfren6(x) для (x>=5)
 implicit none                   ! находит дополнения косинус- и
 real*8 x, dr, di                ! синус- интегралов Френеля путем
 real*8 cr(0:34), ci(0:33)       ! суммирования соответствующего
 real*8 w,dcheb0                 ! разложения в ряд по полиномам
 integer  ier                    ! Чебышева чётного порядка
 data cr / 0.99056 04793 73497 54867d0,  !
1         -0.01218 35098 31478 99746d0,  ! Коэффициенты разложения
2         -0.00248 27428 23113 06034d0,  ! взяты из книги Ю.Люка
3          0.00026 60949 52647 24735d0,  !  Специальные математические
4         -0.00000 10790 68987 40635d0,  ! функции и их аппроксимации.
5         -0.00000 48836 81753 93328d0,  ! Издательство "МИР". Москва,
6          0.00000 09990 55266 36813d0,  ! стр. 147-148 (табл. 4.8).
7 -0.00000 00750 92717 37211d0, -0.00000 00190 79487 57288d0,
9  0.00000 00090 90797 29266d0, -0.00000 00019 66236 03267d0,
1  0.00000 00001 64772 91058d0,  0.00000 00000 63079 71380d0,
3 -0.00000 00000 36423 21895d0,  0.00000 00000 10536 93030d0,
5 -0.00000 00000 01716 43801d0, -0.00000 00000 00107 12365d0,
7  0.00000 00000 00204 09885d0, -0.00000 00000 00090 06395d0,
9  0.00000 00000 00025 50616d0, -0.00000 00000 00004 03556d0,
1 -0.00000 00000 00000 56958d0,  0.00000 00000 00000 76174d0,
3 -0.00000 00000 00000 36288d0,  0.00000 00000 00000 11797d0,
5 -0.00000 00000 00000 02467d0, -0.00000 00000 00000 00016d0,
7  0.00000 00000 00000 00331d0, -0.00000 00000 00000 00203d0,
9  0.00000 00000 00000 00083d0, -0.00000 00000 00000 00025d0,
1  0.00000 00000 00000 00004d0,  0.00000 00000 00000 00001d0,
3 -0.00000 00000 00000 00001d0,  0.00000 00000 00000 00001d0 /
 data ci / 0.04655 77987 37516 45606d0,
1          0.04499 21302 01239 41396d0,
2         -0.00175 42871 39651 45324d0,
3 -0.00014 65340 02581 06784d0,  0.00003 91330 40863 01585d0,
5 -0.00000 34932 28659 77307d0, -0.00000 03153 53003 23452d0,
7  0.00000 01876 58200 85285d0, -0.00000 00377 55280 49302d0,
9  0.00000 00026 65516 50103d0,  0.00000 00010 88144 81222d0,
1 -0.00000 00005 35500 76711d0,  0.00000 00001 31576 54466d0,
3 -0.00000 00000 15286 08809d0, -0.00000 00000 03394 76460d0,
5  0.00000 00000 02702 02670d0, -0.00000 00000 00946 31418d0,
7  0.00000 00000 00207 15651d0, -0.00000 00000 00012 69314d0,
9 -0.00000 00000 00013 97562d0,  0.00000 00000 00008 59293d0,
1 -0.00000 00000 00003 10695d0,  0.00000 00000 00000 75146d0,
3 -0.00000 00000 00000 06478d0, -0.00000 00000 00000 05224d0,
5  0.00000 00000 00000 03864d0, -0.00000 00000 00000 01651d0,
7  0.00000 00000 00000 00504d0, -0.00000 00000 00000 00092d0,
9 -0.00000 00000 00000 00011d0,  0.00000 00000 00000 00020d0,
1 -0.00000 00000 00000 00011d0,  0.00000 00000 00000 00005d0,
3 -0.00000 00000 00000 00001d0  /
 w=5/x
 dr=dcheb0(cr,w,22,3,ier)
 if (ier.eq.0) di=dcheb0(ci,w,21,3,ier)
 end
```

```
      program testfren6            ! Программа тестирует алгоритмы расчета
      implicit none                ! интегралов Френеля двумя способами.
      integer ninp, nres           ! Первый -- функция cfren0(x) --  ведет
      real*8 x,r1,r2,a,b,h         ! расчет через разложение в ряд Маклорена.
      real*8 dre0, dim0, cx0, sx0  ! Второй -- функция cfren1(x) -- через
      real*8 dc, ds, sx, z         ! разложение по полиномам Чебышева четного
      real*8 cfren0,cfren1,sfren1  ! порядка при рабочем диапазоне аргумента:
      real*8 sfren0, r3
      real*8 spp, spi2, pi, dpi2   !               0 <= x <= 8.
      integer n, i
      a=0; b=8; n=8; write(*,1100) a, b, n
      h=(b-a)/n
      pi=4*datan(1d0); spp=dsqrt(2*pi);  spi2=dsqrt(2/pi)
      write(*,1200)
      do i=0,n                     ! Тестирование косинус-интеграла Френеля:
        x=a+i*h                    !       /2pi\                cos(t)
        r1=cfren0(x)               !  sqrt(---- )*C2(x)=[0,x) ------- dt
        r2=cfren1(x)               !       \ x /              sqrt(t)
        write(*,1201) i, x, r2, dabs(r1-r2)
      enddo
      write(*,1300)                !  Синус- косинус-интегралы Френеля
      do i=0,n                     !  на промежутке [0,8]
        x=a+i*0.1d0                !
        r1=sfren1(x) * dsqrt(x) / spp
        r2=cfren1(x) * dsqrt(x) / spp
        write(*,1301) i, x, r2, r1
      enddo
      write(*,'(//1x,"#",40x,"При x>=5 вычисляем dc и ds:")')
      write(*,'(1x,"# C(x)=(sqrt(pi/2)-dc/sqrt(x))/sqrt(2*pi)")')
      write(*,'(1x,"# S(x)=(sqrt(pi/2)+ds/sqrt(x))/sqrt(2*pi)"//)')
      write(*,'(1x,"#   x",12x,"C(x,dc)",17x,"S(x,ds)",3x)')
      dpi2=dsqrt(pi/2)
      do i=0,10
        x=5+i
        call csfren6(x,dre0,dim0,nres)
        cx0=dcos(x)
        sx0=dsin(x)
        sx=1d0/dsqrt(x)
        dc=sx*(dim0*cx0-dre0*sx0)
        ds=-sx*(sx0*dim0+cx0*dre0)
        write(*,'(f7.2,d25.16,d25.16)')
     >         x, (dpi2-dc)  / spp, (dpi2+ds) / spp
      enddo
 1100 format(1x,' #  a=',d15.7,'  b=',d15.7,'  n=',i5)
 1200 format(1x,' #  i',4x,'x',7x,'sqrt(2pi/x)*C2(x) Чеб',6x,'err(Мак)')
 1201 format(1x,2x,i3,d11.3, d23.16, d14.3)
 1300 format(//1x,' #  i',4x,'x',18x,'C(x)',18x,'S(x)')
 1301 format(1x,2x,i3,d11.3, d23.16, d23.16)
      end
```

```
 #  a=  0.0000000D+00  b=  0.8000000D+01  n=    8
 #  i    x       sqrt(2pi/x)*C2(x) Чеб      err(Мак)
    0  0.000D+00 0.2000000000000000D+01     0.000D+00
    1  0.100D+01 0.1809048475800544D+01     0.222D-15
    2  0.200D+01 0.1335193696294337D+01     0.222D-15
    3  0.300D+01 0.8119100277625464D+00     0.111D-15
    4  0.400D+01 0.4614614624332163D+00     0.000D+00
    5  0.500D+01 0.3681992994700683D+00     0.167D-15
    6  0.600D+01 0.4536139761247247D+00     0.000D+00
    7  0.700D+01 0.5590856547499323D+00     0.888D-15
    8  0.800D+01 0.5665659793916513D+00     0.444D-15


 #  i    x                  C(x)                   S(x)
    0  0.000D+00 0.0000000000000000D+00 0.0000000000000000D+00
    1  0.100D+00 0.2520610557345520D+00 0.8404436192811759D-02
    2  0.200D+00 0.3554001646459753D+00 0.2372044135830648D-01
    3  0.300D+00 0.4331025521178710D+00 0.4342179944862118D-01
    4  0.400D+00 0.4966120675730535D+00 0.6651848300841634D-01
    5  0.500D+00 0.5502471546450064D+00 0.9236576020981417D-01
    6  0.600D+00 0.5961570868448002D+00 0.1204654399344022D+00
    7  0.700D+00 0.6355814754703859D+00 0.1503961102304534D+00
    8  0.800D+00 0.6693095468747258D+00 0.1817820356386231D+00


#                                       При x>=5 вычисляем dc и ds:
# C(x)=(sqrt(pi/2)-dc/sqrt(x))/sqrt(2*pi)
# S(x)=(sqrt(pi/2)+ds/sqrt(x))/sqrt(2*pi)


#   x            C(x,dc)                 S(x,ds)
  5.00   0.3284566248675526D+00   0.4659414967662584D+00
  6.00   0.4432738563376234D+00   0.3498523653539780D+00
  7.00   0.5901160610939774D+00   0.3811944739449676D+00
  8.00   0.6393012479306049D+00   0.5120096184674641D+00
  9.00   0.5608039810639548D+00   0.6172135970241897D+00
 10.00   0.4369639527293821D+00   0.6084362590651110D+00
 11.00   0.3803918718581843D+00   0.5047863386473420D+00
 12.00   0.4345573415131012D+00   0.4058110077591432D+00
 13.00   0.5425104114007678D+00   0.3982677211084485D+00
 14.00   0.6047209589342835D+00   0.4817694215597443D+00
 15.00   0.5693360588834202D+00   0.5758032698078055D+00
```

6.5 Решения задач

7 Приложение II. Суммирование числового ряда

1. Основные понятия.
2. Пример сходящегося ряда.
3. Пример расходящегося ряда.
4. Основная идея суммирования рядов.
5. Оценка остатка знакопостоянного и знакопеременного рядов.
6. Расчёт $\zeta(2)$ простым суммированием числовых рядов (ФОРТРАН).
7. Расчёт $\zeta(2)$ простым суммированием числовых рядов (СИ).
8. Некоторые вопросы.

7.1 Основные понятия

Ряд – это сумма бесконечного числа слагаемых

$$S = a_1 + a_2 + a_3 + \cdots + a_k + \cdots = \sum_{k=1}^{\infty} a_k.$$

Ряд называется

1. **числовым**, если все a_k – числа, или
2. **функциональным**, если a_k – некоторые функции, т.е.

$$S(x) = \sum_{k=1}^{\infty} a_k(x).$$

Частичной суммой ряда называется сумма его первых n слагаемых

$$S_n = \sum_{k=1}^{n} a_k \quad , \qquad\qquad (n = 1, 2, \cdots).$$

Ряд называется **сходящимся**, если существует конечный предел последовательности его частичных сумм, т.е.

$$\lim_{n\to\infty} S_n = S < \infty.$$

Предел S называется суммой ряда.
Условия сходимости числовых рядов

1. необходимое: $\lim\limits_{k\to\infty} a_k = 0$;
2. достаточное: стремление к нулю общего элемента a_k должно быть достаточно сильным.

7.2 Пример сходящегося ряда

Сумма элементов бесконечно убывающей геометрической прогрессии:

$$S = 1 + \lambda + \lambda^2 + \cdots + \lambda^k + \cdots = \sum_{k=0}^{\infty} \lambda^k \quad , |\lambda| < 1$$

$$S_n = \frac{(1-\lambda^n)}{1-\lambda} \quad ; \qquad \lim_{n\to\infty} S_n = \frac{1}{1-\lambda} \equiv S.$$

1. В прикладных задачах величину $\lambda \in [0,1]$ можно толковать как вероятность реализации некоторого исследуемого процесса. В теории переноса излучения через λ часто обозначается вероятность выживания кванта (доля излучения однократно рассеянного элементарным объемом). Тогда величина $1-\lambda$ может считаться вероятностью гибели кванта, а отношение $\dfrac{1}{1-\lambda}$ – средним числом рассеяний кванта в бесконечной среде. Каждое слагаемое разложения $\dfrac{1}{1-\lambda}$ дает вклад рассеяния соответствующей кратности в среднее число рассеянний.

2. Разложение $(1-\lambda)^{-1}$ в ряд производит дискретную бесконечную последовательность числовых коэффициентов при степенях λ: **{ 1,1,1,...}**. Поэтому функцию $(1-\lambda)^{-1}$ называют **производящей** по отношению к этой последовательности.

3. Понятие **производящей функции** широко используется в математике в качестве формального средства получения нужной последовательности. Поэтому нет нужды с обязательностью требовать сходимости соответствующего ряда. Например, просто получить **производящую функцию** для последовательности **чисел Фибоначчи**: $\mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{5}, \mathbf{8}, \cdots$, в которой первые два элемента – числа **0** и **1**, а каждый последующий равен сумме двух предыдущих. Следуя [17], имеем по определению понятия **производящей функции**

$$\begin{aligned} G(z) &= F_0 + F_1 z^1 + F_2 z^2 + F_3 z^3 \cdots \\ zG(z) &= F_0 z^1 + F_1 z^2 + F_2 z^3 + \cdots \\ z^2 G(z) &= F_0 z^2 + F_1 z^3 + \cdots \end{aligned}$$

Вычитая из первого равенства второе и третье имеем:

$$G(z)(1-z-z^2) = F_0 + z(F_1 - F_0) + z^2(F_2 - F_1 - F_0) + z^3(F_3 - F_2 - F_1) + \cdots$$

Поскольку за исключнением первых двух каждый очередной элемент последовательности чисел Фибоначчи равен сумме двух предыдущих, то все содержимое всех круглых скобок кроме втрого слагаемого справа равно нулю. Так как $F_0 = 0$ и $F_1 = 1$, то имеем:

$$G(z) = \frac{z}{1-z-z^2}$$

Коэффициенты разложения G(z) в ряд Маклорена есть числа Фибоначчи. Много интересного об использовании производящих функциях можно прочитать в [17] и [16]. В качестве производящей функции для слагаемых гармонического (расходящегося) ряда может служить функция **ln(1-x)**.

7.3 Пример расходящегося ряда – гармонический ряд:

$$H = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k} + \cdots = \sum_{k=1}^{\infty} \frac{1}{k}.$$

Частичные суммы гармонического ряда называют гармоническими числами H_n:

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k}.$$

H_n при достаточно большом n может стать больше любого наперед заданного числа.

$$H_{2^{m+1}} > 1 + \frac{m}{2}.$$

Для H_n справедлива оценка: $\quad H_n = ln(n) + \gamma + \dfrac{1}{2\pi} - \dfrac{1}{12n^2} + \dfrac{1}{120n^4} - \epsilon \ \ ,$

где $\gamma = 0.5772156649$ – постоянная Эйлера, а $0 < \epsilon < \dfrac{1}{252n^6}$.

Гармонический ряд можно рассматривать как предельный случай широко известной в математике дзета–функции Римана

$$H_{\infty}^{(r)} = \zeta(r) = 1 + \frac{1}{2^r} + \frac{1}{3^r} + \cdots + \frac{1}{n^r} \ \ , \qquad r > 1.$$

Последний ряд сходится при любом вещественном $r > 1$. При четных r значения ζ–функции выражаются через числа Бернулли:

$$\zeta(2n) = \frac{1}{2}|B_{2n}|\frac{(2\pi)^{2n}}{(2n)!} \ \ ; \ \ \zeta(2) = \frac{\pi^2}{6} \ \ ; \ \ \zeta(4) = \frac{\pi^4}{90} \ \ ; \ \ \zeta(6) = \frac{\pi^6}{945} \ \ ; \ \ \zeta(8) = \frac{\pi^8}{9450}.$$

Числа Бернулли возникают при исследовании сумм одинаковых степеней натуральных чисел. Для чисел Бернулли известна рекуррентная формула:

$$B_n = \frac{-1}{n+1} \sum_{k=0}^{n} C_{n+1}^{k+1} \cdot B_{n-k}$$

Числа Бернулли B_{2m+1} (с нечётными индексами), кроме B_1, равны нулю.

При $1 < r \leq 2$ ряд для $\mathbf{H}_{\infty}^{(\mathbf{r})}$ является медленно сходящимся, так что расчет его суммы простым суммированием невыгоден (см. пункты 7 и 8 этой главы).

Примерами формально расходящихся рядов могут служить так называемые асимптотические ряды, частичные суммы которых, тем не менее, с высокой степенью точности позволяют вычислять в определенной области изменения аргумента соответствующие функции.

В частности, асимптотическая формула Эйлера-Маклорена широко применяется для суммирования медленно сходящихся рядов (см. главу **О суммировании асимптотических рядов**).

7.4 Основная идея суммирования рядов.

Дан сходящийся числовой ряд

$$S = \sum_{k=1}^{\infty} a_k.$$

Представим его в виде

$$S = S' + R_n \quad ,$$

где S'– сумма первых n слагаемых:

$$S' = \sum_{k=1}^{n} a_k \quad ,$$

а R_n – сумма всех остальных (остаток ряда):

$$R_n = \sum_{k=n+1}^{\infty} a_k = S - S'.$$

Получить величину суммы с требуемой точностью значит добиться необходимой малости остатка R_n, т.е. выполнения условия:

$$|R_n| \leq \epsilon_{abs} \quad ,$$

где $\epsilon_{abs} > 0$ – максимально допустимая величина остатка, при которой величину S' можно считать приближенным значением суммы S с заданной предельной абсолютной погрешностью ϵ_{abs}. Условие прекращения процесса уточнения суммы по требуемой предельной относительной погрешности метода ϵ_{rel} имеет вид

$$|R_n| \leq \epsilon_{rel} \cdot |S'| \quad .$$

Так что, если найдена явная зависимость R_n (или подходящей мажоранты) от n, то из последних неравенств можно оценить n, при котором метод гарантирует достижение требуемой точности.

Замечания:

1. Задание абсолютной погрешности метода требует предварительной оценки абсолютной величины искомой суммы (грубо, с точностью до порядка величины).

2. Задание предельной относительной погрешности метода подобной оценки не требует и гарантирует получение нужного количества верных цифр в результате независимо от его величины.

3. Абсолютная и относительная погрешности метода не учитывают влияния вычислительной погрешности, связанной с конечностью разрядной сетки. Поэтому упомянутыми критерии хороши, если известно, что накоплением вычислительной погрешности суммы можно пренебречь.

7.5 Оценка остатка числового ряда.

В случае знакопостоянного ряда (т.е. с неизменным знаком общего элемента) для оценки остатка часто бывает выгодно применение следуюшей теоремы.

Теорема 1.
Если $a_k = f(k) \quad , \quad (k = 1, 2, \cdots)$, где $f(x) > 0$ – монотонно убывающая функция $f(x) > 0$, то

$$\int_{N+1}^{\infty} f(x)dx \le R_n \le \int_{N}^{\infty} f(x)dx.$$

Теорема полезна, если интегралы оцениваются аналитически.
Так, если $a_k \le k^{-r}$ при $r > 1$, то выбираем $f(x) = x^{-r}$. Тогда

$$R_n \le \int_{n}^{\infty} x^{-r}dx = \left.\frac{x^{-r+1}}{-r+1}\right|_{n}^{\infty} = \frac{1}{(r-1)n^{r-1}}$$

и достаточное для достижения требуемой **абсолютной погрешности метода** ϵ_{abs} количество слагаемых n оценивается по формуле

$$n_{min} \ge \left[\frac{1}{(r-1)\cdot\epsilon_{abs}}\right]^{\frac{1}{r-1}} + 1.$$

Используем наименьшее значение n_{min}, удовлетворяющее последнему условию, при расчете через суммирование знакопостоянного ряда для ζ–функции Римана значения

$$\zeta(2) = \frac{\pi^2}{6} = \sum_{k=1}^{\infty}\frac{1}{k^2} \approx 1.644934066848226$$

и проследим эффекты, обусловленные конечностью разрядной сетки.

Расчет выполним двумя способами. В первом суммирование ведётся в направлении уменьшения слагаемых; во втором — в направлении увеличения.

Каждая строка приведенной ниже таблицы содержит:

1. Требуемую предельную абсолютную погрешность метода суммирования (eps).
2. Минимальное количество слагаемых достаточное для ее достижения (n_+) при суммировании знакоположительного ряда.
3. Результат ($\zeta_+^{\rightarrow} \equiv \sum_{k=1}^{n_+}$) при суммировании в направлении уменьшения слагаемых и его фактическая погрешность.
4. Результат ($\zeta_+^{\leftarrow} \equiv \sum_{k=n_+}^{1}$) при суммировании в направлении увеличения слпгаемых и его фактическая погрешность.

eps	n_+	$\zeta_+^{\rightarrow}$	$\frac{\zeta_+^{\rightarrow}-\zeta(2)}{\zeta(2)}$	$\zeta_+^{\leftarrow}$	$\frac{\zeta_+^{\leftarrow}-\zeta(2)}{\zeta(2)}$
10^{-1}	11	1.558032	.87E-01	1.558032	.87E-01
10^{-2}	101	1.635082	.98E-02	1.635082	.98E-02
10^{-3}	1001	1.643936	.10E-02	1.643936	.10E-02
10^{-4}	10001	1.644725	.21E-03	1.644834	.10E-03
10^{-5}	100001	1.644725	.21E-03	1.644924	.10E-04
10^{-6}	1000001	1.644725	.21E-03	1.644933	.11E-05
10^{-7}	10000001	1.644725	.21E-03	1.644934	.13E-06

Замечаем, что:

1. суммирование в направлении увеличения слагаемых явно точнее суммирования в направлении их уменьшения;

2. вычислительная погрешность последнего при медленно сходящемся знакопостоянном ряде оказывается в тысячу раз хуже погрешности которую гарантирует метод (например, при $eps = 10^{-7}$), хотя формально ЭВМ складывает вполне достаточное (10^7 по оценке остатка) количество слагаемых, каждое из которых записано в ячейке семью верными десятичными цифрами.

   Причина в том, что любое слагаемое (с номером большим некоторого n_{lim} и меньшим n_+) по величине на семь десятичных порядков меньше уже накопленной частичной суммы. Поэтому результат непосредственного прибавления к ней такого слагаемого на одинарной точности фактически эквивалентен прибавлению нуля. Сумма же всех таких малых слагаемых оказывается больше наибольшего из них (равного $1/n_{lim}^2$) на три десятичных порядка. Таким образом, в окончательном результате оказываются потерянными три младших десятичных разряда из семи. Это убедительно демонстрирует суммирование слагаемых в направлении их увеличения, которое позволяет получить результат верный до семи значащих цифр.

3. Непосредственное суммирование медленно сходящихся рядов нецелесообразно из-за временных затрат (суммирование в направлении увеличения слагаемых выгодно при борьбе за последнюю младшую цифру, но не за четыре десятичных разряда из семи ценой сложения десяти миллионов слагаемых).

4. **Медленно сходящиеся ряды суммируются специальными методами, позволяющими достигать очень высокой точности за небольшое число сложений.** Так найти сумму знакопостоянного ряда для $\zeta(2)$–функции Римана на основе формулы Эйлера-Маклорена можно с точностью до 10^{-17} всего за два десятка сложений.

Для оценки остатка **знакочередующегося** (т.е. $a_k \cdot a_{k+1} < 0$) ряда полезна

Теорема 2.

Если ряд $S = \sum_{k=1}^{\infty} a_k$ знакочередующийся и $|a_k|$ образуют монотонно убывающую последовательность, то

$$|R_n| \le |a_{n+1}|,$$

при этом знак остатка совпадает со знаком первого отброшенного элемента.

Так что, условие, гарантирующее достижение точности не хуже требуемой для знакопеременного ряда имеет вид:

$$|a_{n+1}| \le \epsilon_{abs} \qquad \text{или} \qquad |a_{n+1}| \le \epsilon_{rel} * S'.$$

В таблице ниже приведены результаты расчета значения $\zeta(2)$, полученные суммированием соответствующего знакопеременного ряда:

$$\zeta(2) = \frac{\pi^2}{6} = 2 \cdot \sum_{k=1}^{\infty} (-1)^k \frac{1}{k^2}.$$

eps	$n_\pm$	$\zeta_\pm^{\rightarrow}$	$\frac{\zeta_\pm^{\rightarrow}-\zeta(2)}{\zeta(2)}$	$\zeta_\pm^{\leftarrow}$	$\frac{\zeta_\pm^{\leftarrow}-\zeta(2)}{\zeta(2)}$
10^{-1}	4	1.597222	-.48E-01	1.558032	-.48E-01
10^{-2}	11	1.652453	.75E-02	1.635082	.75E-02
10^{-3}	33	1.645825	.89E-03	1.643936	.89E-02
10^{-4}	101	1.645031	.97E-04	1.644834	.97E-03
10^{-5}	317	1.644945	.11E-04	1.644924	.99E-04
10^{-6}	1001	1.644936	.17E-05	1.644933	.11E-05
10^{-7}	3163	1.644934	.35E-06	1.644934	.11E-06

Здесь $n_\pm$ – число слагаемых достаточное для достижения требуемой погрешности метода, а $\zeta_\pm^{\rightarrow}$ и $\zeta_\pm^{\leftarrow}$ обозначают результаты суммирования соответственно в сторону уменьшения абсолютной величины слагаемого и в сторону её увеличения.

Видим, что в данном случае знакопеременный ряд позволяет:

1) достичь той же точности за гораздо меньшее число сложений;

2) добиться в тысячу раз меньшей вычислительной погрешности суммирования в направлении уменьшения абсолютной величины слагаемых нежели при аналогичном суммировании знакопостоянного ряда;

3) избежать при оценке остатка поиска мажоранты и ее интегрирования;

Перечисленные достоинства суммирования знакопеременного ряда теряются, когда искомая сумма горазо меньше по абсолютной величине большинства составляющих ее слагаемых (*Почему?*)

Результаты, приведённые в таблицах, получены программой **tstdzet.f95** (см. следующий пункт), которая, вызывает подпрограммы **dzplus** и **dzminus** расчёта $\zeta(2)$.

Для закрепления навыка работы с единицей компиляции, называемой в современном ФОРТРАНе **модулем**, описания обеих подпрограмм помещены в модуль **dzeta**. В него же помещено описание и инициализация констант удвоенной точности **pi** $= \pi$, **pi2** $= \pi^{\mathbf{2}}$ и **rexact2** $= \zeta(\mathbf{2})$.

7.6 ФОРТРАН-расчёт $\zeta(2)$ простым суммированием ряда

7.6.1 Программа тестирования подпрограмм dzplus и dzminus

```
!======================================================================
! Программа предназначена для тестирования подпрограмм
! dzplus и dzminus  расчета значения дзета-функции Римана
! с аргументом равным 2 через суммирование ее представления
! в виде знакопостоянного и знакопеременного рядов соответственно.
! Цель: проявить на конкретном расчете эффекты,
! обусловленные конечностью разрядной сетки ЭВМ.
! Обратить внимание:
!  1. на поведение реальной точности результата по сравнению с требуемой;
!  2. результат суммирования в направлении УМЕНЬШЕНИЯ слагаемых
!     менее точен чем при суммировании в направлении их УВЕЛИЧЕНИЯ.
!......................................................................
      program tstdzet2
      use dzeta
      implicit none
      real eps, eraf,  erab,  rf,  rb, erafm, erabm, rfm, rbm
      integer nres6 / 6 /
      integer nres8 / 8 /
      integer i, nplus, nminus
      open (unit=6,file='dzplus.res', status='replace')  ! Открытие файлов
      open (unit=8,file='dzminus.res', status='replace')  !          вывода.
      write(nres6,*) 'real(8) результат (rexact2)=',rexact2
      write(nres8,*) 'real(8) результат (rexact2)=',rexact2
      write(nres6,1010); write(nres8,1020)   ! Вывод заголовоков таблиц
      write(nres6,1000); write(nres8,1000)   !              результата.
      eps=1
      do i=1,7
        eps=eps/10                           ! Требуемая точность.
        call dzplus(eps,nplus,rf,rb)         ! Расчет по знакопост. ряду
        eraf=rf-rexact2;   erab=rb-rexact2   !        и его погрешностей;
        call dzminus(eps,nminus,rfm,rbm)     ! Расчет по знакочеред. ряду
        erafm=rfm-rexact2; erabm=rbm-rexact2 !        и его погрешностей;
        write(nres6,1001) i, eps, nplus,   rf,  eraf,  rb,  erab
        write(nres8,1001) i, eps, nminus, rfm, erafm, rbm, erabm
      enddo
      close(nres6)                           ! Закрытие файлов вывода.
      close(nres8)
 1000 format(1x,'i',4x,'eps',9x,'n',9x,'rf',7x,'eraf',10x,'rb',7x,'erab'/)
 1001 format(1x,i2,1x,e6.1,1x,i10,3x,f9.7,e11.3,3x,f9.7,e11.3)
 1010 format(1x,'Ряд знакопостоянный')
 1020 format(1x,'Ряд знакопеременный')
      end
```

7.6.2 Модуль dzeta

```
module dzeta
implicit none
real(8), parameter :: pi =4d0*datan(1d0), pi2=pi*pi, rexact2=pi2/6
real,    private :: rf, rb, dk, dn, zf, zb;
real, private, parameter :: c1=1.0; integer, private :: k, n
contains
!=================================================================
! Подпрограммы dzplus (eps, nval, rforw, rback) и dzminus (с теми же
! аргументами) для заданного  (eps), абсолютной погрешности МЕТОДА,
! вычисляют значения дзета-функции Римана dzeta(2) посредством
! простого суммирования знакопостоянного (dzplus)
!               1          1       1                       pi**2
!         1 + ------  + ------ + ------ + ... = dzeta(2)= ------
!             2**2      3**2     5**2                        6
! и знакопеременного (dzminus)
!         1          1       1            dzeta(2)   pi**2
!   1 - ------  + ------ - ------ + ... = -------- = -----
!       2**2      3**2     5**2             2       2 * 6
! числовых рядов. Каждая из подпрограмм получает два значения суммы
! rforw и rback (в направлениях уменьшения и увеличения слагаемых
! соответственно). nval - число потребовавшихся слагаемых.
!.................................................................
      subroutine dzplus( eps, nval, rforw, rback)
      real eps, rforw, rback; integer nval
      nval=int(1/eps+0.5)+1; n=nval; rf=0; rb=0; k=0
      do; k=k+1; dk=k; rf=rf+c1/(dk*dk); dn=n; rb=rb+c1/(dn*dn); n=n-1
        if (k.ge.nval) exit
      enddo
      rforw=rf; rback=rb
      end subroutine dzplus
      subroutine dzminus (eps, nval, rforw, rback)
      real eps, rforw, rback; integer nval
      nval=int(1 / sqrt(eps)+0.5) +1; n=nval; rf=0; rb=0; k=0
      zf=c1; zb=c1; if (n/2*2.eq.n) zb=-c1
      do; k=k+1; dk= k; rf=rf+zf/(dk*dk); zf=-zf;
                 dn=n; rb=rb+zb/(dn*dn); zb=-zb; n=n-1
        if (k.ge.nval) exit
      enddo
      rforw=rf*2; rback=rb*2
      end subroutine dzminus
end module dzeta
```

Замечание: Атрибут **private** при описании переменных и констант **модуля** означает, что их имена не экспортируются в подсоединяющую **модуль** единицу компиляции, оставаясь доступными исключительно процедурам, описанным в модуле, что позволяет сократить размер модульных процедур.

7.6.3 Вариант Make-файла

```
COMP:=gfortran
OPT:= -c
PATTERN:=*.f95
SOURCE:=$(wildcard $(PATTERN))
OBJ:=$(patsubst %.f95, %.o, $(SOURCE))
main:=tstdzet2
$(main)   : $(obj)
   [TAB]    $(COMP) $^ -o $@
%.o %mod  : %.f95
   [TAB]    $(comp) $(opt) $<
$(main).o : dzeta.mod
result :    $(main)
   [TAB]    ./$(main)
```

7.6.4 Результаты тестирования dzplus.for

```
 real(8) результат (rexact2)=   1.6449340668482264
 Ряд знакопостоянный
 i    eps         n         rf       eraf          rb       erab

  1 .1E+00         11   1.5580322 -0.869E-01   1.5580322 -0.869E-01
  2 .1E-01        101   1.6350820 -0.985E-02   1.6350819 -0.985E-02
  3 .1E-02       1001   1.6439358 -0.998E-03   1.6439356 -0.999E-03
  4 .1E-03      10001   1.6447253 -0.209E-03   1.6448340 -0.100E-03
  5 .1E-04     100001   1.6447253 -0.209E-03   1.6449240 -0.100E-04
  6 .1E-05    1000001   1.6447253 -0.209E-03   1.6449330 -0.108E-05
  7 .1E-06   10000003   1.6447253 -0.209E-03   1.6449339 -0.128E-06
```

7.6.5 Результаты тестирования dzminus.for

```
 real(8) результат (rexact2)=   1.6449340668482264
 Ряд знакопеременный
 i    eps         n         rf       eraf          rb       erab

  1 .1E+00          4   1.5972222 -0.477E-01   1.5972222 -0.477E-01
  2 .1E-01         11   1.6524533  0.752E-02   1.6524533  0.752E-02
  3 .1E-02         33   1.6458246  0.890E-03   1.6458246  0.890E-03
  4 .1E-03        101   1.6450311  0.970E-04   1.6450311  0.970E-04
  5 .1E-04        317   1.6449449  0.108E-04   1.6449440  0.989E-05
  6 .1E-05       1001   1.6449357  0.166E-05   1.6449351  0.106E-05
  7 .1E-06       3163   1.6449344  0.349E-06   1.6449342  0.111E-06
```

7.7 СИ-расчёт $\zeta(2)$ простым суммированием ряда

7.7.1 Программа тестирования функций dzplus.c и dzminus.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void dzplus (float,int*,float*,float*);  // прототип dzplus;
void dzminus(float,int*,float*,float*);  // прототип dzminus;
int main()
{ /* ================================================================
 Программа предназначена для тестирования подпрограмм
 dzplus и dzminus  расчета значения дзета-функции Римана
 с аргументом равным 2 через суммирование ее представления
 в виде знакопостоянного и знакопеременного рядов соответственно.
 Цель: проявить на конкретном расчете эффекты, обусловленные конечностью
       разрядной сетки ЭВМ (сравни с ФОРТРАН-программой).
.....................................................................*/
double rexact2;
float  eraf,  erab,  rf,  rb, erafm, erabm, rfm, rbm, eps;
FILE   *res6, *res8;
int i, nplus, nminus; char s=' ';
res6=fopen("dzplus.res" ,"wt");        // инициализация указателей
res8=fopen("dzminus.res","wt");        // на структуры файлов вывода;
if (!res6){ printf("\n file not created \n"); exit(6); }
if (!res8){ printf("\n file not created \n"); exit(8); }
rexact2=M_PI*M_PI/6.0;                              // double-результат
fprintf(res6," double результат (rexact2)=%24.16le\n",rexact2);
fprintf(res8," double результат (rexact2)=%24.16le\n",rexact2);
fprintf(res6,"Ряд знакопостоянный\n");
fprintf(res8,"Ряд знакопеременный\n");
fprintf(res6," i%3c eps %9c n %3c rf %5c eraf %6c rb %5c erab\n",s,s,s,s,s,s);
fprintf(res8,
    " i%3c eps %9c n %3c rf %5c eraf %6c rb %5c erab\n",s,s,s,s,s,s);
eps=1.0;
for (i=0;i<=6;i++)                  // цикл по номерам погрешностей:
  { eps=eps/10;                    // выборка требуемой точности;
    dzplus(eps,&nplus,&rf,&rb);      eraf=rf-rexact2;   erab=rb-rexact2;
    dzminus(eps,&nminus,&rfm,&rbm); erafm=rfm-rexact2; erabm=rbm-rexact2;
    fprintf(res6,"%2d %6.1e %10d %9.7f %10.3e %9.7f %10.3e \n",
            i, eps, nplus,   rf,  eraf,   rb,  erab);
    fprintf(res8,"%2d %6.1e %10d %9.7f %10.3e %9.7f %10.3e \n",
            i, eps, nminus,  rfm, erafm, rbm,  erabm);
   }
fclose(res6);
fclose(res8);
return 0;
}
```

7.7.2 Подпрограмма dzplus (знакопостоянное суммирование)

```
void dzplus(float eps,int *nval, float *rforw, float *rback)
{ float rf, rb, dk, dn, c1=1e0;
  int n,k;
  *nval=(int)(c1/eps+0.5)+1; n=*nval;
  rf=0; rb=0; k=0;
  do
    { k=k+1;  dk=k;  rf=rf+c1/(dk*dk);
               dn=n;  rb=rb+c1/(dn*dn); n=n-1;
    }
  while (k<*nval);
  *rforw=rf;
  *rback=rb;
}
```

7.7.3 Подпрограмма dzminus (знакопеременное суммирование)

```
#include <math.h>
void dzminus(float eps, int *nval, float *rforw, float *rback)
{ float c1=1.0, rf, rb, zf, zb, dk, dn;
  int n, k;
  *nval=(int)(c1/sqrt(eps)+0.5) +1; n=*nval;
  rf=0; rb= 0; k=0;
  zf=c1; zb=c1; if (n%2==0) zb=-1;
  do
    { k=k+1; dk=k; rf=rf+zf/(dk*dk); zf=-zf;
              dn=n; rb=rb+zb/(dn*dn); zb=-zb; n=n-1;
    }
  while (k<*nval);
  *rforw=rf*2;
  *rback=rb*2;
}
```

7.7.4 Вариант make-файла

```
COMP   :=gfortran
OPT    := -c
PATTERN:=*.c
SOURCE :=$(wildcard $(PATTERN))
OBJ    :=$(patsubst %.c, %.o, $(SOURCE))
main:=tstdzet2
$(main) : $(OBJ)
 [ TAB ]  $(COMP) $^ -lm -o $@
   %.o  : %.c
 [ TAB ]  $(COMP) $(OPT) $<
 result : $(main)
 [ TAB ]  time -p  ./$(main)
```

7.7.5 Результаты тестирования dzplus.c

```
 double результат (rexact2)=  1.6449340668482264e+00
Ряд знакопостоянный
 i    eps           n     rf       eraf        rb       erab
 0 1.0e-01         11 1.5580322 -8.690e-02 1.5580322 -8.690e-02
 1 1.0e-02        101 1.6350820 -9.852e-03 1.6350819 -9.852e-03
 2 1.0e-03       1001 1.6439358 -9.983e-04 1.6439356 -9.985e-04
 3 1.0e-04      10001 1.6447253 -2.087e-04 1.6448340 -1.000e-04
 4 1.0e-05     100001 1.6447253 -2.087e-04 1.6449240 -1.002e-05
 5 1.0e-06    1000001 1.6447253 -2.087e-04 1.6449330 -1.082e-06
 6 1.0e-07   10000002 1.6447253 -2.087e-04 1.6449339 -1.279e-07
```

7.7.6 Результаты тестирования dzminus.c

```
 double результат (rexact2)=  1.6449340668482264e+00
Ряд знакопеременный
 i    eps           n     rf       eraf        rb       erab
 0 1.0e-01          4 1.5972222 -4.771e-02 1.5972222 -4.771e-02
 1 1.0e-02         11 1.6524533  7.519e-03 1.6524533  7.519e-03
 2 1.0e-03         33 1.6458246  8.905e-04 1.6458246  8.905e-04
 3 1.0e-04        101 1.6450311  9.703e-05 1.6450311  9.703e-05
 4 1.0e-05        317 1.6449449  1.084e-05 1.6449440  9.886e-06
 5 1.0e-06       1001 1.6449357  1.660e-06 1.6449351  1.064e-06
 6 1.0e-07       3163 1.6449344  3.490e-07 1.6449342  1.106e-07
```

7.8 Некоторые вопросы

1. Какое условие, экономящее время расчёта, естественно добавить в условие выхода из уточняющего цикла в подпрограммах **dzplus** и **dzminus**? Письменно сформулировать его и включить в тексты подпрограмм.

2. Требовать достижения завышенной для используемой разрядности абсолютной погрешности результата:

   а) **в лучшем случае**, т.е. при достаточно быстром убывании слагаемых) — бесполезная трата машинного времени (*Почему?*);

   (а) **в худшем случае** (медленно сходящиеся ряды) суммирования *"в лоб"* правильность результата — сомнительна (*Почему?*).

3. **Метод суммирования** предполагает разрядность **бесконечно большой**, а машинная разрядность — конечна и может оказаться недостаточной для получения верного результата при выбранной схеме суммирования (*пояснить на примере*).

8 Приложение III. Суммирование степенного ряда.

Степенные ряды

$$S(x) = \sum_{k=1}^{\infty} c_k \cdot x^k$$

играют важную роль в теории и в приложениях. В частности, в приложениях, степенные ряды применяются в качестве основы для

1. получения полиномиальных и иных приближений элементарных $(exp(x), ln(x), sin(x), cos(x)$ и др.) и специальных функций $(erf(x), erfc(x), \Gamma(x), E_n(x)$ и др).

2. расчета значения функции из-за отсутствия или неудобства явной формулы (например, наличия в последней арифметического вычитания близких чисел).

3. расчета значения функции по так называемым **асимптотическим** разложениям (см. след. параграф), которые формально являются **расходящимися**.

Множество значений x, на котором степенной ряд сходится, называют **областью сходимости** ряда. При фиксированном x степенной ряд превращается в числовой. Поэтому оценить его остаток можно по теоремам 1 и 2 из пункта **6.5**.

8.1 Примеры степенных рядов элементарных функций

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} \quad , \qquad (-\infty < x < \infty);$$

$$e^{-x} = \sum_{k=0}^{\infty} (-1)^k \frac{x^k}{k!} \quad , \qquad (-\infty < x < \infty).$$

Функция e^x при $(x > 0)$ имеет простое физическое толкование. Именно во столько раз ослабляется пучок света, проходящий в чисто поглощающей (не рассеивающей) среде характерное оптическое расстояние x. Иначе говоря, величина e^{-x} – доля прошедшего излучения, а величина $1 - e^{-x}$ – доля поглощенного. (Какое толкование можно дать величине $\frac{1-e^{-x}}{x}$?)

$$ln(1-x) = \sum_{k=1}^{\infty} \frac{x^k}{k} \quad , \qquad (|x| < 1);$$

$$\frac{1}{2x} \ln \frac{1+x}{1-x} = \sum_{k=0}^{\infty} \frac{x^{2k}}{2k+1} \quad , \qquad (|x| < 1);$$

$$\frac{1}{1-x} = \sum_{k=0}^{\infty} x^k, \qquad (|x| < 1);$$

$$\sqrt{1+x} = 1 + \frac{x}{2} + \sum_{k=2}^{\infty} (-1)^{k-1} \frac{(2k-3)!!}{(2k)!!} x^k \quad , \quad (|x| \leq 1);$$

$$sin(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!} \quad , \qquad (-\infty < x < \infty);$$

$$cos(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!} \quad , \qquad (-\infty < x < \infty);$$

$$\frac{x}{e^x - 1} = 1 - \frac{x}{2} + \sum_{k=1}^{\infty} (-1)^{k-1} \frac{|B_{2k}| x^{2k}}{(2k)!} \quad , \qquad (x < 2\pi).$$

В последней формуле коэффициенты B_{2k} – числа Бернулли:

k	0	1	2	4	6	8	10	12	14	16
B_{2k}	1	$-\frac{1}{2}$	$\frac{1}{6}$	$-\frac{1}{30}$	$\frac{1}{42}$	$-\frac{1}{30}$	$\frac{5}{66}$	$-\frac{691}{2730}$	$\frac{7}{6}$	$-\frac{3617}{510}$

Числа Бернулли с нечетным индексом, кроме $B_1 = -\frac{1}{2}$, равны нулю. Функция $\frac{x}{e^x - 1}$ является производящей для последовательности чисел Бернулли. (Последняя функция может оказаться полезной при расчете функции Планка).

8.2 Примеры степенных рядов специальных функций.

В теоретической астрофизике часто используются специальные функции:

1. E_1 (первая интегрально–показательная функция):

$$E_1(x) = \int_1^\infty e^{-xy}\frac{dy}{y} = -\gamma - ln(x) + x - \frac{x^2}{2\cdot 2!} + \frac{x^3}{2\cdot 3!} - \cdots, \qquad (0 < x < \infty);$$

2. $E_n(x)$ (**n**–я интегрально–показательная функция):

$$E_n(x) = \int_1^\infty e^{-xy}\frac{dy}{y^n} = \frac{(-x)^{n-1}}{(n-1)!}E_1(x) + e^{-x}P(x) \quad ,$$

$$P(x) = \frac{1}{(n-1)!}\sum_{k=0}^{n-2}(-x)^k(n-k-2)!$$

3. $erf(x)$ (интеграл вероятности или интеграл ошибок)

$$erf(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt = \frac{2x}{\sqrt{\pi}}\left[1 - \frac{x^2}{3\cdot 1!} + \frac{x^4}{5\cdot 2!} - \frac{x^6}{7\cdot 3!}\cdots\right], \qquad (x^2 < \infty).$$

4. $erfc(x)$ (дополнительный интеграл вероятности):

$$erfc(x) = 1 - erf(x) = \frac{2}{\sqrt{\pi}}\int_x^\infty e^{-t^2}dt, \qquad (x^2 < \infty).$$

5. J_n (функция Бесселя n-го порядка).

$$J_n(x) = \frac{1}{\pi}\int_0^\pi \cos{(x\cdot\sin\theta - n\theta)}d\theta = \sum_{k=0}^\infty \frac{(-1)^k}{k!(n+k)!}\left(\frac{x}{2}\right)^{n+2k},$$

Здесь $\mathbf{n} \geq \mathbf{0}$ — целое.

8.3 Типичная схема суммирования степенного ряда.

Входные параметры: x – аргумент ($real$);
eps – погрешность ограничения ($real$).
Выходной параметр: основной результат – сумма ряда ($real$).

1. Шаг 1. Инициализация (или присваивание начальных значений):

   1а) счетчика учтенных слагаемых $k = 0$;

   1б) накапливаемой суммы.....$s = 0$;

   1в) начального слагаемого.....$u = ?$ (оно своё у каждого ряда).

2. Шаг 2. Основной цикл накопления суммы:

   do

   2а) учет очередного слагаемого....$s = s + u$;

   2б) расчет следующего слагаемого..$u = ?$ (по рекуррентной формуле, устанавливающей связь между двумя соседними слагаемыми; формула — своя для каждого ряда);

   2в) расчет его номера.....$k = k + 1$;

   2г) Если не достигнута погрешность метода, то возврат на 2а).

   enddo

3. Шаг 3. Формировка окончательного результата:

   3а) если алгоритм оформляется функцией, то возврат результата через ее имя. Последнее удобно, когда известен диапазон допустимых значений аргумента, при котором достигнутая погрешность метода гарантирует правильно характеризует точность результата. Если же этот диапазон не известен, то важно помнить, что факт достижения погрешности метода предполагает бесконечнозначную арифметику. При конечнозначной арифметике вычислительная погрешность может оказаться на много порядков хуже погрешности метода, и тогда формальное достижение последней не характеризует правильность результата;

   3б) если алгоритм оформляется подпрограммой, то в качестве дополнительных результатов могут потребоваться, например, количество слагаемых, величина максимального слагаемого, или характерная медленно изменяющаяся величина, отличающаяся от основного результата некоторым функциональным сомножителем. Часто именно последняя оказывается наиболее удобной по существу дела в расчетных работах (см. пример 3.).

8.4 Суммирование ряда Маклорена для sin(x)

Цель:

1) освоить алгоритм суммирования на примере хорошо знакомой функции;

2) оценить фактическую точность результата, сравнив ее с точностью работы встроенной функции;

3) проявить типичные для суммирования рядов "неожиданные"эффекты, вызванные конечностью разрядной сетки;

4) сформулировать правила безопасности, предупреждающие 3).

Из общей схемы суммирования детализации требуют пункты 1в), 2д) и условие 2г) — продолжения (или прекращения) уточняющего цикла. В случае $\sin(x)$ имеем:

1в) $u = x$

2д) $u_k = (-1)^k \frac{x^{2k+1}}{(2k+1)!}, \quad u_{k+1} = (-1)^{k+1} \frac{x^{2k+3}}{(2k+3)!}.$

$$\frac{u_{k+1}}{u_k} = -\frac{x^2}{(2k+2)(2k+3)}.$$

Таким образом, пункт 2д) сводится к оператору присваивания

$$u = -u * x^2/(2k+2)/(2k+3).$$

2г) Поскольку ряд знакопеременный, то по теореме 2 из пункта **6.5** погрешность ограничения не должна быть больше первого неучтенного слагаемого, т.е. пока $|u_{k+1}| > eps$ сумму следует уточнять.

Алгоритм суммирования оформим подпрограммой-функцией $sinmy(x, eps)$.

8.4.1 Пример 1. Программа тестирования функции sinmy

Перед использованием подпрограммы $sinmy$ надо убедиться в правильности ее работы. Для этого полезно разработать отдельную тестирующую программу.

В данном случае такая программа проста, так как результат работы функции $sinmy(x, eps)$ можно сравнить с результатом работы встроенной функции $sin(x)$. Результаты – многозначные числа. Поэтому на печати выгодно видеть порядок разности $sinmy(x, eps) - sin(x)$, принимая результат, получаемый $sin(x)$, за точный. Ряд для функции $sin(x)$ имеет конечную сумму (т.е. сходиться) при любом конечном x. Оценим качество работы $sinmy$ для следующей выборки значений аргумента

$$x = 0; \quad 0.1; \quad \frac{\pi}{6}; \quad 2\pi + \frac{\pi}{6}; \quad 4\pi + \frac{\pi}{6}; \quad 6\pi + \frac{\pi}{6}; \quad 8\pi + \frac{\pi}{6}$$

.

Как известно, точное значение $\sin(\pi/6 + 2k\pi) = 0.5$.

```
!=====================================================================
! Программа предназначена для тестирования функции sinmy(x,eps),
! реализующей расчет значения sin(x) через суммирование ее ряда Маклорена.
!.....................................................................
      program tstsinmy
      implicit none
      real, parameter :: pi=4*atan(1e0)
      real xx(7) / 0e0, 30e0, 390e0, 750e0, 1110e0, 1470e0, 1830e0/
      real r, res1, res2, err, eps, sinmy
      integer nres / 6 /, i
      open (unit=5,file='input')
      open (unit=nres,file='result', status='replace')
      read(5,100) eps
      write(nres,1000) eps
      do i=1,7
        r=pi*xx(i)/180; res1=sinmy(r,eps); res2=sin(r)
        err=res1-res2
        write(nres,1001) i,xx(i),r,res1,res2,err
      enddo
  100 format(e10.3)
 1000 format(1x,'требуемая абсолютная погрешность (eps)=',e10.3//&
&         1x,'N',2x,' градусы (x)',3x,&
            ' радианы (r)',4x,'sinmy(r,eps)',&
&         5x,'sin(r)',5x,' погрешность'//)
 1001 format(1x,i1,e15.7,e15.7,e15.7,e15.7,e10.2)
      end
```

8.4.2 Функция sinmy

```
!=========================================================
! Подпрограмма-функция sinmy(x,eps) для заданных аргумента (x) и
! абсолютной погрешности ограничения (eps) вычисляет значение
! функции sin(x) по ее разложению в ряд Маклорена.
!.........................................................
      function sinmy( x, eps)
      implicit none
      real sinmy, x, eps, x2, s, u
      integer k, k2
      x2=x*x
      s=0; u=x; k=0
      do
        s=s+u
        k2=k+k; u=-u*x2/(k2+2)/(k2+3)
        k=k+1
        if ( (abs(u).lt.eps)) exit
      enddo
      sinmy=s
      end
```

8.4.3 Уяснение результатов тестирования.

1. Содержимое файла **input**:

```
1.000e-08<--= погрешность метода
```

   Обратите внимание на выгоду форматного ввода в ФОРТРАНе. Форматный ввод при указать позиций строки, из которых необходимо ввести данное, позволяет оставшуюся её часть использовать в качестве комментария, поясняющего смысловую нагрузку данного.

2. Содержимое файла **result**:

```
требуемая абсолютная погрешность (eps)= 0.100E-07

N   градусы (x)    радианы (r)    sinmy(r,eps)     sin(r)      погрешность


1  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.00E+00
2  0.3000000E+02  0.5235988E+00  0.5000000E+00  0.5000000E+00  0.00E+00
3  0.3900000E+03  0.6806785E+01  0.4999886E+00  0.5000005E+00 -0.12E-04
4  0.7500000E+03  0.1308997E+02  0.4994409E+00  0.5000002E+00 -0.56E-03
5  0.1110000E+04  0.1937316E+02  0.6142077E+00  0.5000008E+00  0.11E+00
6  0.1470000E+04  0.2565634E+02  0.6468157E+03  0.4999997E+00  0.65E+03
7  0.1830000E+04  0.3193953E+02  0.1648255E+06  0.5000002E+00  0.16E+06
```

Видно, что для угла в $\mathbf{750}^\circ$ фактическая погрешность расчета почти в 10000 раз хуже требуемой погрешности ограничения, хотя уточнение суммы прекращено в полном соответствии с требованием теоремы 2 из пункта **6.5**, т.е. как только абсолютная величина очередного вычисленного слагаемого оказалось меньше $\mathbf{10^{-7}}$. Для угла равного $\mathbf{1110}^\circ$ и больших величина синуса, найденная **sinmy**, вообще абсурдна.

Причина в недостаточной для правильного расчета разрядности ячейки. Мантиссы и результата, и слагаемых представляются в ячейке семью десятичными цифрами. Если аргумент велик по сравнению с единицей (например, $\mathbf{r = 13,08997}$), то две его старшие цифры из семи верных занимают разряды единиц и десятков, а на дробную часть числа отводится только пять. Именно последние (младшие пять) от первого слагаемого и дадут вклад в старшие пять разрядов результата (ведь $|\sin(\mathbf{r})| < \mathbf{1}$). Остальные две младшие (шестая и седьмая) цифры нормализованного результата к верным не будут иметь никакого отношения, поскольку получены без учета нужных (восьмого и девятого), но отсутствующих в ячейке разрядов, хранящей первое слагаемое. Третье слагаемое порядка 1000 – теряем четвертую цифру мантиссы результата, что и наблюдается в таблице. Так что, реальная погрешность суммы не может оказаться меньше погрешности округления наибольшего из слагаемых. Если оно настолько велико, что погрешность его округления больше величины искомой суммы ряда, то абсурдность суммирования налицо.

8.4.4 Пример 2. Ещё одна демонстрация тестирования sinmy

Причина неудовлетворительной работы **sinmy(x,eps)** при $\mathbf{x} \gg \mathbf{1}$, как уже выяснено в предыдущем пункте, в недостаточной разрядности ячеек, используемых для размещения данных типа **real**.

Корректная работа **sinmy** обеспечивается аргументом **x**, при котором каждое очередное слагаемое оказывалось по абсолютной величине меньше предыдущего, т.е., например, чтобы $\mathbf{x} < \mathbf{1}$ (или чтобы по порядку величины $\mathbf{x} \approx \mathbf{1}$).

Для ещё одной демонстрации происходящего при $\mathbf{x} \gg \mathbf{1}$ модифицируем **sinmy** так, чтобы наряду со значением синуса получать и **umax** — наибольшее по модулю слагаемое (из учитываемых в сумме). Оценим также количество неверных в значении $\mathbf{sin(x)}$ цифр, сопоставляя порядок **umax** с условием $|\mathbf{sin}(\mathbf{x})| \leq \mathbf{1}$.

```
!============================================================
! Программа тестирует функцию sinmy1(x,eps,umax,k,ier) расчета
! sin(x) через суммирование соответствующего ряда Маклорена.
! В качестве дополнительного результата для заданного аргумента
! выводятся umax - значение наибольшего по модулю слагаемого,
! k - число слагаемых, обеспечивающее нужную погрешность метода
! ier - код надёжности вычисленного значения синуса.
!............................................................
      program tstsinmy1
      implicit none
      real, parameter :: pi=4*atan(1e0)
      real xx(7) / 0e0, 30e0, 390e0, 750e0, 1110e0, 1470e0, 1830e0/
      real r, res1, res2, err, eps, sinmy1, umax
      integer nres / 6 /, i, k, ier
      open (unit=5,file='input')
      open (unit=nres,file='result', status='replace')
      read(5,100) eps
      write(nres,1000) eps
      do i=1,7
        r=pi*xx(i)/180
        res1=sinmy1(r,eps,umax,k,ier)
        res2=sin(r)
        err=res1-res2
        write(nres,1001) i,xx(i),res1,err, umax, k, ier
      enddo
  100 format(e10.3)
 1000 format(1x,'требуемая абсолютная погрешность (eps)=',e10.3/&
         1x,'N',2x,' градусы (x)',3x,&
            'sinmy1(r,eps)',&
         6x,'err',7x,' umax',6x,' k',3x, 'ier')
 1001 format(1x,i1,e15.7,e15.7, e12.3,e12.3,i5,i5)
      end
```

После обращения к **sinmy1** о правильности расчета можно судить по значению параметра **ier**.

```
!=========================================================
! Подпрограмма-функция sinmy(x,eps) для заданных значений
! аргумента (x) и абсолютной погрешности ограничения (eps)
! вычисляет sin(x) по ее разложению в ряд Маклорена.
! Дополнительные выходные параметры:
!   k  -- число слагаемых потребовавшееся для достижения eps;
! umax -- максимальное по абсолютной величине слагаемое;
! ier  -- индикатор качества результата:
!  0   -- результат верен в пределах требуемого eps
!  1   -- результат верен, но разрядность низка для заданного eps
!  >1  -- не менее чем ier десятичных разрядов результата неверны.
!.........................................................
      function sinmy1( x, eps, umax, k, ier)
      implicit none
      real sinmy1, x, eps, x2, s, u, au, umax, alumax
      integer ier, k, k2
      ier=0
      x2=x*x; s=0; u=x; au=abs(u); umax=au; k=0
      do
        s=s+u; k2=k+k; u=-u*x2/(k2+2)/(k2+3); au=abs(u)
        if (umax.lt.au) umax=au
        k=k+1
        if ( (au.lt.eps) .or. (s.eq.s+au)) exit
     enddo
     if (s.eq.s+au ) ier=1
     alumax=alog10(umax); if (alumax.gt.1.0) ier=int(alumax)
     sinmy1=s
     end
```

Результат тестирования sinmy1

```
 требуемая абсолютная погрешность (eps)= 0.100E-07
 N   градусы (x)   sinmy1(r,eps)      err        umax       k   ier
 1  0.0000000E+00  0.0000000E+00   0.000E+00   0.000E+00    1    1
 2  0.3000000E+02  0.5000000E+00   0.000E+00   0.524E+00    4    0
 3  0.3900000E+03  0.5000055E+00   0.542E-05   0.134E+03   15    2
 4  0.7500000E+03  0.4999414E+00  -0.588E-04   0.532E+05   24    4
 5  0.1110000E+04  0.2587202E+00  -0.241E+00   0.235E+08   33    7
 6  0.1470000E+04 -0.4480292E+03  -0.449E+03   0.109E+11   42   10
 7  0.1830000E+04 -0.1136961E+06  -0.114E+06   0.524E+13   50   12
```

Ещё раз убеждаемся, что обращаться к функции **sinmy** естественно лишь со значениями аргумента **x**, обеспечивающими при достижении требуемой погрешности метода желаемой фактической точности результата.

В то же время возникает вопрос:

Почему при x $\gg$ 1 встроенная sin(x) гораздо точнее sinmy?

хотя работает (по сути дела) почти по аналогичному алгоритму.

8.4.5 Почему при x ≫ 1 встроенная sin(x) гораздо точнее sinmy?

Встроенная **sin(x)** вычисляет значение синуса вовсе не для исходного аргумента, поданного ей на вход, а для некоего рабочего аргумента, находимого из исходного. Авторы встроенной **sin(x)**, конечно, знали, что результат суммирования степенного ряда для синуса при x ≫ 1 будет удручающим. Однако, они знали и то, что **sin(x)** — периодическая функция с периодом 2π:

$$\mathbf{x}=\mathbf{2k}\pi+\mathbf{r} \quad , \quad \mathbf{sin(x)}=\mathbf{sin(r)},$$

и использовали это свойство для нахождения аргумента **r** — более подходящего для расчёта **sin(x)**, исключая из исходного **x** полное число оборотов $\mathbf{2k}\pi$. Для простоты выполним исключение посредством вызова встроенной функции **amod(x,2π)**, которая находит остаток от деления первого аргумента на второй. Приведём результат тестирования функции **sinmy2(x,eps)** (*написать, отладить и протестировать самостоятельно*), которая указанным способом находит рабочий аргумент **r**.

```
требуемая абсолютная погрешность (eps)= 0.100E-07

N   градусы (x)    радианы (r)    sinmy2(r,eps)     sin(r)      погрешность


1  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.00E+00
2  0.3000000E+02  0.5235988E+00  0.5000000E+00  0.5000000E+00  0.00E+00
3  0.3900000E+03  0.6806784E+01  0.4999999E+00  0.5000001E+00 -0.18E-06
4  0.7500000E+03  0.1308997E+02  0.4999999E+00  0.5000002E+00 -0.36E-06
5  0.1110000E+04  0.1937316E+02  0.5000003E+00  0.5000008E+00 -0.48E-06
6  0.1470000E+04  0.2565634E+02  0.5000007E+00  0.5000013E+00 -0.60E-06
7  0.1830000E+04  0.3193953E+02  0.4999995E+00  0.5000002E+00 -0.77E-06
```

Замечания:

1. На самом деле встроенная фунция **sin(x)** ведёт суммирование по несколько иной и более эффективной схеме (схеме Горнера), не тратя время на перерасчёт коэффициентов при степенях **x**, и ищет аргумент **r**, используя и другие свойства функции **sin(x)**, сводя диапазон рабочего аргумента к промежутку в несколько раз меньшему промежутка $\mathbf{[0,2\pi]}$, что существенно ускоряет время работы встроенной функции и улучшает точность расчёта.

2. Не следует думать, что указанный приём исключения из исходного аргумента полного числа оборотов при $\mathbf{x \gg 1}$ всегда гарантирует правильность всех значащих цифр мантиссы. Если **погрешность** исходного аргумента **x** больше единицы, то вряд ли нас устроит значение **sin(x)**, найденное встроенной функцией.

3. В некоторых руководствах к компиляторам иногда сообщается, что для встроенной функции **sin(x)** никаких ограничений по аргументу нет. Здесь имеется ввиду просто факт работоспособности функции, т.е. значение синуса ею будет получено. Однако насколько оно нас устроит (учитывая вышесказанное) зависит от величины аргумента поданного нами (см. следующий пункт).

8.5 О работе встроенной функции sin(x)

Важно различать возможность формального получения результата работы встроенной функции и возможность получения по ней результата точного в пределах используемой разрядности.

Ниже приведена программа и результат её работы при обращении ко встроенным функциям **sin(x)** и **cos(x)**, вычисляющих значения синуса и косинуса для аргументов кратных π и контрольного соотношения $\mathbf{sin^2(x) + cos^2(x) \equiv 1}$.

```
program testsin; use my_prec; implicit none; real(mp) x, s, c, sum
integer i
write(*,'(" #   mp=",i2)') mp;  x=4*atan(1.0_mp)/10
write(*,'(" # k",7x,"x=pi*10^k",7x,"sin(x)",9x,"cos(x)",10x,"sum")')
do i=0,8;  x=x*10; s=sin(x);  c=cos(x); sum=s*s+c*c
           write(*, '(i4,3x,4(e15.7))'  ) i, x, s, c, sum
enddo
end
```

```
 #   mp= 4
 # k       x=pi*10^k       sin(x)         cos(x)          sum
   0     0.3141593E+01 -0.8742278E-07 -0.1000000E+01  0.1000000E+01
   1     0.3141593E+02  0.1351065E-05  0.1000000E+01  0.1000000E+01
   2     0.3141593E+03  0.5881255E-05  0.1000000E+01  0.1000000E+01
   3     0.3141593E+04  0.1198477E-03  0.1000000E+01  0.1000000E+01
   4     0.3141593E+05  0.1198477E-02  0.9999993E+00  0.1000000E+01
   5     0.3141593E+06  0.1589035E-01  0.9998738E+00  0.1000000E+01
   6     0.3141593E+07  0.9626092E-01  0.9953561E+00  0.1000000E+01
   7     0.3141593E+08  0.9943135E+00  0.1064919E+00  0.1000000E+01
   8     0.3141593E+09 -0.6996488E+00  0.7144869E+00  0.1000000E+01
```

Информация к размышлению:

1. Устроят ли нас значения переменных **s** и **c** при $\mathbf{x \approx 0.314 \cdot 10^9}$ — решать нам!

2. *Якобы контрольный результат* (сумма $cos(x)^2 + sin(x)^2$) — особо опасен тем, что создаёт иллюзию правильности значений **s** и **c**, а они — принципиально ошибочны, так как получены для значения аргумента, которое не имеет никакого отношения к требуемому. Действительно, о какой правильности можно говорить, если для аргумента $x = \pi \cdot 10^9$ (т.е. кратного π) значение синуса равно ≈ -0.700 вместо нуля, а значение косинуса ≈ 0.714 вместо единицы.

3. Получение на основе свойства периодичности функций sin(x) и cos(x) встроенными процедурами **sin(x)** и **cos(x)** рабочего аргумента, когда погрешность представления заданного оказывается порядка единицы, приведёт к рабочему аргументу, отражающему лишь величину этой погрешности, а никак не величину заданного. Ясно, что синус и косинус вычислятся верно именно для аргумента равного этой погрешности, в которой не осталось ни одной верной цифры от изначально заданного числа. Результат перехода на переменные более высокой разрядности (например, **real(16)**) при определённых условиях иногда может трактоваться, как приемлемый:

```
#   mp=16
# k       x=pi*10^k       sin(x)         cos(x)          sum
  0     0.3141593E+01  0.8671810E-34 -0.1000000E+01  0.1000000E+01
  1     0.3141593E+02 -0.8671810E-33  0.1000000E+01  0.1000000E+01
  2     0.3141593E+03  0.3654142E-32  0.1000000E+01  0.1000000E+01
  3     0.3141593E+04  0.3654142E-31  0.1000000E+01  0.1000000E+01
  4     0.3141593E+05  0.1943136E-29  0.1000000E+01  0.1000000E+01
  5     0.3141593E+06  0.6809585E-29  0.1000000E+01  0.1000000E+01
  6     0.3141593E+07  0.6809585E-28  0.1000000E+01  0.1000000E+01
  7     0.3141593E+08  0.6809585E-27  0.1000000E+01  0.1000000E+01
  8     0.3141593E+09  0.6809585E-26  0.1000000E+01  0.1000000E+01
```

— всё-таки число **-26**-го порядка очень мало (гораздо ближе к нулю нежели **0,7**), хотя и в этом случае среди цифр **sin(x)** нет ни одной верной.

4. Для исключения подобных ситуаций следует аналитически находить формулы, подходящие для расчёта рабочего аргумента даже на одинарной точности, и программировать именно их.

5. Заметим также, что при задании ФОРТРАН-констант следует строго придерживаться правил формального синтаксиса. Например, если при **mp=16** вычисление аргумента **x** вести по формуле **x=4*atan(1.0)/10**, то получим

```
#   mp=16
# k       x=pi*10^k       sin(x)         cos(x)          sum
  0     0.3141593E+01 -0.8742278E-07 -0.1000000E+01  0.1000000E+01
  1     0.3141593E+02  0.8742278E-06  0.1000000E+01  0.1000000E+01
  2     0.3141593E+03  0.8742278E-05  0.1000000E+01  0.1000000E+01
  3     0.3141593E+04  0.8742278E-04  0.1000000E+01  0.1000000E+01
  4     0.3141593E+05  0.8742277E-03  0.9999996E+00  0.1000000E+01
  5     0.3141593E+06  0.8742167E-02  0.9999618E+00  0.1000000E+01
  6     0.3141593E+07  0.8731146E-01  0.9961811E+00  0.1000000E+01
  7     0.3141593E+08  0.7670483E+00  0.6415894E+00  0.1000000E+01
  8     0.3141593E+09  0.6307350E+00 -0.7759983E+00  0.1000000E+01
```

что при внимательном рассмотрении должно вызвать недоумение:

Почему значение синуса от аргумента кратного **pi** столь велико при 117-битовой мантиссе (это 35 десятичных значащих цифр)?

Оно по порядку величины должно быть меньше на показатель порядка аргумента (так при $\mathbf{x} \sim \mathbf{10^{+8}}$) в рабочем аргументе должно остаться 35-8=27 верных значащих цифр). Дело в том, что по форме записи **1.0** аргумент арктангенса трактуется как константа одинарной точности, и, следовательно, под родовым именем **atan** (в силу перегрузки функций) понимается специфическое имя **atan**-функции, вычисляющей арктангенс с одинарной точностью.

8.6 Приведение явной формулы к расчетному виду.

Наличие в языках программирования высокоточных и быстрых встроенных алгоритмов расчета элементарных функций вовсе не означает, что никогда не придется использовать соответствующие им ряды при вычислениях на ЭВМ.

Пример 3. Табулирование функции $\mathbf{f(x) = 1 - \frac{\sin(x)}{x}}$.

Предложенная формула проста, но при $\mathbf{x \ll 1}$ требует машинного вычитания почти равных чисел (**почему?**), что чревато потерей всех верных цифр (**почему?**). Для исключения упомянутого вычитания при небольших и малых $\mathbf{x}$ выгодно в качестве метода расчёта использовать разложение функции $\mathbf{1 - \frac{\sin(x)}{x}}$ или даже функции $\left[\mathbf{1 - \frac{\sin(x)}{x}}\right]\frac{\mathbf{1}}{\mathbf{x^2}}$ в ряд Маклорена:

$$\left[1 - \frac{\sin(x)}{x}\right]\frac{1}{x^2} = \sum_{k=0}^{\infty}(-1)^k\frac{x^{2k}}{(2k+3)!} = \left(\frac{1}{3!} - \frac{x^2}{5!} + \frac{x^4}{7!} + \cdots\right).$$

1/6 — наибольшее значение последней, что упрощает задание абсолютной погрешности (**почему?**). Разложение допускает расчёт и при $\mathbf{x{=}0}$. Результат работы программы тестируюшей расчёт функции $[\mathbf{1 - \sin(x)/x}]/\mathbf{x^2}$ по явной формуле и по разложению для погрешности ограничения $\epsilon_{\mathbf{abs}} = \mathbf{10^{-7}}$:

```
требуемая абсолютная погрешность (eps)= 0.100E-06
      x       явная    разложение   ier   k     aer1
 0.100E-09  0.0000000  0.1666667     0    1   0.167E+00
 0.100E-08  0.0000000  0.1666667     0    1   0.167E+00
 0.100E-07  0.0000000  0.1666667     0    1   0.167E+00
 0.100E-06  0.0000000  0.1666667     0    1   0.167E+00
 0.100E-05  0.0000000  0.1666667     0    1   0.167E+00
 0.100E-04  0.0000000  0.1666667     0    1   0.167E+00
 0.100E-01  0.1668930  0.1666658     0    2   0.227E-03
 0.100E+00  0.1665771  0.1665833     0    2   0.626E-05
 0.100E+01  0.1585290  0.1585290     0    4   0.596E-07
 0.200E+01  0.1363378  0.1363378     0    6   0.000E+00
 0.100E+02  0.0105440  0.0105439     0   16   0.112E-06
 0.300E+02  0.0011477  -.3020051     0   41   0.303E+00
```

Видно, что на арифметике одинарной точности:

1) при $\mathbf{x}\epsilon[\mathbf{10^{-5}}, \mathbf{2}]$ результаты расчета разными способами совпадают;

2) при дальнейшем уменьшении аргумента точность явной формулы резко падает;

3) первое слагаемое разложения дает верный результат для любых $\mathbf{x < 10^{-3}}$;

4) с ростом аргумента, начиная с некоторого $\mathbf{x \gg 1}$, вычислительная погрешность суммирования ряда превосходит требуемую погрешность ограничения (**почему?**) и применение ряда для расчета $\mathbf{f(x)}$ на ЭВМ бессмысленно; зато в диапазоне приемлемой точности расчёта $\mathbf{\sin(x)}$ при $\mathbf{x \gg 1}$, оказывается работоспособной исходная формула.

8.6.1 Функция расчёта выражения $(1 - \sin(x)/x)/x^2$

```
!================================================================
! Функция delta(x,eps,k,ier) для заданных аргумента (x) и абсолютной
! погрешности ограничения (eps) вычисляет значение функции
!   [1-sin(x)/x]/x**2 по её разложению в ряд Маклорена:
!                              [k=0,...) (-1)**(2k) x**(2k) / (2k+3)!
! Выходные параметры: искомое значение - через имя функции;
!  k -- количество слагаемых метода достаточное для достижения eps;
!  ier -- код завершения работы подпрограммы:
!   0  -- точность метода достигнута;
!   1  -- на данной разрядности невозможно достичь требуемой eps;
!................................................................
function delta( x, eps, k, ier); implicit none
real delta, x, eps, x2, s, s1, u; real, parameter :: c16=1.0/6.0
integer k, k2, ier
ier=0; x2=x*x; s=0; u=c16; k=0
do; s1=s; s=s+u; k=k+1
    k2=k+k
    u=-u*x2/(k2+2)/(k2+3); if ( (abs(u).lt.eps) .or. (s.eq.s1) ) exit
enddo
if (s.eq.s1) ier=1; delta=s
end
```

Если погрешность очередного слагаемого $> 1/6$ то ier=3 и дальнейший расчёт бессмысленен.

8.6.2 Тестирующая программа

```
!================================================================
! Программа предназначена для сравнения способов расчета функции
!                        (1-sin(x)/x)/x**2.
! Первый способ --  по данной (явной) формуле.
! Второй -- на основе разложения функции в ряд Маклорена.
!.............................................................
real xx(12) /30.0, 10.0, 2.0, 1.0, 0.1, 0.01,1e-5,1e-6,1e-7,1e-8,1e-9,1e-10/
real x, x2, f1, f2, relf1, eps
integer k, ier
read (*,100) eps; write(*,1000) eps
do i=12,1,-1
  x  =xx(i); x2 =x*x
  f1 =(1-sin(x)/x)/x2;  f2 = delta(x,eps,k,ier); aer=abs(f1-f2)
  write(*,1001) x,f1,f2,ier,k,aer
enddo
100 format(e10.3)
1000 format(1x,'требуемая абсолютная погрешность (eps)=',e10.3/&
     1x,6x,'x',7x,'явная',4x,'разложение',3x,'ier',3x,'k',5x,'aer1')
1001 format(1x,e10.3,2x,f9.7,2x,f9.7,5x,i1,i5, e12.3)
end
```

8.7 Информация к размышлению.

Ряд – это аналитически точное, но неявное представление функции. Оно может использоваться и в формульных выкладках для получения аналитически точных результатов, и в качестве метода расчета на ЭВМ. Важно помнить, что, как и явная формула, ряд может быть выгоден для расчета при одних значениях аргумента и невыгоден при других, хотя и те, и другие принадлежат промежутку сходимости ряда. Причина невыгодности – потеря большого количества верных цифр результата в процессе суммирования, обусловленная конечной разрядностью ячейки.

Правила подстраховки:

1. Расчет значения функции через ее разложение в ряд вести только при величине аргумента, гарантирующей правильный численный результат.

2. Расчет очередного слагаемого осуществлять через рекуррентное соотношение между двумя соседними слагаемыми (иначе не избежать появления больших значений числителя и знаменателя, что часто приводит к переполнению).

3. При нарушении правила 2 надежнее не использовать выражения целого типа, так как на них легко пропустить эффект переполнения.

4. В качестве критерия прекращения суммирования наряду с условием малости остатка ряда удобно и условие выхода прибавляемого очередного слагаемого за разрядную сетку накапливаемой суммы.

5. В качестве дополнительного результата порою полезны:

   (a) величина максимального слагаемого, поскольку сравнение ее с величиной суммы знакопеременного ряда позволит оценить возможность серьезной потери точности;

   (b) количество слагаемых необходимое для получения требуемой погрешности ограничения. Чрезмерное их число указывает либо на медленную сходимость, либо на неоправданно завышенную требуемую точность. В любом случае необходимо убедиться, что вычислительная погрешность не больше требуемой погрешности метода).

6. При необходимости минимизировать затраты времени, повысить точность или сохранить ее при уменьшении количества слагаемых выгодно использовать специальные приемы, например, представление функции усеченным рядом не по степеням аргумента x, а по полиномам Чебышова (см., например, [15]). Обычно (при некотором навыке) проще использовать пакеты прикладных программ. Например, **GSL** (GNU Scientific Library) — свободная библиотека с процедурами практически по всем темам числового программирования.

   Однако наличие в них высокопроизводительных и, работающих с высокой точностью, алгоритмов само по себе не всегда может гарантированно обеспечить правильность результатов (приведённые примеры с функциями **sin(x)** и **cos(x)** налицо).

8.8 Подпрограмма расчета функции Фойгта.

```
!============================================================
! Программа тестирует работу подпрограммы  dcerf(x,y,re,rm),
! вычисляющей вещественную (re) и мнимую  rm) части интеграла
! ошибок  W(Z)=EXP(-Z*Z)*ERFC(-I*Z)
! от комплексного аргумента z=x+iy при x>=0 И y>=0.
! Вещественная часть (rm) - функция Ф о й г т а :
!                                 EXP(-T*T)
! RE(W(Z))=U(X,Y)=Y*[-...;+...] ---------------  DT
!                                (X-T)**2 + Y**2
!............................................................
program tst_Voigt
implicit none
interface
subroutine dcerf(x,a,v,w); real(8),intent(in ) :: x,a
                           real(8),intent(out) :: v,w
end subroutine dcerf
end interface
real(8), parameter :: spi=1.772453850905516027 30d0
real(8) a0, ha, am, a
real(8) x0, hx, xn, x, v, w, zv, zw
integer m, n, j, i
read(*,'(e10.4)',end=777) a0, ha, am
read(*,'(e10.4)',end=777) x0, hx, xn
m=idint((am-a0)/ha+1.5d0)
n=idint((xn-x0)/hx+1.5d0)
do j=1,m
  a=a0+(j-1)*ha
  write(*,1000) a, a, a
  do i=1,n
    x=x0+(i-1)*hX
    call dcerf(x,a,v,w)
    zv=v/spi
    zw=w/spi
    write(*,1001) x,v,w,zv,zw
  enddo
  write(*,'(/)')
enddo
777 stop
1000 format(1X,3('A=',D10.4,3X)//&
&  7X,'X',6X,'ФОЙГТ=Re(dcerf)',9x,'IM(dcerf)',6x,'2*Re/sqrt',&
&                                              4x,'2*Im/sqrt'/)
1001 format(1X,e10.3,e20.13,e20.13,e13.6,e13.6)
end
```

```
!=====================================================================
!  dcerf (x,y,re,rm) вычисляет вещественную ( re ) и мнимую ( rm )
! части интеграла ошибок от комплексного аргумента   z = x + i * y
! при  x >= 0  и  y >= 0 :
!                          w( z ) = exp( -z * z ) * erfc( -i*z ),
! где  erfc(z)=1-erf(z).
! Вещественная часть  w( z ) - является функцией Фойгта:
!
!                                          exp(-t*t)
!       Re(w(z))=U(x,y)= y * (-...,+...) -------------- dt      .
!                                        (x-t)**2 +y*y
!
!  Подпрограмма представляет собой переведенный на язык ФОРТРАН
!  АЛГОЛ-алгоритм COMPLEX ERROR FUNCTION  ( Walter Gautchi ) из
!  CACM 1969 vol. 12, N 11 стр. 635, алгоритм 635.
!.....................................................................
subroutine dcerf(x,y,re,rm); implicit none
real(8) x,y,re,rm
real(8),parameter :: c0=0d0, c05=0.5d0, c1=1.0d0
real(8),parameter :: c112=1.128379167095512d0
real(8) s, h, h2, dl, r1, r2, s1, s2, t1, t2, c
integer nc, nu, k, nu1, m, n, np1
if ( (y.ge.4.29d0).or.(x.ge.5.33d0) ) then; h=c0;  nc=0;  nu=8   !goto 10
  else
    s=( c1 - y / 4.29d0 ) * dsqrt( c1-x*x/28.41d0 )
    h=1.6d0 * s
    h2=h+h
    nc=int(6 + 23 * s); nu=int(9 + 21 * s)
endif
if (h.gt.c0) dl=h2**nc
k=1
if ((h.eq.c0).or.(dl.eq.c0)) k=0; r1=c0; r2=c0; s1=c0; s2=c0
nu1=nu+1
do m=1,nu1;  n=nu1-m;  np1=n+1;  t1=y+h+np1*r1;  t2=x-np1*r2
             c=c05/(t1*t1+t2*t2)
             r1=c*t1
             r2=c*t2
             if ((h.le.c0).or.(n.gt.nc)) cycle
             t1=dl+s1
             s1=r1*t1-r2*s2;  s2=r2*t1+r1*s2;  dl=dl/h2
enddo
 if (y.ne.c0) then; if (k.ne.0) then; re=c112*s1; else; re=c112*r1; endif
              else; re=dexp(-x*x)
 endif
 if (k.eq.0) then; rm=c112*r2; else; rm=c112*s2; endif
return
end subroutine dcerf
```

Параметры вводимые tst_voigt из файла input

```
0.0000-00<-= начальное значение параметра (a0)
0.1000+01<-= шаг по параметру (ha)
0.4000+01<-= конечное значение параметра (am)
0.0000-00<-= начальное значение аргумента (x0)
0.1000+01<-= шаг по аргументу (hx)
0.4000+01<-= конечное значение аргумента (xk)
```

Результаты работы tst_voigt из файла result

```
A=0.0000D+00   A=0.0000D+00   A=0.0000D+00

      X      ФОЙГТ=Re(dcerf)         IM(dcerf)      2*Re/sqrt    2*Im/sqrt

 0.000E+00 0.1000000000000E+01 0.0000000000000E+00 0.564190E+00 0.000000E+00
 0.100E+01 0.3678794411714E+00 0.6071577058413E+00 0.207554E+00 0.342552E+00
 0.200E+01 0.1831563888873E-01 0.3400262170628E+00 0.103335E-01 0.191839E+00
 0.300E+01 0.1234098040867E-03 0.2011573170389E+00 0.696265E-04 0.113491E+00
 0.400E+01 0.1125351747193E-06 0.1459535899015E+00 0.634912E-07 0.823455E-01


A=0.1000D+01   A=0.1000D+01   A=0.1000D+01

      X      ФОЙГТ=Re(dcerf)         IM(dcerf)      2*Re/sqrt    2*Im/sqrt

 0.000E+00 0.4275835761553E+00 0.0000000000000E+00 0.241238E+00 0.000000E+00
 0.100E+01 0.3047442052568E+00 0.2082189382032E+00 0.171934E+00 0.117475E+00
 0.200E+01 0.1402395813664E+00 0.2222134401800E+00 0.791217E-01 0.125371E+00
 0.300E+01 0.6531777728903E-01 0.1739183154163E+00 0.368516E-01 0.981229E-01
 0.400E+01 0.3628145649000E-01 0.1358389510007E+00 0.204696E-01 0.766389E-01


A=0.2000D+01   A=0.2000D+01   A=0.2000D+01

      X      ФОЙГТ=Re(dcerf)         IM(dcerf)      2*Re/sqrt    2*Im/sqrt

 0.000E+00 0.2553956763105E+00 0.0000000000000E+00 0.144092E+00 0.000000E+00
 0.100E+01 0.2184926152749E+00 0.9299780939261E-01 0.123271E+00 0.524684E-01
 0.200E+01 0.1479527595120E+00 0.1311797170842E+00 0.834734E-01 0.740102E-01
 0.300E+01 0.9271076642644E-01 0.1283169622283E+00 0.523064E-01 0.723951E-01
 0.400E+01 0.5968692961044E-01 0.1132100561245E+00 0.336747E-01 0.638719E-01
```

```
A=0.3000D+01   A=0.3000D+01   A=0.3000D+01

      X      ФОЙГТ=Re(dcerf)         IM(dcerf)      2*Re/sqrt    2*Im/sqrt

 0.000E+00 0.1790011511813E+00 0.0000000000000E+00 0.100991E+00 0.000000E+00
 0.100E+01 0.1642611363930E+00 0.5019713513523E-01 0.926744E-01 0.283207E-01
 0.200E+01 0.1307574696698E+00 0.8111265047747E-01 0.737720E-01 0.457629E-01
 0.300E+01 0.9640250558304E-01 0.9123632600421E-01 0.543893E-01 0.514746E-01
 0.400E+01 0.6979096164965E-01 0.8934000024036E-01 0.393753E-01 0.504047E-01


A=0.4000D+01   A=0.4000D+01   A=0.4000D+01

      X      ФОЙГТ=Re(dcerf)         IM(dcerf)      2*Re/sqrt    2*Im/sqrt

 0.000E+00 0.1369994576267E+00 0.0000000000000E+00 0.772937E-01 0.000000E+00
 0.100E+01 0.1298881599307E+00 0.3077886081584E-01 0.732815E-01 0.173651E-01
 0.200E+01 0.1121394779017E+00 0.5348899385316E-01 0.632679E-01 0.301779E-01
 0.300E+01 0.9093390419486E-01 0.6559233052796E-01 0.513040E-01 0.370065E-01
 0.400E+01 0.7157043342646E-01 0.6937451861372E-01 0.403793E-01 0.391404E-01
```

Функция Фойгта встречается в задачах теории переноса излучения. В частности, в модельных задачах через неё выражается профиль коэффициента поглощения спектральной линии. В данном алгоритм использует расчёт функции Фойгта через её представление в виде цепной дроби.

8.9 Пятое домашнее задание (III-семестр).

1. Написать процедуру **E1_my(x,eps,res,umax,k)** расчёта функции

$$\mathbf{E_1}(\mathbf{x}) + \gamma + \mathbf{ln}(\mathbf{x})$$

на основе разложения функции $\mathbf{E_1}(\mathbf{x})$ в ряд Маклорена

$$\mathbf{E_1}(\mathbf{x}) \;=\; \int_{\mathbf{1}}^{\infty} \mathbf{e}^{-\mathbf{xy}} \frac{\mathbf{dy}}{\mathbf{y}} = -\gamma - \mathbf{ln}(\mathbf{x}) + \mathbf{x} - \frac{\mathbf{x^2}}{\mathbf{2 \cdot 2!}} + \frac{\mathbf{x^3}}{\mathbf{2 \cdot 3!}} - \cdots,$$
$$(\mathbf{0 < x < \infty});$$

Входные формальные аргументы:

- **x** — аргумент функции; **eps** — относительная погрешность ограничения;

Выходные формальные аргументы:

- **res** — результат суммирования ряда;
- **umax** — абсолютная величина максимального слагаемого разложения
- **k** — число слагаемых, формально обеспечившее достижения **eps**.

Тестирующая программа должна для равномерного дробления промежутка **[a,b]** на **n** подучастков выводить таблицу, каждая строка которой содержит **x**, **res**, **E1(x)**, **umax**, **k**.

2. Написать процедуру **erf_my(x,eps,res,umax,k)**расчёта функции **erf(x)** по её разложению в ряд Маклорена:

$$\mathbf{erf}(\mathbf{x}) = \frac{\mathbf{2}}{\pi} \sum_{\mathbf{n=0}}^{\infty} \frac{\mathbf{(-1)^n x^{2n+1}}}{\mathbf{n!(2n+1)}}.$$

- **x** — аргумент функции; **eps** — относительная погрешность ограничения;

Выходные формальные аргументы:

- **res** — искомый результат;
- **umax** — абсолютная величина максимального слагаемого разложения
- **n** — число слагаемых, формально обеспечившее достижение **eps**.

Тестирующая программа должна для равномерного дробления промежутка $\mathbf{x} \in [\mathbf{a}, \mathbf{b}]$ на **n** подучастков выводить таблицу, каждая строка которой содержит **x** (аргумент), **res**, **umax**, **n**, **aer** — абсолютная погрешность **res**, если за точный результат принять значение, получаемое встроенной функцией **erf(x)**, **rer** (соответствующая относительная погрешность).

9 Приложение IV. Суммирование асимптотического ряда

Важную роль в теории и приложениях играют так называемые **асимптотические ряды**. Для конкретной функции $f(x)$ асимптотический ряд часто представляется суммой вида

$$T(x) = \sum_{k=0}^{\infty} c_k \cdot x^{-k}.$$

В обычном смысле такие ряды расходятся, так как величины коэффициентов c_k растут быстрее чем x^k, так что конечного предела у полной суммы нет. Однако ценным свойством обладают частичные суммы

$$T_n(x) = \sum_{k=0}^{n} c_k \cdot x^{-k},$$

которые при достаточно большом x сколь-угодно близко по величине могут приблизиться к значению исходной функции $f(x)$, т.е.

$$|f(x) - T_n(x)| \leq \epsilon.$$

Другими словами, в случае асимптотического ряда для заданного n всегда можно подобрать такой x, что для любого наперед выбранного малого ϵ будет выполняться последнее неравенство. Так как $T_n(x)$ – конечная сумма, то программирование суммирования не встречает принципиальных трудностей.

Суммирование асимптотического ряда следует прекратить как только достигается

1) либо желаемая точность расчета;

2) либо увеличение модуля очередного слагаемого по сравнению с модулем предыдущего, что служит признаком начала формальной расходимости асимптотики.

Последнее означает, что достичь требуемой точности расчета $f(x)$ по данному асимптотическому разложению для заданного аргумента x невозможно (надо либо увеличить аргумент, либо понизить требуемую точность для требуемого). Если рабочий диапазон аргумента асимптотики, позволяющей с требуемой погрешностью ограничения выполнить расчёт, оценён заранее, то алгоритм суммирования может обойтись и без проверки признака начала расходимости.

План дальнейшего изложения материала:

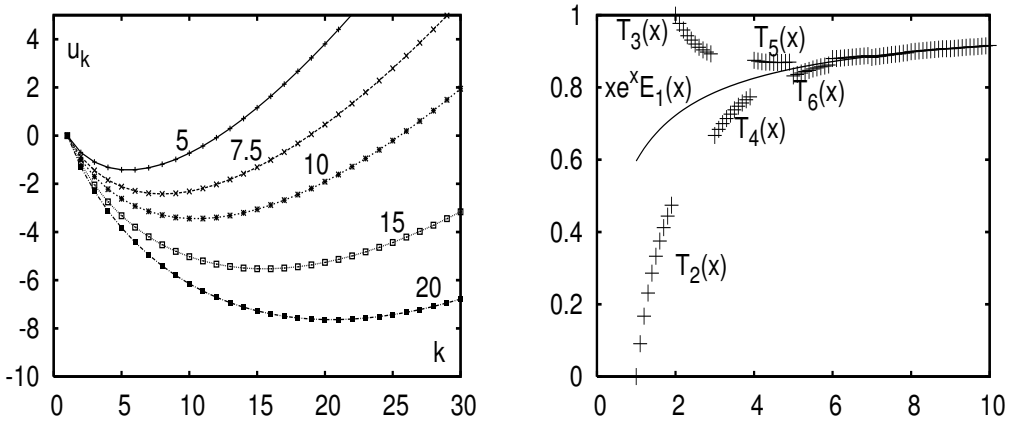
1. Примеры асимптотических разложений.
2. Получение асимптотического разложения E_1.
3. О формуле Эйлера–Маклорена.
4. Подпрограмма расчета дзета–функции Римана.
5. Результаты тестирования подпрограммы.

9.1 Примеры асимптотических разложений

1. Первая интегрально показательная функция: $\mathbf{E_1(x)} = \int_{\mathbf{x}}^{\infty} \frac{\mathbf{e^{-t}}}{\mathbf{t}}\mathbf{dt}$.

$$E_1(x) = \frac{e^{-x}}{x} \cdot \left[1 - \frac{1}{x} + \frac{2!}{x^2} - \frac{3!}{x^3} + \cdots\right], \qquad x \gg 1;$$

В частности, имеем $\qquad T_n(x) = xe^x E_1(x) = \sum_{k=0}^{n} (-1)^{-k} \frac{k!}{x^k}$.



На левом рисунке по оси абсцисс отложен $\mathbf{k}$ – номер очередного слагаемого асимптотики $\mathbf{xe^x E_1(x)}$, а по оси ординат – десятичный логарифм его абсолютной величины. Числа у кривых – значения аргумента $\mathbf{x}$. Видно, что с ростом номера очередное слагаемое сначала уменьшается, а затем возрастает (т.е. асимптотика выходит за пределы своего рабочего диапазона). Так при $\mathbf{x = 7.5}$ погрешность расчёта искомой суммы оценивается слагаемым порядка $\approx 10^{-2}$; однако при $\mathbf{x = 20}$ оценка погрешности уже $\approx 10^{-7}$.

Правый рисунок: жирная кривая – график функции $\mathbf{xe^x E_1(x)}$. *"Плюсы"* — значения $\mathbf{T_n(x)}$ при $\mathbf{n = 2, 3, 4, 5, 6}$ слагаемых. Уточнение прекращалось как только модуль очередного слагаемого превосходил модуль предыдущего.

2. Интеграл вероятностей: $\mathbf{erf(x)} = \frac{\mathbf{2}}{\sqrt{\pi}} \int_{\mathbf{0}}^{\mathbf{x}} \mathbf{e^{-t^2}}\mathbf{dt}$.

$$erf(x) = 1 - \frac{e^{-x^2}}{x\sqrt{\pi}} \cdot \left[1 - \frac{1}{2x^2} + \frac{1 \cdot 3}{2^2 x^4} - \frac{1 \cdot 3 \cdot 5}{2^3 x^6} + \cdots\right], \qquad x \gg 1$$

В частности, при $x > 3$ асимптотика

$$erf(x) = 1 - \frac{e^{-x^2}}{x\sqrt{\pi}} \cdot \left[1 - \frac{0.5}{x^2} + \frac{0.75}{x^4} - \frac{15}{8x^6}\right]$$

имеет погрешность $\epsilon \approx 6 \cdot 10^{-8}$. Заметим, что при $\mathbf{x{=}3}$ той же точности можно достичь, используя соответствующее разложение в ряд Маклорена.

9.2 Получение асимптотического разложения E_1

$$\mathbf{E_1(x)} = \int\limits_{\mathbf{1}}^{\infty} \mathbf{e^{-xt}} \frac{\mathbf{dt}}{\mathbf{t}} = \int\limits_{\mathbf{x}}^{\infty} \mathbf{e^{-z}} \frac{\mathbf{dz}}{\mathbf{z}}.$$

Интегрируя по частям последний интегал, зарабатываем в знаменателе степень $\mathbf{z}$:

$$\mathbf{E_1(x)} = \frac{\mathbf{e^{-x}}}{\mathbf{x}} - \int\limits_{\mathbf{x}}^{\infty} \mathbf{e^{-z}} \frac{\mathbf{dz}}{\mathbf{z^2}}.$$

Введем обозначение

$$\mathbf{E_k(x)} = \int\limits_{\mathbf{x}}^{\infty} \mathbf{e^{-z}} \frac{\mathbf{dz}}{\mathbf{z^k}}.$$

Тогда

$$\mathbf{E_1(x)} = \frac{\mathbf{e^{-x}}}{\mathbf{x}} - \mathbf{E_2(x)}.$$

Повторяя интегрирование по частям относительно $\mathbf{E_2(x)}$, получим

$$\mathbf{E_1(x)} = \frac{\mathbf{e^{-x}}}{\mathbf{x}} - \frac{\mathbf{e^{-x}}}{\mathbf{x^2}} + \mathbf{2E_3(x)}.$$

Аналогичный прием относительно $\mathbf{E_k(x)}$ дает рекурентное соотношение

$$\mathbf{E_k(x)} = \frac{\mathbf{e^{-x}}}{\mathbf{x}} - \mathbf{kE_{k+1}(x)}.$$

Используя его для представления $\mathbf{E_3}$ в предыдущем выражении для $\mathbf{E_1(x)}$ получаем

$$\begin{aligned}
E_1(x) &= \frac{e^{-x}}{x} - \frac{e^{-x}}{x^2} + 2\left[\frac{e^{-x}}{x^3} - 3E_4(x)\right] = \\
&= \frac{e^{-x}}{x} - \frac{e^{-x}}{x^2} + 2\cdot\frac{e^{-x}}{x^3} - 2\cdot 3\cdot E_4(x) = \\
&= \frac{e^{-x}}{x} - \frac{e^{-x}}{x^2} + 2\cdot\frac{e^{-x}}{x^3} - 2\cdot 3\cdot\left[\frac{e^{-x}}{x^4} - 4\cdot E_5(x)\right] = \\
&= \frac{e^{-x}}{x} - \frac{e^{-x}}{x^2} + 2\cdot\frac{e^{-x}}{x^3} - 2\cdot 3\cdot\frac{e^{-x}}{x^4} + 2\cdot 3\cdot 4\cdot E_5(x) = \\
&= \frac{e^{-x}}{x}\left[1 - \frac{1}{x} + \frac{2}{x^2} - \frac{3!}{x^3} + \cdots\right] + (-1)^k k! E_{k+1}(x).
\end{aligned}$$

Ясно, что

$$E_{k+1}(x) = \int\limits_{x}^{\infty} e^{-z}\frac{dz}{z^{k+1}} \le e^{-x}\int\frac{dz}{z^{k+1}} = k\frac{e^{-x}}{x^k}.$$

Если $\mathbf{x}$ выбрать настолько большим, что величина $\dfrac{\mathbf{k!}}{\mathbf{x^k}}$ окажется меньше ϵ, то сумма внеинтегральных слагаемых получит $\mathbf{E_1(x)}$ с абсолютной погрешностью ограничения не хуже ϵ. Формально можно записать, что

$$\mathbf{E_1(x)} = \frac{\mathbf{e^{-x}}}{\mathbf{x}} \sum_{\mathbf{k=0}}^{\infty} \mathbf{(-1)^k} \frac{\mathbf{k!}}{\mathbf{x^k}} \quad \text{или, что удобнее,} \quad \mathbf{xe^x E_1(x)} = \sum_{\mathbf{k=0}}^{\infty} \mathbf{(-1)^k} \frac{\mathbf{k!}}{\mathbf{x^k}}.$$

Формально последний ряд расходится, но для конкретного $\mathbf{n}$ всегда можно указать $\mathbf{x_0}$ такое, что для всех $\mathbf{x} \geq \mathbf{x_0}$ частичная сумма

$$\sum_{\mathbf{k=0}}^{\mathbf{n}} \mathbf{(-1)^k} \frac{\mathbf{k!}}{\mathbf{x^k}}$$

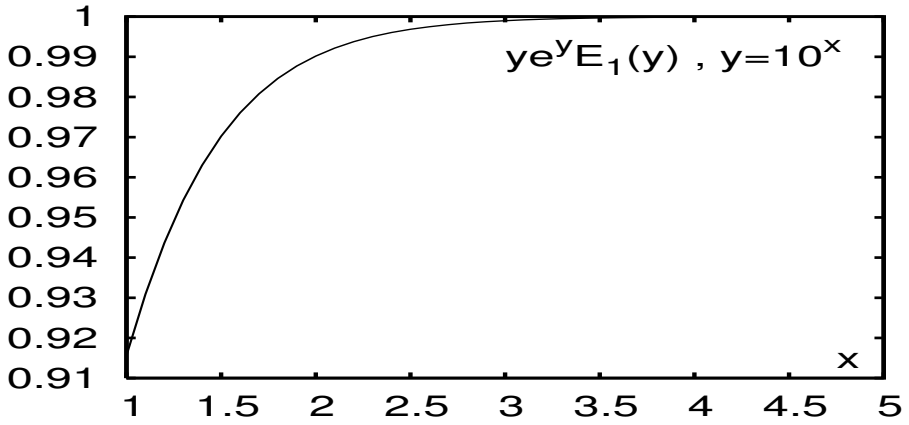
будет отличаться от

$$\int_{\mathbf{x}}^{\infty} \mathbf{e^{-z}} \frac{\mathbf{dz}}{\mathbf{z}}$$

меньше чем на заданное ϵ. Например, при $\mathbf{n = 8}$ ($n! = 40320$) имеем для $\mathbf{x = 10}$ абсолютную погрешность расчета не хуже $\mathbf{4 \cdot 10^{-4}}$, а для $\mathbf{x \geq 100}$ последняя улучшается не менее чем на двенадцать порядков.

Замечания:

1. Интегрирование по частям бывает удобно для получения асимптотик.
2. $\mathbf{x} \exp(\mathbf{x}) \mathbf{E_1(x)}$ – медленно возрастающая от 0,91 до 1 на полуоси $[10, \infty)$ функция:



Поэтому ее таблицы с постоянным шагом выгоднее для интерполяции, например, чем аналогичные таблицы непосредственно функции E_1.

9.3 О формуле Эйлера–Маклорена

Асимптотические разложения применимы и для суммирования медленно сходящихся рядов. В качестве примера рассмотрим суммирование ряда для ζ–функции Римана

$$\zeta(\mathbf{p}) = \sum_{\mathbf{k=1}}^{\infty} \frac{\mathbf{1}}{\mathbf{k^p}} \tag{1}$$

при $\mathbf{1 < p \le 2}$. Непосредственное суммирование этого ряда на ячейках одинарной точности в направлении уменьшения слагаемых, как убедились ранее (см. ZZZZ), в принципе не позволяет получить желаемый результат даже при $\mathbf{p = 2}$. Аналогично (см. приложение **IV**) суммирование в направлении увеличения слагаемых для достижения точности $\mathbf{10^{-7}}$ требовало до $\mathbf{10^7}$ слагаемых. Предлагаемый метод позволяет всего за 20 операций сложения достичь погрешности ограничения не хуже чем $\mathbf{10^{-17}}$.

Суть метода – ускорение сходимости остатка исходного ряда, проведенное на основе применения соответствующей асимптотической формулы Эйлера–Маклорена (см., например, Г.М. Фихтенгольц, (трехтомник) Курс дифференциального и интегрального исчисления, II том, стр. 549 1962). Следуя указанному первоисточнику, положим $\mathbf{p = 1 + s}$ и представим исходный ряд двумя суммами

$$\zeta(\mathbf{1+s}) = \mathbf{S(1+s)} + \mathbf{R(1+s)}, \tag{2}$$

где $\mathbf{S(1+s)}$ – сумма первых (скажем, $\mathbf{a-1}$) слагемых (пока $\mathbf{a}$ произвольно):

$$\mathbf{S(1+s)} = \sum_{\mathbf{k=1}}^{\mathbf{a-1}} \frac{\mathbf{1}}{\mathbf{k^{1+s}}}, \tag{3}$$

а $\mathbf{R(1+s)}$ – сумма всех остальных, начиная с $\mathbf{a}$-го, то есть остаток ряда:

$$\mathbf{R(1+s)} = \sum_{\mathbf{k=a}}^{\infty} \frac{\mathbf{1}}{\mathbf{k^{1+s}}}. \tag{4}$$

$\mathbf{S(1+s)}$ вычислим непосредственно по (3), а $\mathbf{R(1+s)}$ – с помощью соответствующей (4) формулы Эйлера–Маклорена. Фактически будет вычисляться даже не $\mathbf{R(1+s)}$, а функция

$$\mathbf{s \cdot a^s \cdot R(1+s)} \equiv \left[\zeta(\mathbf{1+s}) - \sum_{\mathbf{i=1}}^{\mathbf{a-1}} \frac{\mathbf{1}}{\mathbf{i^{1+s}}}\right] \cdot \mathbf{s \cdot a^s} =$$

$$= \mathbf{1} + \mathbf{s} \cdot \left[\frac{\mathbf{1}}{\mathbf{2a}} + \sum_{\mathbf{k=1}}^{\infty} (\mathbf{-1})^{\mathbf{k}} \frac{\mathbf{c_k}}{\mathbf{a^{2k}}}\right], \tag{5}$$

где

$$\mathbf{c_k} = \frac{\mathbf{B_k}}{\mathbf{(2k)!}} \prod_{\mathbf{i=0}}^{\mathbf{2k-2}} (\mathbf{1+s+i}), \tag{6}$$

а $\mathbf{B_k}$ – числа Бернулли.

(5) и есть для остатка ряда (4) формула Эйлера–Маклорена.

Заметим, что (5) расходящийся асимптотический ряд. Так что суммирование должно быть прекращено при выполнении хотя бы одного из трех условий:

1. Требуемая погрешность метода достигнута ($\mathbf{a}$ и ϵ заданы грамотно), т.е.

$$\frac{\mathbf{sC_k}}{\mathbf{a^{2k}}} < \epsilon, \tag{7}$$

2. Требуемая погрешность метода не достигнута, но дальнейшее уточнение суммы бессмысленно, так как значение очередного слагаемого на ячейках используемой разрядности не влияет на значение накопленной суммы, т.е.

$$\mathbf{1} + \frac{\mathbf{sC_k}}{\mathbf{a^{2k}}} = \mathbf{1}. \tag{8}$$

   Такая ситуация возможна при грамотном выборе $\mathbf{a}$, но чрезмерно завышенной (по сравнению с используемой разрядностью) погрешностью метода. Так что, хотя последнее слагаемое, учтенное в сумме, и меньше по модулю требуемой погрешности, но вклад его в величину результата практически нулевой.

3. Требуемая погрешность метода не достигается из-за включения расходимости:

$$\frac{\mathbf{C_{k+1}}}{\mathbf{a^{2k+2}}} > \frac{\mathbf{C_k}}{\mathbf{a^{2k}}}, \tag{9}$$

   т.е. для достижения требуемой погрешности метода величина $\mathbf{a}$ выбрана недостаточно большой. Увеличим $\mathbf{a}$ и тогда возможно добьемся (7).

Суть используемого метода ускорения сходимости имеет наглядное геометрическое толкование. Сумма из $\mathbf{n}$ слагаемых ряда равна площади ступенчатой фигуры. Если найти монотонно убывающую функцию $\mathbf{f(x)}$, значение которой при совпадении аргумента $\mathbf{x}$ с номером слагаемого $(\mathbf{i})$ равно величине слагаемого, то величина $\int_{\mathbf{a}}^{\mathbf{b}} \mathbf{f(x)dx}$ возможно не будет слишком грубым приближением к величине суммы. Если же при этом интеграл $\int_{\mathbf{a}}^{\mathbf{b}} \mathbf{f(x)dx}$ берется аналитически, то его величина – основной вклад в сумму ряда – оценивается по явной формуле без суммирования огромного количества слагаемых, а учет поправок к основному вкладу дается формулой Эйлера–Маклорена. Ее общий вид

$$\sum_{\mathbf{i=0}}^{\mathbf{m-1}} \mathbf{f(a+i*h)} \equiv \sum_{\mathbf{a}}^{\mathbf{b}} \mathbf{f(x)} = \frac{\mathbf{1}}{\mathbf{h}} \int_{\mathbf{a}}^{\mathbf{b}} \mathbf{f(x)dx} + \mathbf{A_1}[\mathbf{f(b)} - \mathbf{f(a)}] + \mathbf{A_2 h}[\mathbf{f'(b)} - \mathbf{f'(a)}] + \cdots + \mathbf{A_{m-1} h^{m-1}}[\mathbf{f^{(m-2)}(b)} - \mathbf{f^{(m-2)}(a)}] + \ldots \ ,$$

где

$$\mathbf{A_1} = -\frac{\mathbf{1}}{\mathbf{2}}; \quad \mathbf{A_{2k-1}} = \mathbf{0(k>1)}; \quad \mathbf{A_{2k}} = (\mathbf{-1})^{\mathbf{k-1}} \frac{\mathbf{|B_{2k}|}}{\mathbf{(2k)!}}.$$

9.3.1 Чуть-чуть о числах Бернулли

Для расчёта по асимптотической формуле Эйлера-Маклорена (5) требуются числа Бернулли $\mathbf{B_{2k}}$ (с чётными индексами). $\mathbf{B_{2k-1}} = \mathbf{0}$, кроме $B_1 = -0.5$. $\mathbf{B_{2k}}$ можно вычислить, например, по известной рекуррентой формуле:

$$B_n = \frac{-1}{n+1} \sum_{k=1}^{n} C_{n+1}^{k+1} B_{n-k} \quad , n = 1, 2, \cdots$$

Правда, непосредственный расчёт $\mathbf{B_{2k-1}}$ именно по ней из-за конечности разрядной сетки приведёт к значениям $\mathbf{B_{2k-1}}$ формально отличным от нуля. Поместим описание рабочего вектора **b** и инициализацию его первых двадцати элементов двадцатью первыми числами Бернулли с чётными номерами в модуль **bern.f**

```
 module bern; implicit none; real(8), parameter :: b(17)=(/
1  0.16666666666666666D+00, 0.33333333333333333D-01,
3  0.2380952380952380 8D-01, 0.33333333333333333D-01,
5  0.7575757575757576 0D-01, 0.2531135531135531 0D+00,
7  0.1166666666666666 7D+01, 0.7092156862745097 7D+01,
9  0.5497117794486215 6D+02, 0.5291242424242424 2D+03,
1  0.6192123188405797 0D+04, 0.8658025311355311 7D+05,
3  0.1425517166666666 7D+07, 0.2729823106781609 4D+08,
5  0.6015808739006424 0D+09, 0.1511631576709215 7D+11,
7  0.4296446430611666 9D+12 /)
 contains
 function contr(np) result(r);   ! вычисляет zeta-функцию Римана
 implicit none                   ! для чётного аргумента (np) по
 real(8) pi, pi2, r              ! формуле:     2n-1    2n
 integer np, n, i                !             2    * pi  * B(n)
 pi =4d0*datan(1d0); n=np/2;     ! dzeta(2n) = ----------------- ,
 pi2=pi*pi; r=pi2/6; i=1         !                   ( 2n ) !
 do                              ! где n=np/2.
   if (i.ge.n) exit
   r=r*4*pi2*b(i+1)/b(i)/(2*i+1)/(2*i+2)
   i=i+1
 enddo
 end function contr
 end module bern
```

с тем, чтобы далее их использовать не только при работе **dzeta1**, но и при вызове функции **contr(n)**, вычисляющей $\zeta(\mathbf{n})$ при чётном **n** по формуле

$$\zeta(2n) = \frac{2^{2n-1}\pi^{2n} \cdot B(n)}{(2n)!},$$

для контроля результатов **dzeta1**.

9.4 Подпрограмма расчета дзета–функции Римана

Здесь приводится исходный текст подпрограммы **dzeta1**, вычисляющей ζ-функцию Римана по соответствующей формуле Эйлера-Маклорена. В подпрограмме специально выделены все три упомянутые выше возможности завершения процесса суммирования. Если пользователь гарантирует правильность задания значений входных аргументов процедуры, то исходный текст подпрограммы можно существенно сократить. Иногда (по невнимательности) числовые значения аргумента ζ-функции или погрешности метода задаются вне приемлемого рабочего диапазона. Возникающие при этом эффекты легко выявить, анализируя результаты работы **dzeta1**.

```
!================================================================
! Подпрграмма dzeta1(s,eps,ka,res,erfact,number,ier) вычисляет для
! заданной величины параметра (s) значение дзета-функции Римана от
! аргумента (1+s) на основе применения формулы Эйлера-Маклорена.
! Входные аргументы:
!  s  - поправка величины аргумента (p=1+s);
! eps - требуемая абс.погр. ограничения поправочной суммы;
! ka - номер слагаемого, учитываемого первым в остатке ряда R(1+s),
!      или, что то же --- нижний предел, явно берущейся, интегральной
!      составляющей формулы Эйлера-Маклорена.
! Выходные параметры:
! 1) res - вектор из шести элементов:
!    res(1) = \sum_{k=1}^{=infty} (-1)**k c_k/a**(2k)
!    res(2) = 1+s*(1/2a + res(1))=s*a^s * R(1+s))
!    res(3) = s * ka^s .....: весовой множитель
!    res(4) = R(1+s) при s>0: величина остатка или huge(1.0_8) при s=0.
!    res(5) = S(1+s) .......: сумма первых ka-1 слагаемых.
!    res(6) = S(1+s)+R(1+s).: dzeta(1+s) или huge(1d0) при s=0;
! 2) erfact - величина последнего слагаемого учтенного в асимптотике;
! 3) num    - полное количество слагаемых;
! 4) ier - код причины завершения суммирования: требуемое EPS
!     0  - достигнуто (ура!)
!     1  - НЕ ДОСТИГНУТО, т.к. конфликтует с разрядностью типа REAL(8);
!     2  - НЕ ДОСТИГНУТО при заданном KA из-за начала расходимости.
! DZETA1 использует модуль BERN в качестве импортёра чисел Бернулли.
!................................................................
      subroutine dzeta1(s,eps,ka,res,erfact,num,ier); use bern
      implicit none
      logical(1) okey,okeps,okcet,okpac
      real(8) res(6), s, eps, dk2, predrk, absrk, erfact, di
      real(8) c1, s1, ss, a1, a2, pk, rk, rs
      integer ka, num, ka1, ier, k, i
      c1=1.0d0; ier=0; ka1=ka-1; s1=c1+s;
      ss=0d0; do i=1,ka1              ! Суммирование первых
                di=i                  !      ka-1 слагаемых:
                ss=ss+c1/di**s1 .
      enddo
```

```
      a1=c1/ka; a2=a1*a1; pk=s1*0.5d0*a2; rk=pk*b(1);
      rs=0; absrk=rk; k=1
      do; predrk=absrk; rs=rs +rk; dk2 =k+k
        pk=-pk*a2*(c1+s/dk2)*(c1+s/(c1+dk2))/(c1+c1/k)
        k=k+1
        rk=b(k)*pk; absrk=dabs(rk)     ! ok???: критерии прекращения уточнения
        okeps=(absrk.lt.eps)           ! TRUE : модуль тек.слагаемого < eps .
        okcet=(rs.eq.rs+absrk)         ! TRUE : тек.слагаемое не меняет суммы.
        okpac=(absrk.ge.predrk)        ! TRUE : |тек.слагаемое| >= |предыдущего|
        okey=(okeps.or.okcet.or.okpac) ! TRUE : прекращаем уточнять сумму
        if (okey) exit                 !      : и выходим из цикла
      enddo
      if(okcet) ier=1; if(okpac) ier=2 ! Уточнение кода причины завершения.
      res(1)=rs
      res(2)=c1+s*(0.5d0*a1+rs)        ! s*(ka**s)*R(1+s) по асимптотике.
      res(3)=s*ka**s                   ! s*(ka**s)
      if(s.eq.0d0)then; res(4)=huge(1d0)
                  else; res(4)=res(2) / res(3)  ! расчёт остатка: R(1+s)
      endif
      res(5)=ss                        ! сумма первых ka-1 слагаемых: S(1+s)
      res(6)=res(4)+res(5)             ! Значение dzeta-функции Римана
      num=k+ka1                        ! полное число слагаемых
      erfact=predrk
      end
```

Tdzeta1 — программа тестирования dzeta1.

```
!=====================================================
! Программа предназначена для тестирования подпрограммы
! dzeta1, вычисляющей дзета-функцию Римана:
! Вводимые данные (из файла riman.inp):
!   1) eps (по формату d10.3) - требуемая абс.погр.метода;
!   2)  p0 (-- " --) - начальное значение аргумента (p);
!   3)  pk (-- " --) - конечное  значение аргумента (p);
!   4)  hp (-- " --) - шаг по аргументу (p);
!   5)  ka (   i2  ) - номер первого слагаемого остатка;
! Выводимые данные  (в файл riman.res):
!  1) контрольная печать вводимых данных;
!  2) таблица результата (каждая ее строка содержит:
!     номер аргумента; величину аргумента.............   (   p  );
!     значение поправкм Эйлера-Маклорена (неявная сумма) ( re(1)
!     искомое значение дзета-функции Римана...........   ( re(5));
!     число слагаемых, востребованных суммированием...   (  kce );
!     последнее учтенное слагаемое....................   (erfact);
!    код причины завершения работы подпрограммыююю...   (  ier );
!    абс.погр. найденного re(5) для четного аргумента.   (  aer ).
      program tdzeta1; use bern; implicit none
      real(8) res(6)                    ! вектор текущего результата;
```

```
      integer ninp / 5 /, nres / 6 /    ! номера файлов ввода/вывода;
      integer i, ka, kp, np, kce, ier
      real(8) eps, p0, pk, hp, p, s, erfact, aer, rcontr
      open (unit=ninp,file='riman.inp')
      open (unit=nres,file='riman.res', status='replace')
      read (ninp, '(d10.3)') eps, p0, pk, hp ! Ввод исходных
      read (ninp, '(  i2 )') ka              !            данных.
      write(nres,1000) eps, p0, pk, hp       ! Контрольный вывод
      write(nres,1001) ka                    !        введённого.
      write(nres,1010)                       ! Вывод заголовка таблицы.
      kp=idint((pk-p0)/hp+0.5) + 1           ! Число строк в таблице.
      do i=1,kp; p=p0+hp*(i-1)               ! Аргумент дзета-функции
        s=p-1                                ! Аргумент процедуры dzeta1.
        call dzeta1(s,eps,ka,res,erfact,kce,ier)
        np=idint(p); if (dabs(p-np/2*2).lt.1d-16)  ! Если p чётно, то расчёт
     >                 then; rcontr=contr(np)      ! контроля и
                           aer=abs(rcontr-res(6))!        абс.погрешности,
                      else; rcontr=-1d0; aer=-1d0 ! иначе ответ фиктивен.
                    endif                                  ! Вывод
        write(nres,1011) i,p,res(1),res(6),kce,erfact,ier,aer ! результата.
      enddo
      close(nres)
 1000 format(1x,'eps=',d10.3/                              ! Форматы
     >       1x,' p0=',d15.7/1x,' pk=',d15.7/1x,' hp=',d15.7) ! вывода
 1001 format(1x,' ka=',i4)
 1010 format(1x,'  N',4x,'p',6x,'res(1)',7x,'dzeta(p)=res(6)',5x,
     >          'kce',2x,'erfact',1x,'ier',3x,'aer')
 1011 format(1x,i3,f7.3,e11.3,d25.17,i4,d8.1,1x,i2,1x,d10.3)
      end
```

Содержимое файла Riman.inp, вводимое Tdzeta1.

```
  1.000-16 eps: абсолютная погрешность суммирования асимптотики.
  1.100+00  p0: начальное значение аргумента дзета-функции Римана.
  2.000+00  pk: конечное  --"--"--"--"--"--"--"--"--"--"--"--"--
  0.100+00  hp: шаг по аргументу
10 ka: ka-1 число элементов исходного ряда суммируемых непосредственно.
```

Обратите внимание на удобство использования оператора **format** для организации ввода данных. Он позволяет не только ввести параметр, размещённый в определённом поле строки, но и *не обращать внимания* при этом на содержимое остальных её позиций, так что в них можно поместить комментарий к вводимому параметру. Удобство состоит в том, что когда через полгода или год возникнет необходимость пропуска этой программы с другими данными, то Вам не придётся *влезать* в текст программы вообще — достаточно посмотреть на содержимое файла с вводимыми данными — и, благодаря комментариям, сразу станет ясно, что после чего вводится, не говоря уже о полях записи и формате ввода новых числовых данных.

Результаты тестирования dzeta1: eps=1d-11, ka=10, чётный аргумент.

```
eps= 0.100D-10
 p0=  0.2000000D+01
 pk=  0.8000000D+01
 hp=  0.2000000D+01
 ka=  10
  N    p      res(1)       dzeta(p)=res(6)     kce  erfact ier   aer
  1  2.000  0.166E-02  0.16449340668474932D+01  14 0.3D-09  0  0.733D-12
  2  4.000  0.332E-02  0.10823232337111455D+01  15 0.2D-09  0  0.755D-14
  3  6.000  0.495E-02  0.10173430619844490D+01  16 0.9D-10  0  0.222D-15
  4  8.000  0.657E-02  0.10040773561979444D+01  17 0.6D-10  0  0.444D-15
```

1. **ier=0**, т.е. требуемая точность достигнута.

2. При **p=2** расчёт res(1) использовал **14-9=5** слагаемых.

3. Последнее учтённое асимптотикой слагаемое **erfact** $\sim \mathbf{10^{-10}}$, т.е. следующее слагаемое было бы согласно (5) $\sim 10^{-12}$, что меньше требуемого $\mathbf{eps = 10^{-11}}$.

4. Наконец, сравнение с результатом функции **contr**, принятым за точный, даёт величину абсолютной погрешности ζ-функции Римана равную $7 \cdot 10^{-13}$.

5. Итак, просуммировав всего 14 слагаемых по методу Эйлера-Маклорена нашли с высочайшей точностью сумму, которую при суммировании *в лоб* десяти миллионов слагаемых нашли бы с точностью только в 4 верные значащие цифры.

6. В справочнике [21] можно найти, что $\zeta(\mathbf{2}) = \mathbf{1.64493406684822643647}$. Введя $eps = 10^{-16}$, легко убедиться, что **dzeta1** за 18 сложений прекрасно справляется со своей работой в пределах разрядной сетки типа **real(8)**:

   ```
   eps= 0.100D-15
    p0=  0.2000000D+01
    pk=  0.8000000D+01
    hp=  0.2000000D+01
    ka=  10
     N    p      res(1)       dzeta(p)=res(6)     kce  erfact ier   aer
     1  2.000  0.166E-02  0.16449340668482264D+01  18 0.7D-15  0  0.000D+00
     2  4.000  0.332E-02  0.10823232337111384D+01  20 0.4D-15  0  0.444D-15
     3  6.000  0.495E-02  0.10173430619844490D+01  22 0.4D-15  0  0.222D-15
     4  8.000  0.657E-02  0.10040773561979444D+01  25 0.2D-15  0  0.444D-15
   ```

7. При $\mathbf{eps = 10^{-11}}$ фактическая погрешность расчёта ζ-функции имеет гораздо более высокий порядок малости: $\mathbf{aer = 7 \cdot 10^{-13}}$. Дело в том, что **dzeta1** под **eps** понимает ограничение очередного слагаемого из под знака $\sum$ формулы (5), а вовсе не погрешность расчёта ζ-функции. Иногда в качестве результата требуется высокоточное значение только лишь неявной асимптотической компоненты формулы (5), и **dzeta1** позволяет его получить.

Посмотрим теперь на результаты **dzeta1**, когда требуемая абсолютная пргрешность метода значительно превышает возможности разрядной сетки ЭВМ.

Результаты тестирования dzeta1: eps=1d-31, ka=10; чётный аргумент.

```
 eps= 0.100D-30
  p0=  0.2000000D+01
  pk=  0.8000000D+01
  hp=  0.2000000D+01
  ka=  10
   N    p      res(1)       dzeta(p)=res(6)     kce  erfact ier   aer
   1  2.000  0.166E-02  0.16449340668482264D+01  21 0.6D-18  1  0.000D+00
   2  4.000  0.332E-02  0.10823232337111384D+01  24 0.4D-18  1  0.444D-15
   3  6.000  0.495E-02  0.10173430619844490D+01  27 0.6D-18  1  0.222D-15
   4  8.000  0.657E-02  0.10040773561979444D+01  27 0.2D-16  1  0.444D-15
```

1. Если разрядность позволяет вести расчёт лишь с 16-ю значащими цифрами (а результат расчёта порядка единицы), то бессмысленно требовать от алгоритма достижения абсолютной погрешности $\mathbf{eps = 10^{-31}}$, так как погрешность округления искомой суммы будет не менее чем $\mathbf{10^{-16}}$, т.е. на 15 порядков больше требуемой точности расчёта.

2. Естественно, **kce** (полное количество слагаемых) возрастёт, **erfact** (величина последнего слагаемого, учтённого в асимптотике) уменьшится. Однако **ier** (код причины завершения работы **dzeta1**) окажется равным **единице**, поскольку было обнаружено, что от добавления очередного корректного слагаемого накапливаемая сумма не изменяется. В итоге, результат практически не будет отличаться от случая $\mathbf{eps = 10^{-16}}$.

Теперь рассмотрим случай, когда при непосредственном суммировании исходного ряда учтено недостаточное число слагаемых (к примеру, не 10, только 5) при прежнем требовании по точности $eps = 10^{-11}$.

Результаты тестирования dzeta1: eps=1d-11, ka=5; чётный аргумент.

```
 eps= 0.100D-10
  p0=  0.2000000D+01
  pk=  0.8000000D+01
  hp=  0.2000000D+01
  ka=   5
   N    p      res(1)       dzeta(p)=res(6)     kce  erfact ier   aer
   1  2.000  0.661E-02  0.16449340668489960D+01  14 0.1D-10  0  0.770D-12
   2  4.000  0.131E-01  0.10823232337107254D+01  19 0.1D-09  2  0.413D-12
   3  6.000  0.193E-01  0.10173430619852644D+01  18 0.5D-08  2  0.816D-12
   4  8.000  0.252E-01  0.10040773561971781D+01  17 0.1D-06  2  0.766D-12
```

1. Видим, что когда **ka=5**, то при **p=2** за 14 сложений удаётся достичь требуемой абсолютной погрешности результата: **ier** (код причины завершения работы **dzeta1**) равен **нулю**.

2. Однако, при **p=4** получаем **ier=2**, т.е., асимптотический ряд, не достигнув требуемой точности в процессе суммирования, начал расходиться (очередное слагаемое оказалось больше по модулю предыдущего). При этом значение самой ζ-функции находится с точностью 10^{-12} (на два порядка более высокой нежели точность асимптотики).

3. Причина: найденное с точностью $\mathbf{eps} = \mathbf{10^{-10}}$ (худшей нежели 10^{-11}) значение $\mathbf{res(1)} \approx \mathbf{0.013}$ при расчёте res(2) складывается с $1/(2a) = 1/(2{\cdot}5) = 0.1$ (числом на порядок большим res(1)). Так что при сложении десять верных старших цифр res(1) сдвигаются на один десятичный разряд вправо. Тем самым, десятая верная цифра res(1) корректно уточняет одиннадцатую верную цифру суммы.

4. Далее, при расчёте $RES(2) = 1 + s((1/2a) + res(1))$ происходит сдвиг произведения $s((1/2a) + res(1))$ ещё на один десятичный разряд вправо. Так что в корректировке одиннадцати верных старших цифр единицы принимают участие лишь 9 старших цифр res(1).

5. При делении res(2) (числа равного ≈ 0.3) на 3*5**3=375 (см. исходный текст dzeta1) получаем ~ 0.001. Таким образом, для корректировки 12-13 десятичных цифр суммы (res(5)$\approx$1.6), накопленной непосредственным суммированием исходного ряда, достаточно лишь семи-восьми цифр старших цифр значения RES(1).

9.5 Ещё один пример использования асимптотик

Асимптотические разложения, корректно работающие в допустимой области применения, могут быть использованы и в качестве полезного вспомогательного средства, когда аргумент оказывается вне области применимости асимптотики. Так, если существует простое функциональное соотношение, связывающее два значения функции (для большего и меньшего аргументов), то последовательно применяя его можно попытаться довести аргумента до значения, приемлемого для асимптотики, накопив при этом вклад соответствующих элементов функционального соотношения. В итоге, добавив этот вклад к результату асимптотики (уже корректно вычисленного для нового аргумента) есть шанс получить высокоточное значение функции для исходного аргумента. Например, расчёт значения дигамма-функции $\psi(\mathbf{x}) = \dfrac{\mathbf{\Gamma}'(\mathbf{x})}{\mathbf{\Gamma}(\mathbf{x})}$ при $\mathbf{x} \gg \mathbf{1}$ можно осуществить с высокой точностью по асимптотическому разложению

$$\psi(\mathbf{1}+\mathbf{x}) = \ln \mathbf{x} + \frac{\mathbf{1}}{\mathbf{2x}} - \frac{\mathbf{1}}{\mathbf{12x^2}} + \frac{\mathbf{1}}{\mathbf{120x^4}} - \frac{\mathbf{1}}{\mathbf{252x^6}} + \epsilon$$

В то же время для $\psi(x)$ известны соотношения (см., например, [21]):

$$\psi(\mathbf{x}+\mathbf{1}) = \psi(\mathbf{x}) + \frac{\mathbf{1}}{\mathbf{x}}$$

и

$$\psi(\mathbf{n}+\mathbf{x}) = \frac{\mathbf{1}}{\mathbf{n-1+x}} + \frac{\mathbf{1}}{\mathbf{n-2+x}} + \cdots + \frac{\mathbf{1}}{\mathbf{2+x}} + \frac{\mathbf{1}}{\mathbf{1+x}} + \frac{\mathbf{1}}{\mathbf{x}} + \psi(\mathbf{x})$$

При $|\mathbf{x}|<\mathbf{1}$ можно, подобрав $\mathbf{n}$ такое, что $\mathbf{x}+\mathbf{n} \geq \mathbf{a} \gg \mathbf{1}$):

1) вычислить по асимптотике значение $\psi(\mathbf{x}+\mathbf{n})$ с точностью не хуже $\epsilon \sim \dfrac{\mathbf{1}}{\mathbf{240a^8}}$
2) накопить сумму $\mathbf{s} = \dfrac{\mathbf{1}}{\mathbf{n-1+x}} + \dfrac{\mathbf{1}}{\mathbf{n-2+x}} + \cdots + \dfrac{\mathbf{1}}{\mathbf{2+x}} + \dfrac{\mathbf{1}}{\mathbf{1+x}} + \dfrac{\mathbf{1}}{\mathbf{x}}$
3) найти разность $\psi(\mathbf{n}+\mathbf{x}) - \mathbf{s}$, т.е. значение $\psi(\mathbf{x})$, когда $\mathbf{x}$ невелико.

Ниже приведены исходные тексты:

- модуля **my_prec** перехода на любую допустимую разновидность типа **real**;
- процедуры **inspect0(xp)**, которая вычисляет контрольные значения $\psi(\mathbf{x})$ для $x = -2.5(1)0.5(0.5)3.0$);
- функции **psifun(x,a)**, работающей по указанному алгоритму;
- тестирующей главной программы;
- типичное содержимое вводимого файла **input**;
- таблицы результатов работы **psifun** при **a=10** и разных **mp**;
- **make**-файл, обеспечивающий пропуск программы.

```
module my_prec
implicit none
integer,  parameter :: mp=16   ! возможные значения mp: 4, 8, 10, 16
character(6), parameter :: frmt(4)=(/' e15.7','e25.16','e27.18','e45.36'/)
contains
function frm()        ! Функция frm() возвращает в вызывающую программу
character(6) frm      ! значение типа character(6), которое представляет
integer k             ! собой формат вывода вещественного значения
k=mp/4                ! типа real(mp). Формат выбирается из элемента
if (mp==10) k=k+1     ! вектора frmt под номером mp/4, если mp=4, 8, 16
frm=frmt(k)           ! и из элемента frmt(3), если mp=10.
end function frm
end module my_prec
```

Модуль задаёт

- значение константы **mp**;
- массив **frmt(4)**, каждый из элементов которого есть шестисимвольная строка с форматом вывода значения $\psi(x)$, соответствущего **mp** параметру разновидности типа **real**
- функцию генерации формата текущей строки результата

```
subroutine inspect0(xp); use my_prec
implicit none
real(mp), parameter :: cee= 0.57721566490153286060651209008240243l_mp
real(mp):: xp(9)
xp=reshape((/46.0_mp/15.0_mp -cee - log(4.0_mp),   & ! -2.5_mp
&             8.0_mp/3.0_mp - cee - log(4.0_mp),   & ! -1.5_mp
&             2.0_mp - cee - log(4.0_mp),          & ! -0.5_mp
&             -cee - log(4.0_mp),                  & !  0.5_mp
&             -cee,                                & !  1.0_mp
&             2.0_mp - cee - log(4.0_mp),          & !  1.5_mp
&             1.0_mp - cee,                        & !  2.0_mp
&             8.0_mp/3.0_mp - cee - log(4.0_mp),   & !  2.5_mp
&             1.5_mp - cee/),(/9/),order=(/1/))      !  3.0_mp
end subroutine inspect0
```

Подпрограмма **inspect0(xp)** заполняет девятиэлементный массив **xp** значениями $psi(x)$ для $x = -2.5, -1.5, -0.5, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0$.

- подпрограмма использует постоянную Эйлера (константа **cee**);
- формулы для значений элементов взяты из [21];

```
!=====================================================================
! Функция psifun (x,a) вычисляет значение дигамма-функции psi(x)
! по схеме алгоритма 147 из книги:
! Библиотека технической кибернетики. М.И. Агеев, В.П. Алик, Ю.И.
! Марков. Библиотека алгоритмов 1016-1506. Справочное пособие.
! Выпуск 3. Москва "Советское радио" 1978; Алгоритм 147, стр. 74-76.
! Алгоритм использует асимптотику
!                   1      1          1          1
! psi(1+x)=ln(x)+ --- - ------  + -------  - -------  + eps  (1)
!                 2x    12 x^2    120 x^4    252 x^6
!
! и рекуррентное соотношение          psi(x+1)=psi(x) + 1/x .   (2)
!
! Входные параметры: x --- аргумент;
!  A    --- при x >= A точность работы (1) не хуже eps=1/240/x^8.
! Если x<A, то (2) используется для наращивания аргумента x до
! значения большего A с накапливанием суммы соответствующих слагаемых,
! так что для нового аргумента x расчёт psi(1+x) возможен по (1) с
! точностью не хуже eps=1/240/a^8, после чего значение асимптотики
! добавляется к накопленной ранее сумме.
!
! psifun(x,a) не нацелена на обработку случаев, когда аргумент
! x есть 0, отрицательное целое или находится в ближайшей их
! окрестности. Программа пользователя должна сама отреагировать
! на ситуации разрыва функции или бесконечности её значений.
!.....................................................................
function psifun(xx,a) result(w)
use my_prec
implicit none
real(mp), parameter :: c0=0.0_mp, c05=0.5_mp, c1=1.0_mp,              &
&       c12=1e0_mp/12e0_mp, c120=1e0_mp/120e0_mp, c252=1e0_mp/252e0_mp
real(mp), intent( in) :: xx, a
real(mp) x, y, w
x=xx
w=0.0_mp
do while (x<a)
  x=x+1
  w=w-c1/x
enddo
y=c1/x/x
w=w+log(x)+c05/x-((y*c252-c120)*y+c12)*y
w=w-c1/xx
end function psifun
```

Заметим, что если при $x \gg 1$ нужна разность $\psi(\mathbf{x}) - \mathbf{ln(x)}$, то не следует сначала вычислять $\psi(x)$, а затем вычитать из её значения $\mathbf{ln(x)}$, что может привести к большой потере точности на конечнозначной арифметике с плавающей запятой. Несравнимо выгоднее просто запрограммировать расчёт суммы слагаемых без учёта $\mathbf{ln(x)}$.

```
program test_psi
use my_prec
implicit none
interface
  function psifun(xx,a) result(w)
    use my_prec
    real(mp), intent(in)  :: xx, a
    real(mp) w
  end function psifun
  subroutine inspect0(xp); use my_prec
             real(mp), intent(out) :: xp(9)
  end subroutine inspect0
end interface
character(30) sf
real(mp) a, xp(9), r, x, aer, rer
integer i
write(*,*) ' mp=',mp
read (*,'(f7.2)') a;      write(*,*) '  a=', a
call inspect0(xp)                          ! Расчёт контрольных значений.
write(*,'(5x,"X",4x,$)')                                ! Организация
select case(int(mp/4.0+0.7))                            ! вывода
  case(1); write(*,'( 7x,"PSI(X)", 8x,"aer",7x,"rer")') ! заголовка
  case(2); write(*,'(12x,"PSI(X)",12x,"aer",7x,"rer")') ! таблицы
  case(3); write(*,'(13x,"PSI(X)",13x,"aer",7x,"rer")') ! результата
  case(4); write(*,'(22x,"PSI(X)",22x,"aer",7x,"rer")')
end select
sf='(f7.2,'//frm()//',2e10.2)'             ! Формат вывода строки таблицы.
do i=1,9
  if (i<=3) then; x=-3.5_mp+i
            else; x=-1.5_mp+0.5*i
  endif
  r=psifun(x,a)
  aer=abs(r-xp(i))
  rer=aer/abs(xp(i))
  write(*,trim(sf)) x, xp(i), aer, rer
enddo
end program test_psi
```

Файл input.

```
10000.0<--= A: параметр, определяющий переход на расчёт по асимптотике
               при X>=A расчёт psix_1(x,a) ведётся по асимптотике,
               за счёт чего точность оказывается порядка 1/(240*X*8).
               при X < A функция psix_1(x,a) сначала наращивает аргумент
               до значения большего A, накапливая поправочную сумму,
               которую затем вычитает из найденного по асимптотике
               значения psi(n+x); n+x --- наименьшее большее A.
```

```
mp=           4
 a=   10.0000000
   X       PSI(X)        aer       rer
-2.50  0.1103157E+01  0.36E-06  0.32E-06
-1.50  0.7031567E+00  0.24E-06  0.34E-06
-0.50  0.3648996E-01  0.83E-06  0.23E-04
 0.50 -0.1963510E+01  0.12E-06  0.61E-07
 1.00 -0.5772157E+00  0.12E-06  0.21E-06
 1.50  0.3648996E-01  0.12E-06  0.33E-05
 2.00  0.4227843E+00  0.12E-06  0.28E-06
 2.50  0.7031567E+00  0.12E-06  0.17E-06
 3.00  0.9227843E+00  0.12E-06  0.13E-06

mp=           8
 a=   10.000000000000000
   X            PSI(X)             aer       rer
-2.50   0.1103156640645244E+01  0.28E-10  0.25E-10
-1.50   0.7031566406452432E+00  0.28E-10  0.39E-10
-0.50   0.3648997397857667E-01  0.28E-10  0.76E-09
 0.50  -0.1963510026021424E+01  0.28E-10  0.14E-10
 1.00  -0.5772156649015329E+00  0.41E-10  0.71E-10
 1.50   0.3648997397857667E-01  0.28E-10  0.76E-09
 2.00   0.4227843350984671E+00  0.41E-10  0.97E-10
 2.50   0.7031566406452432E+00  0.28E-10  0.39E-10
 3.00   0.9227843350984671E+00  0.41E-10  0.44E-10

mp=           8
 a=   100.00000000000000
   X            PSI(X)             aer       rer
-2.50   0.1103156640645244E+01  0.11E-14  0.10E-14
-1.50   0.7031566406452432E+00  0.30E-14  0.43E-14
-0.50   0.3648997397857667E-01  0.38E-14  0.10E-12
 0.50  -0.1963510026021424E+01  0.31E-14  0.16E-14
 1.00  -0.5772156649015329E+00  0.10E-14  0.17E-14
 1.50   0.3648997397857667E-01  0.14E-14  0.40E-13
 2.00   0.4227843350984671E+00  0.32E-14  0.76E-14
 2.50   0.7031566406452432E+00  0.10E-14  0.14E-14
 3.00   0.9227843350984671E+00  0.23E-14  0.25E-14
```

```
mp=          16
 a=   100.0000000000000000000000000000000
   X                      PSI(X)                        aer       rer
-2.50   0.1103156640645243187225690333667911142E+01  0.40E-18  0.36E-18
-1.50   0.7031566406452431872256903336679111l0E+00  0.40E-18  0.57E-18
-0.50   0.3648997397857652055902366700124437 92E-01  0.40E-18  0.11E-16
 0.50  -0.1963510026021423479440976332998755 62E+01  0.40E-18  0.20E-18
 1.00  -0.5772156649015328606065120900824024 71E+00  0.42E-18  0.72E-18
 1.50   0.3648997397857652055902366700124437 92E-01  0.40E-18  0.11E-16
 2.00   0.4227843350984671393934879099175975 29E+00  0.42E-18  0.99E-18
 2.50   0.7031566406452431872256903336679111l0E+00  0.40E-18  0.57E-18
 3.00   0.9227843350984671393934879099175975 29E+00  0.42E-18  0.45E-18

mp=          16
 a=   1000.0000000000000000000000000000000
   X                      PSI(X)                        aer       rer
-2.50   0.1103156640645243187225690333667911142E+01  0.42E-26  0.38E-26
-1.50   0.7031566406452431872256903336679111l0E+00  0.42E-26  0.59E-26
-0.50   0.3648997397857652055902366700124437 92E-01  0.42E-26  0.11E-24
 0.50  -0.1963510026021423479440976332998755 62E+01  0.42E-26  0.21E-26
 1.00  -0.5772156649015328606065120900824024 71E+00  0.42E-26  0.72E-26
 1.50   0.3648997397857652055902366700124437 92E-01  0.42E-26  0.11E-24
 2.00   0.4227843350984671393934879099175975 29E+00  0.42E-26  0.99E-26
 2.50   0.7031566406452431872256903336679111l0E+00  0.42E-26  0.59E-26
 3.00   0.9227843350984671393934879099175975 29E+00  0.42E-26  0.45E-26

mp=          16
 a=   10000.000000000000000000000000000000
   X                      PSI(X)                        aer       rer
-2.50   0.1103156640645243187225690333667911142E+01  0.15E-31  0.14E-31
-1.50   0.7031566406452431872256903336679111l0E+00  0.13E-31  0.18E-31
-0.50   0.3648997397857652055902366700124437 92E-01  0.33E-31  0.91E-30
 0.50  -0.1963510026021423479440976332998755 62E+01  0.13E-31  0.67E-32
 1.00  -0.5772156649015328606065120900824024 71E+00  0.29E-32  0.50E-32
 1.50   0.3648997397857652055902366700124437 92E-01  0.15E-31  0.41E-30
 2.00   0.4227843350984671393934879099175975 29E+00  0.48E-32  0.11E-31
 2.50   0.7031566406452431872256903336679111l0E+00  0.66E-32  0.94E-32
 3.00   0.9227843350984671393934879099175975 29E+00  0.77E-32  0.83E-32
```

9.6 Шестое домашнее задание (III-семестр).

1. Разработать процедуру **E1_as(x,eps,res,k,ier)** расчёта функции $\mathbf{xe^x E_1(x)}$ при $x \gg 1$ по её асимптотическому разложению

$$\mathbf{xe^x E_1(x)} = \sum_{\mathbf{k=0}}^{\infty} (\mathbf{-1})^{\mathbf{k}} \frac{\mathbf{k!}}{\mathbf{x^k}}.$$

   - **x** — аргумент функции; **eps** — относительная погрешность асимптотики.

   Выходные формальные аргументы:

   - **res** — результат расчёта;
   - **k** — число слагаемых асимптотики, обеспечившее достижения требуемого **eps**;
   - **ier** — код завершения работы процедуры:

     0, если точность достигнута

     1, если **eps** недостижимо из-за неподходящей разновидности типа **real**;

     2, если требуемой точности невозможно достичь из-за включения процесса расходимости асимптотики, так как **x** недостаточно велик.

   Тестирующая программа должна для равномерного дробления промежутка **[a,b]** на **n** подучастков выводить таблицу, каждая строка которой содержит **x**, **res**, **E1(x)**, **k**, **ier**.

2. Разработать процедуру **erfc_as(x,eps,res,k,ier)** расчёта функции

$$\sqrt{\pi}\mathbf{xe^{x^2}erfc(x)} \sim \mathbf{1} + \sum_{\mathbf{m=1}}^{\infty} (\mathbf{-1})^{\mathbf{m}} \frac{\mathbf{(2m-1)!!}}{\mathbf{(2x^2)^m}}$$

   Входные и выходные формальные аргументы — те же, что и для первой задачи.

   Тестирующая программа должна для равномерного дробления промежутка **[a,b]** на **n** подучастков выводить таблицу, каждая строка которой содержит **x**, **res**, **erfc(x)**, вычисленную через **res**, **k**, **ier**, **aer**, **rer**, где **aer** — абсолютная погрешность **res** (если за точное значение принять результат, полученный через встроенную **erfc(x)**), а **rer** — соответствующая относительная погрешность.

3. (по желанию) Используя известные асимптотику

$$\ln \mathbf{\Gamma}(\mathbf{x}) \sim (\mathbf{x}-\mathbf{0.5})\ln \mathbf{x} - \mathbf{x} + \mathbf{0.5}\ln \mathbf{2}\pi + \sum_{\mathbf{m=1}}^{\infty} \frac{(\mathbf{-1})^{\mathbf{m}}|\mathbf{B_{2m}}|}{\mathbf{2m(2m-1)x^{2m-1}}}, \quad (\mathbf{x} \to \infty)$$

и функциональное соотношение $\mathbf{\Gamma(x+1) = x\Gamma(x)}$,

разработать процедуру **Gamma_my(x,eps,res,rn,ier)**, которая для аргумента **x>0** вычисляет значение гамма-функции Эйлера и некоторых рабочих величин, используемых требуемым методом расчёта.

Результат работы должен записываться в шестиэлементный вектор **res(1:6)** и целочисленную переменную **ier**:

(a) res(6) - сумма слагаемых асимптотики, зависящих от чисел Бернулли. **eps** — относительная погрешность метода суммирования res(6), а именно: суммирование следует прекратить, как только очередное слагаемое окажется меньше **eps**-доли накопленной суммы.

(b) res(5) - значение ln(Γ(y))/y = res(6)+другие слагаемые асимптотики;

(c) res(4) = ln(Γ(y));

(d) res(3) = накопленное произведение соответствующих сомножителей функционального соотношения **Γ(1+x)=xΓ(x)**, последовательное применение которого позволяет аргумент x увеличить до значения $\mathbf{y \geq RN}$ и приемлемого для расчёта по асимптотике значения $\mathbf{\ln(\Gamma(y))/y}$.

(e) res(2) = Γ(y) = exp(res(4))

(f) res(1) = Γ(x) = res(2)/res(3)

(g) **ier** — код завершения работы процедуры:

- **0**, если требуемое **eps** достигнуто;
- **1**, если **eps** недостижимо из-за неподходящей разновидности типа **real**;
- **2**, если **eps** недостижимо из-за исчерпания чисел Бернулли, используемых подпрограммой. $\mathbf{B_{2m}}$ — числа Бернулли; их значения можно найти в [21].
- **3**, если **eps** недостижимо по причине включения процесса расходимости асимптотики, так как заданное **RN** недостаточно велико.

Желательно, чтобы тестирующая программа, выводила, в частности, оценки абсолютной и относительной погрешностей **res(1)**, если в качестве точного значения $\mathbf{\Gamma(x)}$ использовать результат работы встроенной функции **gamma(x)**.

Список литературы

[1] Горелик А.М. 2006. Программирование на современном Фортране. – М.: Финансы и статистика, – 352 с.

[2] Бартеньев О.В. 1999. Фортран для студентов. – М.:"ДИАЛОГ – МИФИ – 400 с.

[3] Бартеньев О.В. 1999. Visual Fortran. Новые возможности. Издательство "ДИАЛОГ – МИФИ – ??? с.

[4] Бартеньев О.В. 2000. Современный ФОРТРАН. "3-е изд., доп. и перераб. – М.; ДИАЛОГ – МИФИ, – 448 с.

[5] Рыжиков Ю.И. 2004. Современный ФОРТРАН: Учебник. – СПб.: КОРОНА принт, –288 с.

[6] Немнюгин М.А., Стесик О.Л. 2004. Современный ФОРТРАН: Самоучитель. – СПб.: БХВ-Петербург, 496 с.

[7] Браун С. Операционная система UNIX: Пер. с англ. – М.: Мир, 1986.–463 с.

[8] Кэвин Рейчард, Эрик Форстер Джонсон 1999. UNIX–справочник – СПб: Питер Ком, – 384 с.

[9] Немнюгин М.А., Стесик О.Л. 2002. Параллельное программирование для многопроцессорных вычислительных систем. – СПб.: БХВ-Петербург, 400 с.

[10] Немнюгин М.А., Стесик О.Л. 2008. ФОРТРАН в задачах и примерах многопроцессорных вычислительных систем. – СПб.: БХВ-Петербург, 320 с.

[11] Гриффитс А. 2004. gcc. Настольная книга пользователей, программистов и системных администраторов. Пер. с англ./Артур Гриффитс. – К.: “ТИД“ДС”, –624 с.

[12] Игнатов В. 2000. Эффективное использование GNU Make.

[13] Richard M.Stallman, Roand McGrath GNU Make Программа управления компиляцией. GNU make Версия 3.79 Апрель 2000. перевод (С) Владимира Игнатова $http://linux.yaroslavl.ru/docs/prog/gnu\_make\_3-79\_russian\_manual.html$

[14] Левин М. 2005. СИ++: Самоучитель / Максим Левин. – М.: ЗАО “Новый издательский дом”, – 176с.

[15] Люк ???2004. Полный курс С++. Профессиональная работа. – М.: Издательский дом “Вильямс”, –672 с.: ил.

[16] Липский В. 1988. - Комбинаторика для программистов: Пер. с польск. – М.: Мир,– 213.

[17] Д.Кнут. 2017. - Искусство программирования для ЭВМ т. 1. Основные алгоритмы.

[18] Гашков С.Б. 2014 Сложение однобитных чисел. Треугольник Паскаля, салфетка Серпинского и теорема Куммера. — Издательство Московского центра непрерывного математического образования, — 40с.

[19] Генри Уоррен, мл. 2003 Алгоритмические трюки для программистов. Издательский дом "Вильямс"; Москва; Санкт-Петербург; Киев; 285с. алгоритмы.

[20] Цветков А.С. 2005 Руководство по практической работе с каталогом Hipparcos: Учебно-метод. пособие. СПб., 2005. –104 с.

[21] Справочник по специальным функциям с формулами, графиками и таблицами. Под редакцией М. Абрамовица и И. Стиган. 1979. Москва "Наука" Главная редакция физико-математической литературы. Перевод с английского под редакцией В.А. Диткина и Л.Н. Кармазиной. М., 832 стр. с илл.

[22] Numerical recipes in FORTRAN–77: The art of scientific computing (ISBN 0-521-43064-X) Copyright(C) 1986–1992 by Cambridge University Press.

[23] Numerical recipes in FORTRAN–90: The art of scientific computing (ISBN 0-521-43064-X) Copyright(C) 1986–1992 by Cambridge University Press.

[24] Numerical recipes in C: The art of scientific computing (ISBN 0-521-43108-5) Copyright(C) 1988–1992 by Cambridge University Press.

[25] Вирт Н., Алгоритмы + структуры данных = программы: Пер. с англ. – М.: Москва, Мир, 1985. – 406 с., ил.

[26] ВЖ.ФОРСАЙТ, М.МАЛЬКОЛЬМ, К.МОУЛЕР, Машинные методы математических вычислений, МИР, МОСКВА 1980 СТР. 118

[27] https://habr.com/en/company/intel/blog/255305/.

[28] http://igorsbox.wix.com/lections

[29] www.vbstreets.ru/VB/Articles/66541.aspx