

Мартыненко Борис Константинович

# УЧЕБНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПРОЕКТ РЕАЛИЗАЦИИ АЛГОРИТМИЧЕСКИХ ЯЗЫКОВ: УСЛОВНЫЕ И ВАРИАНТНЫЕ ПРЕДЛОЖЕНИЯ

## Аннотация

Рассматривается реализация условных и вариантных предложений как конструкций в объектно-ориентированной модели семантики на примере языка программирования Алгол 68.

**Ключевые слова:** балансировка видов, вариантное предложение, приводимое, условное предложение.

## 1. ВВЕДЕНИЕ

Статья продолжает обсуждение проекта реализации алгоритмических языков на базе объектно-ориентированного описания семантики [2–6].

В Алголе 68 [1] есть три разновидности *выбирающих* предложений:

- *условные* (выбирающие по логическому значению),
- *вариантные* (выбирающие по целому значению),
- *сопоставляющие* предложения (выбирающие по конкретному виду, объединённому из нескольких видов).

Все они могут играть роль *оператора*, если контекст, в котором они используются, определяет их вид как **void** или *выражения* с результатом определённого вида.

Оставляя реализацию сопоставляющих предложений до рассмотрения объединённых видов на будущее, рассмотрим две первых разновидности *выбирающих* предложений.

Напомним, что основные формы условного и вариантного предложений<sup>1</sup> представляются схемами, изображёнными на рис. 1 и 2.

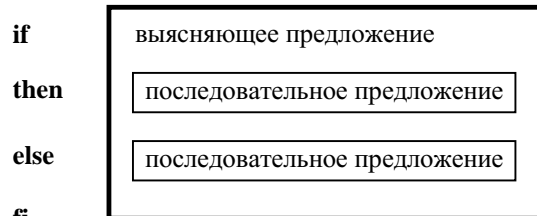


Рис. 1. Блочная структура условного предложения

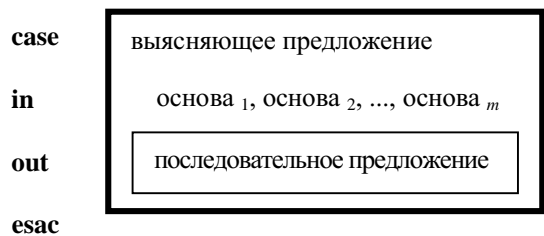


Рис. 2. Блочная структура вариантного предложения

<sup>1</sup> Без синтаксического «сахара» для частных случаев.

Рамками показаны области действия индикаторов, описанных в соответствующих предложениях, составляющих данные выбирающие предложения.

Индикаторы, описанные в выясняющем предложении, можно использовать в любом последовательном предложении в составе данного выбирающего предложения и в основах вариантного. Индикаторы, описанные в любой из ветвей условного предложения и в ветви **-out**, можно использовать только в ней.

Напомним, что последовательное предложение образует *локализирующее* окружение, то есть участок в стеке данных, присоединяемый к текущему окружению, если он содержит описания индикаторов, то есть является блоком в традиционном смысле. Это относится и к последовательным предложениям в составе выбирающего предложения.

*Выясняющее предложение* в составе условного или вариантного предложений – это последовательное предложение *логического* или *целого* вида соответственно, в *раскрытой* позиции [6], не содержащее *определяющих* вхождений меток. Это значит, что за время исполнения таких предложений выясняющее предложение исполняется ровно один раз.

**Then-** и **else-** ветви условного предложения должны приводиться к одному и тому же виду с учётом того, что одна из ветвей занимает *сильную* позицию, а сорт позиции другой ветви – такой же, как всего условного предложения.

Аналогично виды составляющих конструкций вариантного предложения определяются, исходя из того, что одна из основ или **out**-предложение занимают позицию того же сорта, что и всё вариантное предложение, тогда как остальные в сильной позиции. Такая *балансировка видов* является задачей синтаксического анализа и здесь не рассматривается.

В Алголе 68 приведения выполняются над результатами последних основ последовательных предложений перед передачей их объёмлющей конструкции. Причём приведения касаются только *приводимых* (*coercend*), к которым по синтаксису языка Алгол 68 относятся все основы, кроме ЗАМКНУТЫХ<sup>1</sup> предложений, *переходов* и *псевдоимён*.

В рассматриваемой ОО-модели семантики все приводимые являются наследниками объекта типа **TCoercend**, единственное поле данных которого есть указатель на строку, представляющую план приведений [6].

Именно, вводится объект типа TCoercend:

```

type PCoercend = ^TCoercend;
      TCoercend = object (TConstruct)
      { План приведений }
      coercion_plan: PChar;
      constructor Init (cp : PChar);
end;

с реализацией конструктора

constructor TCoercend.Init (r, cp : PChar);
begin inherited Init (r); coercion_plan := cp end;

```

Поле **coercion\_plan**, равное **nil**, означает, что фактически никаких приведений не требуется. Иначе, значение поля **coercion\_plan** используется для управления преобразованиями значения приводимой основы, априорное значение которого зафиксировано в UV [6].

В связи с введением в модель семантики типа **TCoercend**, объекты-конструкции, являющиеся приводимыми, должны быть наследниками этого типа.

Например, конструкции ‘изображение целого’ или ‘изображение вещественного’ следует пересмотреть. Именно, обновлённые версии этих конструкций теперь описываются следующим образом:

<sup>1</sup> То есть кроме замкнутых, выбирающих и циклических предложений.

```
type PIntegralDenotation = ^TIntegralDenotation;  
TIntegralDenotation = object (TCoercend)  
  Value : integer; { априорное значение }  
  constructor Init (r, cp : PChar; v : integer);  
  function Show : PChar; virtual;  
  procedure Run; virtual;  
end;
```

с реализацией методов

```
constructor TIntegralDenotation.Init (r, cp : PChar; v : integer);  
begin inherited Init (r, cp); Value := v end;  
function TIntegralDenotation.Show : PChar;  
begin Show := Representation end;  
procedure TIntegralDenotation.Run;  
begin UV := New (PIntegralValue, Init (Value));  
      if coercion_plan <> nil then Coercing (coercion_plan^)  
end;
```

и

```
type PRealDenotation = ^TRealDenotation;  
TRealDenotation = object (TCoercend)  
  Value : real;  
  constructor Init (r, cp : PChar; v : real);  
  function Show : PChar; virtual;  
  procedure Run; virtual;  
end;
```

с реализацией методов

```
constructor TRealDenotation.Init ( r, cp : PChar; v : real );  
begin inherited Init (r, cp); Value := v end;  
function TRealDenotation.Show : PChar;  
begin Show := Representation end;  
procedure TRealDenotation.Run;  
begin UV := New (PRealValue, Init (Value));  
      if coercion_plan <> nil then Coercing (coercion_plan^)  
end;
```

## 2. РЕАЛИЗАЦИЯ КОНСТРУКЦИИ ‘УСЛОВНОЕ ПРЕДЛОЖЕНИЕ’

Конструкция ‘условное предложение’ представляется как объект типа **TConditional\_clause** со следующим предписанием:

```
type PConditional_clause = ^TConditional_clause;  
TConditional_clause = object (TConstruct)  
  choice_clause, then_clause, else_clause : PConstruct;  
  constructor Init (cc, tc, ec : PConstruct);  
  destructor Done; virtual;  
  function Show : PChar; virtual;  
  procedure Run; virtual;  
end;
```

и реализацией методов

```

constructor TConditional_clause.Init (cc, tc, ec : PConstruct);
begin choice_clause := cc; then_clause := tc; else_clause := ec end;

destructor TConditional_clause.Done;
begin end;

function TConditional_clause.Show : PChar;
var Bf : array [0 .. 1023] of Char;
begin
    StrPCopy (Bf, '.if ');
    StrCat (Bf, choice_clause^.Show);
    StrCat (Bf, ' .then '); StrCat (Bf, then_clause^.Show);
    StrCat (Bf, ' .else '); StrCat (Bf, else_clause^.Show);
    StrCat (Bf, ' .fi');
    Show := Bf
end;

procedure TConditional_clause.Run;
begin
    choice_clause^.Run;
    if PBooleanValue (UV) ^.Value then then_clause^.Run else else_clause^.Run;
end;

```

Представление полей данных **choice\_clause**, **then\_clause** и **else\_clause** в виде указателей на объекты-конструкции *любого* типа вместо *последовательных предложений* (**TRange**), коими они буквально являются по синтаксису языка Алгол 68, даёт возможность не выполнять *предысполнений* последовательных предложений в тех случаях, когда они фактически составлены одной приводимой основой.

В листинге 1 приводится программа на промежуточном инструментальном языке Free Pascal [7] построения семантического дерева алгол-программы, включающей два условных предложения.

Заметим, что в протоколе построения семантического дерева программы и её исполнения (рис. 3) приведены выдачи некоторых семантических модулей, предусмотренных отладочным режимом компиляции.

Отметим следующие особенности этой программы.

1. Единственный блок этой программы не содержит описаний и потому является не локализирующим: в стеке не образуется ни одного участка (locale) и таблица Display не содержит ни одного элемента.

2. Априори изображение вещественного 3.14 и изображение целого 0, составляющие ветви обоих условных предложений, дают значения разных видов – *integral* и *real* соответственно.

3. Балансировка видов этих условных предложений вызывает приведение целого значения 0 к эквивалентному вещественному значению.

4. Результаты условных предложений выбираются по соответствующему логическому значению выясняющего предложения и фиксируются в **UV** как значения вида *real*.

5. Эти значения *опустошаются*, то есть просто игнорируются, что не требует никаких действий.

6. Сама программа как замкнутое предложение имеет пустой результат (empty) в соответствии с описанием конструкции ‘программа’ в Алголе 68.

В предпоследней строчке протокола показано остаточное значение второго условного предложения, зафиксированного в **UV** к моменту завершения интерпретации семантического дерева алгол-программы.

**Листинг 1.** Построение семантического дерева программы

```

.begin .if .true .then 3.14 .else 0 .fi ; .if .false .then 3.14 .else 0 .fi .end'
program conditional_test_1;
uses CRT, PLAIN_VALUES, CONSTRUCTS, CLAUSES, ENVIRON ;
var bd, bdl : PBooleanDenotation;
    id : PIntegralDenotation;
    rd : PRealDenotation;
    bds, bdls, ids, rds, id_cp: string;
    if_cl, if_cl1 : PConditional_Clause;
    cl0 : PConstructList;
    Range0 : PRange;
    main: PClosedClause;
begin ClrScr;
    writeln (#13#10' Тестирование программы Алгола 68:');
    writeln ('.begin .if .true .then 3.14 .else 0 .fi ; .if .false .then 3.14 .else 0 .fi .end');
    writeln (#13#10'   ПРОСТРАСТВО ДАННЫХ: '#10#13);
  { СОЗДАНИЕ СТЕКА ДАННЫХ }
    Stack := New (PStack, Init (1));
    writeln ('Стек на ', Stack^.Limit, ' участок);
  { СОЗДАНИЕ ТАБЛИЦЫ DISPLAY }
    Display := New (PDisplay, Init (1));
    writeln ('  Display на ', Display^.Limit, ' участок'#13#10);
    writeln (' == СОЗДАНИЕ СЕМАНТИЧЕСКОГО ДЕРЕВА ПРОГРАММЫ =='#13#10);
  { СОЗДАНИЕ КОНСТРУКЦИЙ БЛОКА 0 }
    bds := 'true';
    bd := New (PBooleanDenotation, Init (@bds[1], true));
    writeln ('  Изображение логического: ', bd^.Show);
    bdls := 'false';
    bdl := New (PBooleanDenotation, Init (@bdls[1], false));
    writeln ('  Изображение логического: ', bdl^.Show);
    ids := '0'; id_cp := 'd'; { d - обобщение целого до вещественного }
    id := New (PIntegralDenotation, Init (@ids[1], @id_cp[1], 0));
    writeln ('  Изображение целого: ', id^.Show);
    rds := '3.14'; { rd_cp = nil - приведений не требуется! }
    rd := New (PRealDenotation, Init (@ids[1], nil, 3.14));
    writeln ('  Изображение вещественного: ', rd^.Show);
    if_cl := New (PConditional_Clause, Init (bd, rd, id));
    writeln ('  Условное предложение: ', if_cl^.Show);
    if_cl1 := New (PConditional_Clause, Init (bdl, rd, id));
    writeln ('  Условное предложение: ', if_cl1^.Show);
  { Создание списка конструкций блока 0 }
    cl0 := New (PConstructList, Init (2, 0));
    cl0^.Insert (if_cl );
    cl0^.Insert (if_cl1);
  { Создание блока 0 }
    Range0 := New (PRange, Init (0, 10, cl0));
  { Создание программы }
    main := New (PClosedClause, Init (Range0));
    writeln (#13#10'   СЕМАНТИЧЕСКОГО ДЕРЕВА ПРОГРАММЫ СОЗДАНО:', main^.Show);
    writeln (#13#10'   ЗАПУСК ПРОГРАММЫ ... '#13#10);
    main^.Run;
    writeln ('  UV = ', PRealValue (UV)^.Show);
    writeln ('  ПРОГРАММА ВЫПОЛНЕНА!'); readln;
end.

```

```

Тестирование программы Алгола 68:
.begin .if .true .then 3.14 .else 0 .fi; .if .false .then 3.14 .else 0 .fi .end

ПРОСТРАНСТВО ДАННЫХ:

Стек на 1 участок
Display на 1 участок

===== СОЗДАНИЕ СЕМАНТИЧЕСКОГО ДЕРЕВА ПРОГРАММЫ =====

Изображение логического: true
Изображение логического: false
Изображение целого: 0
Изображение вещественного: 3.14
Условное предложение: .if true .then 3.14 .else 0 .fi
Условное предложение: .if false .then 3.14 .else 0 .fi

СЕМАНТИЧЕСКОЕ ДЕРЕВО ПРОГРАММЫ СОЗДАНО:
BEGIN<0>
  [0] .if true .then 3.14 .else 0 .fi
  [1] .if false .then 3.14 .else 0 .fi
END<0>

ЗАПУСК ПРОГРАММЫ ...

TRange.Run начала ...

Stack^.AddLocale <locale>:
Stack [0] =

ConstructList:
  [0] .if true .then 3.14 .else 0 .fi
  [1] .if false .then 3.14 .else 0 .fi
Jump_elaborated = FALSE

Исполнение коллекции фраз блока ...

TConstructList.RunWhile начала ...

[0]: .if true .then 3.14 .else 0 .fi

РЕЗУЛЬТАТ ExecItem <0>: 3.1400000000000000E+000

Состояние стека после исполнения .if true .then 3.14 .else 0 .fi:

Display [0] ::
[1]: .if false .then 3.14 .else 0 .fi

Выполнено widening
UV = 0.0000000000000000E+000
РЕЗУЛЬТАТ ExecItem <1>: 0.0000000000000000E+000

Состояние стека после исполнения .if false .then 3.14 .else 0 .fi:

Display [0] ::

TConstructList.RunWhile кончила!

ТЕКУЩЕЕ ОКРУЖЕНИЕ ПОСЛЕ ВЫХОДА ИЗ БЛОКА ТЕКУЩЕГО УРОВНЯ

ПУСТО

TRange.Run кончила

UV = 0.0000000000000000E+000

ПРОГРАММА ВЫПОЛНЕНА!

```

Рис. 3. Протокол исполнения программы conditional\_test\_1

### 3. РЕАЛИЗАЦИЯ КОНСТРУКЦИИ ‘ВАРИАНТНОЕ ПРЕДЛОЖЕНИЕ’

Конструкция ‘вариантное предложение’ представляется как объект типа **TCase\_clause** со следующим предописанием:

```

type PCase_clause = ^TCase_clause;
      TCase_clause = object (TConstruct)

```

```

choice_clause: PConstruct;
in_part: PConstructList;
out_part : PConstruct;
constructor Init (cc: PConstruct; ip: PConstructList; op : PConstruct);
destructor Done; virtual;
function Show : PChar; virtual;
procedure Run; virtual;
end;

```

и реализацией методов

```

constructor TCase_clause.Init (cc: PConstruct; ip: PConstructList; op : PConstruct);
begin choice_clause := cc; in_part := ip; out_part := op end;

destructor TCase_clause.Done;
begin end;

function TCase_clause.Show : PChar;
var Bf : array [0 .. 1023] of Char;
begin
  StrPCopy (Bf, #13#10      '.case');
  StrCat (Bf, choice_clause^.Show);
  StrCat (Bf, #13#10      ' .in '); StrCat (Bf, in_part^.Show);
  StrCat (Bf, #13#10      ' .out '); StrCat (Bf, out_part^.Show);
  StrCat (Bf, #13#10      ' .esac');
  Show := Bf
end;

procedure TCase_clause.Run;
var i: integer;
begin
  choice_clause^.Run;
  i := PIntegralValue (UV)^.Value;
  if (1 <= i) and (i <= in_part^.Count)
  then in_part^.RunUnit(i-1)
  else out_part^.Run;
end;

```

Здесь используется метод **RunUnit** объекта типа **TConstructList** со следующим предописанием:

```
procedure RunUnit (i : integer); virtual;
```

и реализацией

```

procedure TConstructList.RunUnit (i : integer);
{ Исполнение основы номер i из списка основ }
begin PConstruct (At(i))^.Run end;

```

В листинге 2 приводится программа на промежуточном инструментальном языке Free Pascal [7] построения семантического дерева алгол-программы, включающей три вариантных предложения, не требующих приведений.

Отметим следующие особенности этой программы.

1. Единственный блок этой программы не содержит описаний и потому является не-локализуемым. Поэтому в стеке не образуется ни одного участка (locale), и таблица Display не содержит ни одного элемента.

2. Последовательное предложение, составляющее программу, состоит из трёх вариантных предложений, отличающихся только выясняющими предложениями. Каждое из них представлено конструкцией 'изображение целого' 1, 2 и 0, соответственно. Список in-основ состоит из изображений целых 1 и 2, а out-предложение представлено изображе-

**Листинг 2.** Построение семантического дерева программы

```

.begin .case 1 .in 1, 2 .out 0 .esac; .case 2 .in 1, 2 .out 0 .esac; .case 0 .in 1, 2 .out 0 .esac .end

program case_test_1;
uses CRT, PLAIN_VALUES, CONSTRUCTS, CLAUSES, ENVIRON ;
var id0, id1, id2: PIntegralDenotation;
    ids, id1s, id2s: string;
    ip: PConstructList;
    case_cl1, case_cl2, case_cl3 : PCase_Clause;
    cl0 : PConstructList;
    Range0 : PRange;
    main: PClosedClause;
begin ClrScr;
    writeln (#13#10'      Тестирование программы Алгола  68:');
    writeln ('      .begin .case 1 .in 1, 2 .out 0 .esac;');
    writeln ('      .begin .case 2 .in 1, 2 .out 0 .esac;');
    writeln ('      .begin .case 0 .in 1, 2 .out 0 .esac;');
    writeln ('      .end');
    writeln (#13#10'      ПРОСТРАНСТВО ДАННЫХ: '#10#13);
{ СОЗДАНИЕ СТЕКА ДАННЫХ }
    Stack := New (PStack, Init (1));
    writeln ('Стек на ', Stack^.Limit, ' участок);
{ СОЗДАНИЕ ТАБЛИЦЫ DISPLAY }
    Display := New (PDisplay, Init (1));
    writeln ('      Display на ', Display^.Limit, ' участок'#13#10);
    writeln ('      === СОЗДАНИЕ СЕМАНТИЧЕСКОГО ДЕРЕВА ПРОГРАММЫ ==='#13#10);
{ СОЗДАНИЕ КОНСТРУКЦИЙ БЛОКА 0 }
    id0s := '0';
    id0 := New (PIntegralDenotation, Init (@id0s[1], nil, 0));
    writeln ('      Изображение целого: ', id0^.Show);
    id1s := '1';
    id1 := New (PIntegralDenotation, Init (@id1s[1], nil, 1));
    writeln ('      Изображение целого: ', id1^.Show);
    id2s := '2';
    id2 := New (PIntegralDenotation, Init (@id2s[1], nil, 2));
    writeln ('      Изображение целого: ', id1^.Show);
{ Создание списка основ in_part }
    ip := New (PConstructList, Init (2, 0));
    ip^.Insert (ip1); ip^.Insert (ip2);
    case_cl1 := New (PCase_Clause, Init (id1, ip, id0));
    case_cl2 := New (PCase_Clause, Init (id2, ip, id0));
    case_cl3 := New (PCase_Clause, Init (id0, ip, id0));
{ Создание списка конструкций блока 0 }
    cl0 := New (PConstructList, Init (3, 0));
    cl0^.Insert (case_cl1 );
    cl0^.Insert (case_cl2);
    cl0^.Insert (case_cl3);
{ Создание блока 0 }
    Range0 := New (PRange, Init (0, 10, cl0));
{ Создание программы }
    main := New (PClosedClause, Init (Range0));
    writeln (#13#10' СЕМАНТИЧЕСКОГО ДЕРЕВА ПРОГРАММЫ СОЗДАНО:', main^.Show);
    writeln (#13#10'      ЗАПУСК ПРОГРАММЫ ... '#13#10);
    main^.Run;
    writeln ('      UV = ', PRealValue (UV)^.Show);
    writeln ('      ПРОГРАММА ВЫПОЛНЕНА!'); readln;
end.

```



```

Тестирование программы Алгола 68:
.begin .case 1 .in 1, 2 .out 0 .esac;
      .case 2 .in 1, 2 .out 0 .esac;
      .case 0 .in 1, 2 .out 0 .esac
.end

ПРОСТРАНСТВО ДАННЫХ:

Стек на 1 участок
Display на 1 участок

===== СОЗДАНИЕ СЕМАНТИЧЕСКОГО ДЕРЕВА ПРОГРАММЫ =====

Изображение целого: 0
Изображение целого: 1
Изображение целого: 2

СЕМАНТИЧЕСКОЕ ДЕРЕВО ПРОГРАММЫ СОЗДАНО:

BEGIN(0)
  [0] .case 1
      .in
  [0] 1
  [1] 2
      .out 0
      .esac
  [1] .case 2
      .in
  [0] 1
  [1] 2
      .out 0
      .esac
  [2] .case 0
      .in
  [0] 1
  [1] 2
      .out 0
      .esac
END(0)

ЗАПУСК ПРОГРАММЫ ...

Вариантное предложение:
.case 1
  .in
  [0] 1
  [1] 2
      .out 0
      .esac
даёт результат = 1

Вариантное предложение:
.case 2
  .in
  [0] 1
  [1] 2
      .out 0
      .esac
даёт результат = 2

Вариантное предложение:
.case 0
  .in
  [0] 1
  [1] 2
      .out 0
      .esac
даёт результат = 0

ТЕКУЩЕЕ ОКРУЖЕНИЕ ПОСЛЕ ВЫХОДА ИЗ БЛОКА ТЕКУЩЕГО УРОВНЯ
ПУСТО
UV = 0
ПРОГРАММА ВЫПОЛНЕНА!

```

Рис. 4. Протокол исполнения программы case\_test\_1

нием целого 0. По контексту все они не требуют приведений, хотя являются приводимыми (планы приведений представлены как nil).

3. Балансировка видов в этих вариантных предложениях не требуется.

4. Результаты вариантных предложений выбираются по соответствующему целому значению выясняющего предложения (1, 2 и 0) и фиксируются в UV как значения вида *intergal*.

5. Эти значения *опустошаются*, то есть просто игнорируются, что не требует никаких действий.

6. Сама программа как замкнутое предложение имеет пустой результат (empty) в соответствии с описанием конструкции 'программа' в Алголе 68.

В предпоследней строчке протокола показано остаточное значение второго вариантного предложения, зафиксированного в UV к моменту завершения исполнения программы.

Заметим, что в протоколе, как и в примере 1 построения семантического дерева программы и его интерпретации приведены выдачи семантических модулей, предусмотренных отладочным режимом компиляции промежуточной программы<sup>1</sup>.

## ЗАКЛЮЧЕНИЕ

Условные и вариантные предложения используются во многих языках программирования, поэтому в контексте описываемого проекта их реализация представляет интерес. Именно на этих конструкциях уместно продемонстрировать явление балансировки видов и понятие приводимых конструкций (coercend).

<sup>1</sup> В опубликованных описаниях модулей проекта эти выдачи не показаны.

## Литература

1. Под ред. А. ван Вейнгаарден, Б. Майу, Дж. Пек, К. Костер и др. Пересмотренное сообщение об Алголе 68. М., 1979. 533 с.

2. Мартыненко Б.К. Учебный исследовательский проект реализации алгоритмических языков // Компьютерные инструменты в образовании, 2008. № 5. С. 3–18.

3. Мартыненко Б.К. Учебный исследовательский проект реализации алгоритмических языков: значения и конструкции // Компьютерные инструменты в образовании, 2009. № 1. С. 10–25.

4. Мартыненко Б.К. Учебный исследовательский проект реализации алгоритмических языков: описания и окружения // Компьютерные инструменты в образовании, 2009. № 2. С. 12–29.

5. Мартыненко Б.К. Учебный исследовательский проект реализации алгоритмических языков: генераторы и имена // Компьютерные инструменты в образовании, 2009. № 3. С. 3–17.

6. Мартыненко Б.К. Учебный исследовательский проект реализации алгоритмических языков: приведения // Компьютерные инструменты в образовании, 2009. № 4. С. 5–29.

7. Michaël Van Canneyt. Reference guide for Free Pascal. 2002. 188 p.

## Abstract

The implementation of the conditional clause and the case clause based on the object-oriented model of semantics of the programming language Algol 68, as example, is discussed.



Наши авторы, 2009.  
Our authors, 2009.

Мартыненко Борис Константинович,  
доктор физико-математических  
наук, профессор кафедры  
информатики математико-  
механического факультета СПбГУ,  
mbk@ctinet.ru