Мартыненко Борис Константинович

УЧЕБНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПРОЕКТ РЕАЛИЗАЦИИ АЛГОРИТМИЧЕСКИХ ЯЗЫКОВ: ГЕНЕРАТОРЫ И ИМЕНА

Аннотация

Рассматривается моделирование генераторов Алгола 68 в форме объектов-конструкций, а также результатов их исполнения, то есть значений имён соответствующих видов. В частности, обсуждается представление косвенных имён. Демонстрируется их использование на примерах.

Ключевые слова: генератор, имя, присваивание.

1. ВВЕДЕНИЕ

В [1] была сформулирована тема моделирования гипотетического вычислителя, используемого в [2] для описания семантики алгоритмического языка типа Алгол 68 в терминах понятий объектно-ориентированного программирования (ООП). При этом объекты: *конструкт*, *значение*, *участок*, *окружение* или *сцена*, с которыми манипулирует гипотетический вычислитель, отображаются в соответствующие классы объектов [3–4].

В статье рассматривается моделирование генераторов в форме объектов-конструкций и имён соответствующих видов в качестве результатов их исполнения. При этом существенно используется аналогия между генераторами Алгола 68 и операторами New Паскаля, а также между именами Алгола 68 и типизированными указателями Паскаля.

2. МОДЕЛИРОВАНИЕ ГЕНЕРАТОРОВ

Согласно [2], *имена*, значения, предназначенные для именования других значений, создаются в результате исполнения *генераторов*.

Конструкция генератора текстуально состоит из символа `loc` (генератор *локальный*) или `heap` (генератор *глобальный*) и следующего за ним *фактического* описателя вида значения, именуемого именем, выдаваемым генератором. Фактический описатель сам по себе или контекстно через описание индикатора вида содержит достаточно информации о структуре области памяти и её объёме для представления значения-образца[1] .

Исполнение генератора даёт имя, именующее *неопределённое* значение вида, специфицированного фактическим описателем.

[1] Значение, именуемое именем, выдаваемым генератором.

© Б.К. Мартыненко, 2009

Генераторы *локальные* и *глобальные* различаются областями действия имён, выдаваемых ими. Локальные имена доступны на время существования окружения, в котором исполняется локальный генератор, а глобальное имя существует всё время исполнения программы.

Исполнение генератора в машинно-ориентированных терминах состоит в размещении значения-образца в реальной куче и выдаче типизированного указателя на него в административной переменной полиморфной **UV**, который и служит представлением имени соответствующего вида в описываемой модели семантики. При этом предполагается, что в качестве образца используется значение, традиционно принимаемое по умолчанию (*default value*) во множестве значений данного вида при выбранном способе его адекватного представления в модели.

Глобальное имя может передаваться из одного окружения в другое без всяких ограничений, тогда как локальное – лишь вдоль статических цепочек участков в стеке данных в пределах области действия этого имени.

Модельная реализация генератора базируется на возможности конвертера формировать по тексту фактического описателя генератора эквивалентное описание типа объекта значения-образца (**PMode**) и использовании этого типа в операторе **New** в паре с конструктором (**Init**) объекта значения-образца в форме оператора вида

```
UV := New (PMode, Init(…)).
```

Например, пусть на языке Алгол 68 имеем задание генератора **.loc.bool**. Эффект его исполнения в терминах описываемой модели семантики воспроизводится посредством оператора **UV := New (PBooleanValue, Init (false))**, который оставляет имя логического в форме указателя на *default*-значение **(false)** в полиморфной переменной **UV**.

Как использовать значение генератора, зафиксированное в административной переменной **UV**, зависит от конструкции, в состав которой непосредственно входит данный генератор.

В рассматриваемой семантической модели значение подконструкции, в данном случае генератора, может быть передано конструкции, использующей имя из **UV**, либо «из рук в руки», то есть через поле данных объекта-конструкции, использующей выдачу генератора, либо «через стек», то есть с использованием *именованного* или *анонимного* индикатора на текущем участке стека. Доступ к значению, которым обладает индикатор, как принято в модели, осуществляется посредством его статического адреса.

Например, пусть генератор **.loc.bool** входит в описание тождества

```
.ref.bool b = .loc.bool                                    (1)
```

Оно устанавливает связь идентификатора **b** вида **.ref.bool** со значением генератора, именем такого же вида.

Эквивалент этого описания тождества в расширенном или, если угодно, сокращённом варианте в форме *описания переменной* Алгола 68

```
.bool b                                                    (2)
```

демонстрировался в [4]. По сравнению с конструкцией (1) в (2) пропущен источник, генератор, а формальный описатель вида **.ref.bool**, предшествующий индикатору **b** в полном варианте этой конструкции, заменён на фактический описатель значения-образца именуемое значением (именем), которым обладает идентификатор **b**.

Так или иначе, идентификатор **b** обладает именем логического значения, и, стало быть, ему можно присваивать различные логические значения с помощью конструкции присваивания логического вида.

Рассмотрим два описания тождества:

```
.bool b = false;                                    (3)
.ref.bool rb = .loc.bool;                           (4)
```

В первом случае идентификатор **b** обладает логическим значением **false**, а во втором – именем, которое именует логическое значение по умолчанию **(false)**.

На уровне модельной реализации оба варианта ничем не различаются (*см*. рис. 1). Семантически разница в том, что в описании тождества (4) идентификатор **b** обладает переменной, текущее значение которой можно менять с помощью присваивания, а в (3) – константой в обычном понимании этого термина, и присваивание не допустимо. Контроль за соблюдением этого семантического различия – задача синтаксического анализа входной программы, осуществляемая конвертером (рис. 1).

Как было сказано выше, различие между именами по областям действия должно учитываться при исполнении присваиваний. Именно, область действия имени получателя не должна быть старше области действия значения источника[1] . Для этого и предусмотрено поле **Scope** в экземпляре каждого объекта-значения любого вида.

Контроль областей действия значений, участвующих в присваивании, «забота» метода **Run** конструкции присваивания.

Проблема контроля областей действия значений возникает при исполнении некоторых других конструкций.

3. ОБЛАСТИ ДЕЙСТВИЯ ЗНАЧЕНИЙ

Каждое значение имеет определенную «область действия», зависящую от его вида или способа, которым оно получено. Область действия значения определяется так, чтобы совпадать с областью действия некоторого окружения. Содержательно, область действия значения – это время существования значения в процессе исполнения программы. Это понятие связано с доступностью значений программе в разные моменты её исполнения.

Выяснять на этапе синтаксического анализа, будет ли существовать значение, которым будет обладать индикатор во время исполнения программы, в общем случае невозможно. Поэтому в объектно-ориентированном представлении значения любого вида предусмотрено поле **Scope**, содержащее статическую характеристику области действия данного значения. Этой характеристикой является уровень блока, образующий участок в стеке данных, с которого данное значение непосредственно доступно. Чем больше величина поля **Scope** одного из двух значений, тем младше его область действия, короче период его существования.

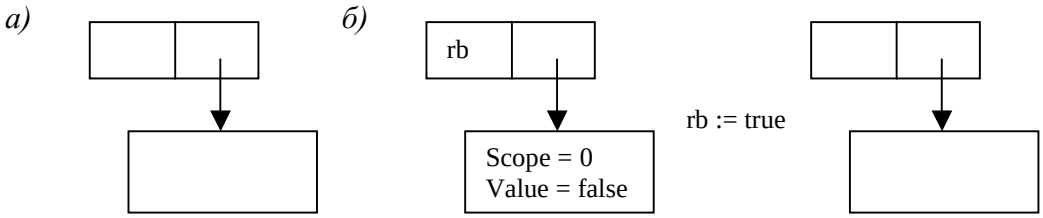

Рис. 1. а) представление идентификатора константы `b`,
б) представление идентификатора `rb` на участке блока
до и после присваивания переменной,

[1] Ибо в противном случае это имя в некоторый другой момент будет именовать уже не существующее значение.

Для значений простых видов, таких как **integral**, **real**, **boolean**, **character** и т. д., поле **Scope = 0** в знак того, что значения этих видов существуют всё время исполнения программы.

Такая же величина поля **Scope** относится к любому значению глобального генератора (глобальному имени) или к процедуре (**routine**), не использующей глобальных индикаторов в своём теле.

В остальных случаях области действия значений устанавливаются следующим образом:

1) имя любого вида как результат локального генератора:

– уровень минимального блока, создающего локализующее окружение, в котором находится данный генератор;

2) процедура (**routine**), представленная как значение сцены:

– уровень необходимого окружения для этой сцены;

3) массив:

– минимальное значение поля **Scope** по всем значениям элементов;

4) структура:

– минимальное значение поля **Scope** по всем значениям полей.

Чтобы пояснить понятие областей действия значений и описать его воплощение в модели семантики, приведём фрагмент модуля **CONSTRUCTS**, описывающего генераторы и присваивания логического вида (см. листинг I), и пример, иллюстрирующий вышесказанное, а именно программу на Алголе 68 вида **.begin** .bool b =.true; .ref.bool rb =.loc.bool; rb := b **.end** (см. листинг II).

Листинг I.

```
unit CONSTRUCTS;
{ Реализация конструкций языка }
interface
uses objects, Strings,
     VALUES, PLAIN_VALUES, ENVIRON, STANDART;
type
      ...
PGenerator = ^TGenerator;
TGenerator = object (TConstruct)
   is_loc : Boolean;
   constructor Init (r : PChar; loc : Boolean);
   function Show : PChar; virtual;
   procedure Run; virtual;
  end;
  PBooleanGenerator = ^TBooleanGenerator;
  TBooleanGenerator = object (TGenerator)
   constructor Init (r : PChar; loc : Boolean);
   function Show : PChar; virtual;
   procedure Run; virtual;
  end;
    ...
  { П Р И С В А И В А Н И Е }
  PAssignment = ^TAssignment;
  TAssignment = object (TConstruct)
  (* Наследуемые поля данных:
   Representation : PChar; *)
   Destination { Получатель },
   Source { Источник } : PConstruct;
   DestinationAddr { Адрес значения получателя },
```

```
  SourceAddr { Адрес значения источника } : PAddr;
  function Show : PChar; virtual;
  procedure Run; virtual;
end;

PBooleanAssignment = ^TBooleanAssignment;
TBooleanAssignment = object (TAssignment)
(* Наследуемые поля данных:
  Representation : PChar;
  Destination : { Получатель }, Source { Источник } : PConstruct;
  DestinationAddr { Адрес значения получателя },
  SourceAddr { Адрес значения источника } : PAddr; *)
  DestinationValue, SourceValue : PBooleanValue;
  constructor Init (r : PChar; D, S : PConstruct; da, sa : PAddr);
  function Show : PChar; virtual;
  procedure Run; virtual;
end;
...
implementation
...
{ Генераторы }

constructor TGenerator.Init (r : PChar; loc : Boolean);
begin Representation := r; is_loc := loc end;

function TGenerator.Show : PChar;
begin abstract end;

procedure TGenerator.Run;
begin abstract end;

{ Генератор логического }

constructor TBooleanGenerator.Init (r : PChar; loc : Boolean);
begin inherited Init (r, loc) end;

function TBooleanGenerator.Show : PChar;
begin Show := Representation end;

procedure TBooleanGenerator.Run;
var ref_to_bool : PBooleanValue;
begin
  { Создание нового экземпляра логического значения }
    ref_to_bool := New (PBooleanValue, Init (false));
  { Обновление области действия имени логического }
   if is_loc
    then ref_to_bool^.Scope := CurrentLevel
    else ref_to_bool^.Scope := 0;
  { Выдача имени логического в виде указателя на контейнер логического}
    UV := ref_to_bool;
end;
{ П Р И С В А И В А Н И Е }

function TAssignment.Show : PChar;
begin abstract end;

procedure TAssignment.Run;
begin abstract end;

constructor TBooleanAssignment.Init (r : PChar; D, S : PConstruct;
                                     da, sa : PAddr);
```

```
  begin Representation := r;
        Destination := D;        Source := S;
        DestinationAddr := da;   SourceAddr := sa
  end;
  function TBooleanAssignment.Show : PChar;
  var Bf, Bf1 : array [0..512] of char;
  begin
    if Destination <> nil
    then StrPCopy (Bf, Destination^.Show)
    else StrPCopy (Bf, DestinationAddr^.Show);
     { Destination представлен в виде PChar }
    if Source <> nil
    then begin StrPCopy (Bf1, Representation); StrCat (Bf1, Source^.Show) end
    else begin StrPCopy (Bf1, Representation);
               StrCat (Bf1, SourceAddr^.Show)
         end;
     { Source представлен в виде PChar }
    StrCat (Bf, Bf1);
    Show := Bf
  end;
  procedure TBooleanAssignment.Run;
  begin
   if Destination  <> nil
   then begin
          Destination^.Run;
          DestinationValue := PbooleanValue (UV)end
   else if DestinationAddr <> nil
        then { Получатель - идентификатор в стеке }
           begin DestinationValue := PbooleanValue
                (DestinationAddr^.GetValue) end
        else Halt (1);
   if Source <> nil
  then begin Source^.Run; SourceValue := PbooleanValue (UV) end
   else if SourceAddr <> nil
        then { Источник - идентификатор в стеке }
           begin SourceValue := PbooleanValue (SourceAddr^.GetValue) end
        else Halt (1);
    if DestinationValue^.Scope <= SourceValue^.Scope
    then begin DestinationAddr^.PutValue (SourceValue);
               UV := SourceValue
         end
    else { область действия получателя старше области действия источника }
      begin writeln ('  НАРУШЕНИЕ ОБЛАСТЕЙ ДЕЙСТВИЯ В ПРИСВАИВНИИ ');
            Halt (1) end;
  end;
    { Обновление области действия имени логического }
    if is_loc
    then ref_to_bool^.Scope := CurrentLevel else ref_to_bool^.Scope := 0;
   { Выдача имени логического в виде указателя на контейнер логического }
    UV := ref_to_bool;
  end;
...
  {Присваивание}
  function TAssignment.Show : PChar;
  begin abstract end;
  procedure TAssignment.Run;
```

```
 begin abstract end;
...
 constructor TBooleanAssignment.Init (r : PChar; D, S : PConstruct;
                                      da, sa : PAddr);
 begin Representation := r;
       Destination := D;
       Source := S;
       DestinationAddr := da;
       SourceAddr := sa
 end;
 function TBooleanAssignment.Show : PChar;
 var Bf, Bf1 : array [0..512] of char;
 begin
  if Destination <> nil
  then { получатель задан в виде конструкции }
       StrPCopy (Bf, Destination^.Show)
  else { получатель задан в виде адреса его значения в стеке }
       StrPCopy (Bf, DestinationAddr^.Show);
  if Source <> nil
  then { источник задан в виде конструкции }
    begin StrPCopy(Bf1,Representation); StrCat (Bf1,Source^.Show) end
  else { источник задан в виде адреса его значения в стеке }
    begin StrPCopy(Bf1,Representation); StrCat(Bf1,SourceAddr^.Show) end;
  StrCat (Bf, Bf1); Show := Bf
 end;
 procedure TBooleanAssignment.Run;
 begin
   if Destination  <> nil
   then { Получатель - конструкция }
        begin Destination^.Run; DestinationValue := PbooleanValue (UV) end
   else if DestinationAddr <> nil
        then { Получатель - идентификатор в стеке }
          begin
            DestinationValue := PbooleanValue(DestinationAddr^.GetValue)
          end
        else { Получатель не задан }
             Halt(1);
   if Source <> nil
   then { Источник - конструкция }
        begin Source^.Run;
              SourceValue := PbooleanValue (UV)end
   else if SourceAddr <> nil
        then { Источник - идентификатор в стеке }
          begin SourceValue := PbooleanValue (SourceAddr^.GetValue) end
        else { Источник не задан }
             Halt(1);

  { Проверка условия на соотношение областей действия
    значений получателя и источника присваивания }
  if DestinationValue^.Scope <= SourceValue^.Scope
  then { правильное соотношение областей действия:
         область действия получателя не старше области действия источника }
    begin DestinationAddr^.PutValue (SourceValue); UV := SourceValue end
  else { неправильное соотношение областей действия:
         область действия получателя старше области действия источника }
    begin writeln ('   НАРУШЕНИЕ ОБЛАСТЕЙ ДЕЙСТВИЯ В ПРИСВАИВНИИ ');
          Halt(1)
    end;
end;
```

Заметим, что параметры конструктора **Init** присваивания служат для задания:
– её внешнего вида на входе конвертера в протоколе (**r**);
– конкретных конструкций получателя (**D**) и источника (**S**);
– адресов индикаторов значений получателя (**da**) и источника (**sa**).

Причём только два из параметров **D**, **S**, **da**, **sa** не равны **nil** (сравним с описанием стандартных формул [4]).

Напомним также, что, согласно соглашениям модели семантики в данном проекте, результат любой конструкции фиксируется в административной переменной **UV**. Что делать с этим результатом, «знает» метод **Run** использующей его конструкции.

Ниже приводится листинг построения семантического дерева вышеупомянутой программы, а также протокол его интерпретации.

Листинг II. Создание семантического дерева программы
.begin .bool b = .true; .ref.bool rb = .loc.bool; rb := b **.end**
и протокол её исполнения.

```
program eq_dec_test;
uses CRT, objects, Strings,
     VALUES, PLAIN_VALUES, STANDART, CONSTRUCTS, CLAUSES, DECLARATIONS,
     ENVIRON;
var bds, bool_default, r : string;
    bd : PBooleanDenotation;
    b, rb : PTag;
    TagList1, TagList2 : PTagList;
    bg : PBooleanGenerator;
    IdentityDeclaration1, IdentityDeclaration2 : PIdentityDeclaration;
    Addr00, Addr01 : PAddr;
    Assignment : PBooleanAssignment;
    cl0 : PConstructList;
    Range0 : PRange;
    main : PClosedClause;
begin
 ClrScr;
 writeln (#13#10'Тестирование программы Алгола 68:');
 writeln('.begin.bool b =.false; .ref.bool rb =.loc.bool; rb := b .end');
 writeln (#13#10'   ПРОСТРАНСТВО ДАННЫХ:' );
 { 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 }
 bool_default := '.loc.bool';
 bds := 'true';
 bd := New (PBooleanDenotation, Init (@bds[1], true));
 writeln ('   Изображение логического: ', bd^.Show);
 writeln ('   Создание Tag''a b ');
 b := New (PTag, Init (true, 'b', bd));
 writeln ('   Tag: ', b^.Show);
 bg := New (PBooleanGenerator, Init (@bool_default[1], true));
 writeln ('   Генератор логического: ', bg^.Show);
 writeln ('   Создание Tag''a rb ');
```

```
 rb := New (PTag, Init (true, 'rb', bg));
 writeln ('   Tag: ', rb^.Show);
 TagList1 := New (PTagList, Init (1, 0));
 TagList1^.Insert (b);
 writeln ('   Список Tag''ов создан: ', TagList1^.Show);
 TagList2 := New (PTagList, Init (1, 0));
 TagList2^.Insert (rb);
 writeln ('   Список Tag''ов создан: ', TagList2^.Show);
{ Создание списка конструкций описание тождества:
   .bool b = .true }
 IdentityDeclaration1 := New (PIdentityDeclaration,
                              Init ('.bool', TagList1));
 write (#13#10'   Конструкция описание тождества создана: ');
 writeln (IdentityDeclaration1^.Show); writeln;
 { Создание списка конструкций описание тождества:
   .ref.bool rb = .loc.bool }
 IdentityDeclaration2 := New (PIdentityDeclaration,
                              Init ('.ref.bool', TagList2));
 write ('   Конструкция описание тождества создана: ');
 writeln (IdentityDeclaration2^.Show);
 Addr00 := New (PAddr, Init (0, 0));
 writeln (#13#10'   Создан адрес Addr00: ', Addr00^.Show);
 Addr01 := New (PAddr, Init (0, 1));
 writeln ('   Создан адрес Addr01: ', Addr01^.Show);
 { Создание конструкции присваивание: rb := b }
 r := ' := ';
 Assignment := New (PBooleanAssignment,
                    Init (@r[1], nil, nil, Addr01, Addr00));
 writeln (#13#10'Конструкция присваивание создана: ', Assignment^.Show);
 { Создание списка конструкций блока 0 }
 cl0 := New (PConstructList, Init (3, 0));
 cl0^.Insert (IdentityDeclaration1);
 cl0^.Insert (IdentityDeclaration2);
 cl0^.Insert (Assignment);
 Range0 := New (PRange, Init (0, 10, cl0));
 main := New (PClosedClause, Init (Range0));
 writeln (#13#10'СЕМАНТИЧЕСКОЕ ДЕРЕВО ПРОГРАММЫ СОЗДАНО:', main^.Show);
 writeln (#13#10'ЗАПУСК ПРОГРАММЫ ... '#13#10 );
 main^.Run;
 writeln (#13#10'ПРОГРАММА ВЫПОЛНЕНА!')
end.
```

4. ИМЕНА ВЫСШИХ ПОРЯДКОВ (КОСВЕННЫЕ ИМЕНА)

Как уже не раз отмечалось, имена используются в присваиваниях, и *апостериорные* виды получателя и источника должны соответствовать схеме

reference to MODE := MODE (5)

Если их *априорные* виды не удовлетворяют условию (5), то исполняются приведения с учётом сортов позиций, так чтобы достичь указанного соответствия. Выстраивание плана приведений – задача конвертера, и исполнение соответствующих действий относится ко времени счёта.

Рассмотрим пример программы на Алголе 68 (**.bool b, b1; b := .true; b1 := b**), в которой описаны переменные **b** и **b1**. Эти переменные после исполнения их описаний

```
 Тестирование программы Алгола 68:
 .begin .bool b = .true; .ref.bool rb = .loc.bool; rb := b .end

ПРОСТРАНСТВО ДАННЫХ:

Стек на 1 участок
Display на 1 участок

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

Изображение логического: true
Создание Tag'а b
Tag: b = true
Генератор логического: .loc.bool
Создание Tag'а rb
Tag: rb = .loc.bool
Список Tag'ов создан: b = true
Список Tag'ов создан: rb = .loc.bool

Конструкция описание тождества создана: .bool b = true

Конструкция описание тождества создана: .ref.bool rb = .loc.bool

Создан адрес Addr00: < 0, 0 >
Создан адрес Addr01: < 0, 1 >

Конструкция присваивание создана: < 0, 1 > := < 0, 0 >

СЕМАНТИЧЕСКОЕ ДЕРЕВО ПРОГРАММЫ СОЗДАНО:
BEGIN(0)
  [0] .bool b = true
  [1] .ref.bool rb = .loc.bool
  [2] < 0, 1 > := < 0, 0 >
END(0)


ЗАПУСК ПРОГРАММЫ ...

ConstructList:
  [0] .bool b = true
  [1] .ref.bool rb = .loc.bool
  [2] < 0, 1 > := < 0, 0 >

Состояние стека после исполнения .bool b = true:

Display [0] ::
[0] b => true

Состояние стека после исполнения .ref.bool rb = .loc.bool:

Display [0] ::
[0] b => true
[1] rb => false

Состояние стека после исполнения < 0, 1 > := < 0, 0 >:

Display [0] ::
[0] b => true
[1] rb => true

ТЕКУЩЕЕ ОКРУЖЕНИЕ ПОСЛЕ ВЫХОДА ИЗ БЛОКА ТЕКУЩЕГО УРОВНЯ

ПУСТО

ПРОГРАММА ВЫПОЛНЕНА!
```

Рис. 2. Протокол исполнения программы
.begin .bool b = .true; .ref.bool rb = .loc.bool; rb := b **.end**
на Алголе 68

обладают значениями по умолчанию с учётом их видов. При этом идентификатор **b** обладает именем логического, которое именует значение **false** (по умолчанию) до исполнения присваивания **b := .true** и **true** – после его исполнения. Виды получателя и источника в этом присваивании соответствуют соотношению (5) при **MODE = boolean**, ибо идентификатор переменной **b** обладает значением вида **'reference to boolean'**, а значение изображения логического **.true** имеет вид **'boolean'**. Так что никаких приведений в этом присваивании не требуется.

Присваивание **b := b** требует разыменования источника, однако для этого никакие действия не нужны, поскольку в рассматриваемой модели семантики значения вида **'reference to MODE'**, где **MODE** не имя, ничем не отличается от представления значения вида **MODE**.

Однако в представлении имён высших порядков, например значений вида **'reference to reference to boolean'**, необходимо реализовать метод для разыменования, состоящий из непустых действий.

Пока без реализации рассмотрим следующий пример программы на Алголе 68

```
(.ref.bool b = .loc.bool;
 .ref.ref.bool bb  = .loc.ref.bool;
 .ref.ref.bool bb1 = .loc.ref.bool;
  b := .true; bb := b; bb1 := bb),
```

в которой описаны: переменная **b** как имя первого порядка вида **'reference to boolean'**, именующее простое логическое значение вида **'boolean'**, и переменные **bb**, **bb1**, причём идентификаторы переменных **bb**, **bb1** обладают значениями вида **'reference to reference to boolean'**, то есть косвенными именами второго порядка.

В первых двух присваиваниях виды получателей и источников согласованы, а в третьем – нет. Требуется разыменование источника этого присваивания.

На рис. 3 показана картина стека после исполнения описаний. После исполнения присваиваний значения **false** меняются на **true** во всех экземплярах объектов, о которых идёт речь.

Представление косвенных имён простых видов описываются в модуле **REF_PLAIN**, из которого приведём описания значений видов **'reference to reference to boolean'** и **'reference to reference to reference to boolean'** (см. листиг III).

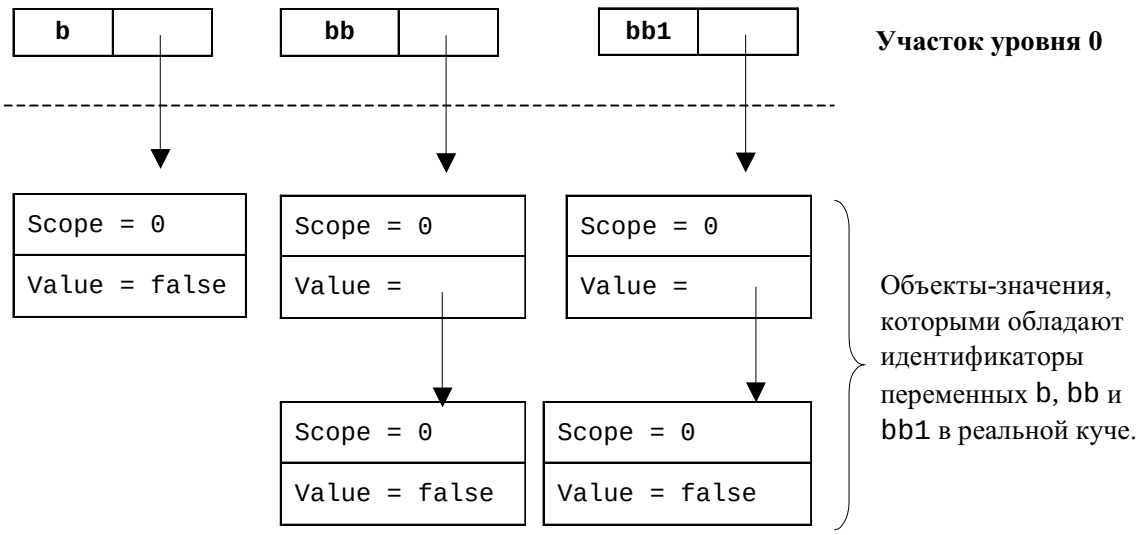

Рис. 3. Картина стека после исполнения описаний

Листинг III. Фрагмент модуля **REF_PLAIN**.

```
Unit REF_PLAIN;
{ Представление косвенных имён простых видов }
interface
uses objects, Strings, VALUES, PLAIN_VALUES, ENVIRON;
type
   { Значения вида ref_ref_bool }
     Pref_ref_bool = ^Tref_ref_bool;
     Tref_ref_bool = object (TValue)
     { Scope - унаследованное поле }
      Value : PBooleanValue;
      constructor Init (v : PBooleanValue);
      destructor Done; virtual;
      procedure deref; virtual;
     end;
  { Значения вида ref_ref_ref_bool }
    Pref_ref_ref_bool = ^Tref_ref_ref_bool;
    Tref_ref_ref_bool = object (TValue)
    { Scope - унаследованное поле }
      Value : Pref_ref_bool;
      constructor Init (v : Pref_ref_bool);
      destructor Done; virtual;
      procedure deref; virtual;
    end;
 ...
implementation
{ РЕАЛИЗАЦИЯ ЗНАЧЕНИЙ КОСВЕНЫХ ИМЁН ПРОСТЫХ ВИДОВ }
{ Значения вида ref_ref_bool }
  constructor Tref_ref_bool.Init (v : PBooleanValue);
  begin inherited Init (0); Value := v end;
  destructor Tref_ref_bool.Done;
  begin ... end;
  procedure Tref_ref_bool.deref;
  begin UV := Value end;
{ Значения вида ref_ref_ref_bool }
  constructor Tref_ref_ref_bool.Init (v : Pref_ref_bool);
  begin inherited Init (0); Value := v end;
  destructor Tref_ref_ref_bool.Done;
  begin ... end;
  procedure Tref_ref_ref_bool.deref;
  begin UV := Value end;
...
end.
```

С учётом этого приведём автономный тест **Name_test1** на создание и разыменование имён, о которых идёт речь (см. листинг IV), а также протокол его исполнения.

Тест автономный в том смысле, что в нём описывается процесс создания и разыменования значений указанного вида непосредственно без использования конструкций реализуемого языка, например генераторов. Это видно по списку модулей (uses), используемому в нём.

Листинг IV. Автономный тест на создание и разыменование имён.

```
program Name_test1;
{ ТЕСТИРОВАНИЕ ИМЁН }
uses CRT, objects, Strings,
     VALUES, PLAIN_VALUES, REF_PLAIN, ENVIRON;
var rrrb : Pref_ref_ref_bool;
    rrb, rrb1 : Pref_ref_bool;
    rb : PBooleanValue;
begin ClrScr;
 writeln 'ТЕСТИРОВАНИЕ ИМЁН ПРОСТЫХ ЗНАЧЕНИЙ 1-ГО ПОРЯДКА');
 writeln 'Создание значения rb');
 rb := New (PBooleanValue, Init (false));
 writeln 'rb - значение вида ''reference to boolean'' создано');
 writeln ('Разыменовывать rb надобности (см. НАПОМИНАНИЕ)');
 writeln ('ТЕСТИРОВАНИЕ ИМЁН ПРОСТЫХ ЗНАЧЕНИЙ 2-ГО ПОРЯДКА');
 writeln ('Создание rrb');
 rrb := New (Pref_ref_bool, Init (rb));
 writeln ('rrb - значение вида ''reference to reference to boolean''
           создано');
 writeln ('rrb^.Scope = ', rrb^.Scope);
 writeln ('Разыменование rrb ...');
 rrb^.deref;
 writeln ('Значение rrb разыменовано');
 writeln ('Эффект UV := rrb^.deref: ', UV^.Show); readln;
 writeln ('ТЕСТИРОВАНИЕ ИМЁН ПРОСТЫХ ЗНАЧЕНИЙ 3-ГО ПОРЯДКА');
 writeln ('Создание значения rrrb');
 rrrb := New (Pref_ref_ref_bool, Init (rrb));
 writeln ('rrrb - значение вида ''reference to reference to reference to
           boolean'' создано');
 writeln ('rrrb^.Scope = ', rrrb^.Scope);
 writeln ('Разыменование rrrb ...');
 rrrb^.deref;
 writeln ('Значение rrrb разыменовано');
 writeln ('Эффект UV := rrrb^.deref: значение вида ''reference to reference
           to boolean''');
 rrb1 := Pref_ref_bool (UV);
 writeln ('Эффект rrb1 := Pref_ref_bool (UV): значение вида ''reference to
           reference to boolean''');
 writeln ('Разыменование rrb1 ...');
 rrb1^.deref;
 writeln ('rrb1 разыменовано');
 writeln ('Эффект UV = rrb1^.deref: значение вида ''reference to  boolean''');
 writeln ('Значение UV =  ', UV^.Show);
 writeln ('НАПОМИНАНИЕ: значения вида ''reference to MODE'' и' +
            '''MODE'', где MODE - не имя, представляются одинаково!');
 end.
```

Что касается примера использования косвенных имён в присваиваниях, упомянутого выше, то это потребует пересмотра описания объекта-конструкции присваивания, а также введения метода *разыменования* имён, *приведения*, характерного именно для этого вида значений.

Разыменование не является единственным способом приведения видов. Существуют и другие, например обобщение от целого к вещественному, векторизация скалярного до

```
ТЕСТИРОВАНИЕ ИМЁН ПРОСТЫХ ЗНАЧЕНИЙ 1-ГО ПОРЯДКА
Создание значения rb
rb - значение вида 'reference to boolean' создано
Разыменовывать rb надобности (см. НАПОМИНАНИЕ)
ТЕСТИРОВАНИЕ ИМЁН ПРОСТЫХ ЗНАЧЕНИЙ 2-ГО ПОРЯДКА
Создание rrb
rrb - значение вида 'reference to reference to boolean' создано
rrb^.Scope = 0
Разыменование rrb ...
Значение rrb разыменовано
Эффект UU := rrb^.deref: false

ТЕСТИРОВАНИЕ ИМЁН ПРОСТЫХ ЗНАЧЕНИЙ 3-ГО ПОРЯДКА
Создание значения rrrb
rrrb - значение вида 'reference to reference to reference to boolean' создано
rrrb^.Scope = 0
Разыменование rrrb ...
Значение rrrb разыменовано
Эффект UU := rrrb^.deref: значение вида 'reference to reference to boolean'
Эффект rrb1 := Pref_ref_bool (UU):
       значение вида 'reference to reference to boolean'
Разыменование rrb1 ...
rrb1 разыменовано
Эффект UU = rrb1^.deref: значение вида 'reference to boolean'
Значение UU =  false
НАПОМИНАНИЕ: значения вида 'reference to MODE' и
             'MODE', где MODE - не имя, представляются одинакого!
```

Рис. 4. Протокол исполнения программы птестирования имен простых значений 1-го порядка

одномерного массива и т. д. Проблема приведений видов в общей постановке нуждается в специальном рассмотрении и составит тему следующей публикации. В ней будет разобран и упомянутый выше пример.

5. ЗАКЛЮЧЕНИЕ

Начиная серию статьей по всей этой обширной теме, я предполагал, что едва ли мне удастся сделать работу в «один проход», не пересматривая уже написанное, поскольку написание очередной статьи совместно с проектированием модели и экспериментами на компьютере с использованием системы Free Pascal [5] неизбежно чревато ошибками. Однако в такой ситуации оказывается любой разработчик большого проекта, ибо публикация в виде текущего отчёта незаконченного проекта необходима для других участников разработки. Желательно только, чтобы этот пересмотр не отменял уже сделанную работу.

Литература

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

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

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

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

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

Abstract

The modelling of the Algol 68 generators in the form of object-constructions, as well as results of their elaboration, i.e. values of names of appropriating types, is considered. In particular, the representing of indirect names is discussed. Their use based on examples is demonstrated.

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

© Наши авторы, 2009.
Our authors, 2009.