*Мартыненко Борис Константинович*

# УЧЕБНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПРОЕКТ РЕАЛИЗАЦИИ АЛГОРИТМИЧЕСКИХ ЯЗЫКОВ: ОПИСАНИЯ И ОКРУЖЕНИЯ

**Аннотация**

Рассматривается представление описаний простых видов в форме объектов-конструкций, а также окружений, реализуемых в виде иерархии участков в стеке, заполненных индикаторами, посредством которых программа получает доступ к значениям.

Описывается сам этот метод доступа.

**Ключевые слова:** окружение, последовательное предложение, стек, участок.

## 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 этот гипотетический вычислитель, отображаются в соответствующие классы объектов.

В [3] описаны основы и значения простых видов в качестве результатов исполнения этих конструкций, а также использование сцен в переходах.

Здесь же мы рассмотрим представление описаний в форме объектов-конструкций, а также окружений, реализуемых в виде иерархии участков стека, заполненных индикаторами. Эти индикаторы возникают в результате исполнения описаний и используются для доступа к значениям.

Основная проблема состоит в том, как организовать взаимодействие конструкций программы (синтаксис) и текущего окружения программы (семантика). Естественно использовать обычный приём ООП: передачу статической информации, извлекаемой из входного текста программы во время его анализа (синтаксис) в объект-конструкцию при его создании посредством *конструктора* (`Init`) этого объекта, с учётом этой информации в методе (`Run`), реализующем исполнение этой конструкции во время счёта (семантика). В интересующем нас случае речь идёт о статических координатах значения в стеке данных, которые представляются уровнем блока и относительным номером индикатора на участке, созданным этим блоком. Индикатор имеет поле, связывающее его со значением, которым он обладает.

Все эксперименты были проведены на компьютере с использованием системы программирования FREE PASCAL [4].

## 2. ОПИСАНИЯ ДАННЫХ И ИНДИКАТОРЫ

Согласно [2], данные любых видов определяются с помощью конструкции *описание тождества* (*см*. объект типа **TIdentityDeclaration** в модуле **DECLARATIONS**). Она связывает индикатор (*см*. объект типа **TTag**) с некоторой конструкцией, относящейся к группе основ, вид которой специфицируется соответствующим описателем. В расширенном варианте конструкции описание тождества может иметь несколько пар (индикатор – основа) (*см*. объект типа **TTagList**). Исполнение этой конструкции устанавливает связь индикатора со значением основы. Всякое применение индикатора в других конструкциях означает использование значения, связанного с ним по описанию тождества.

В терминах описываемой модели эта связь воспроизводится путём образования пар (индикатор – значение) на участке, образуемом блоком, в котором находится описание тождества. Доступ к этим значениям из других конструкций реализуется посредством *статических адресов* индикаторов, то есть пар ($l$, $n$), где $l$ – уровень блока, а $n$ – относительный номер *определяющего вхождения* индикатора в данном блоке.

Общее число индикаторов по всем описаниям блока определяет *статический аппетит* блока, то есть число элементов коллекции, представляющей его участок.

## 3. РЕАЛИЗАЦИЯ КОНСТРУКЦИИ *ОПИСАНИЕ ТОЖДЕСТВА*

Конструкция *описание тождества* в модельном представлении определяется в модуле

```
unit DECLARATIONS;
{ Реализация описаний }
interface
uses objects, Strings, VALUES, PLAIN_VALUES, ENVIRON, STANDART, CONSTRUCTS;
type
 { ВСПОМОГАТЕЛЬНЫЕ СИНТАКСИЧЕСКИЕ СТРУКТУРЫ }
 { Тег в описании тождества }
 PTag = ^TTag;
 TTag = object (TObject)
  identifier : string; { Идентификатор константы, переменной или
                         индикатор операции }
  Source : PConstruct; { Основа, конструкция, результат которой
                         связывается с данным индикатором }
  is_IdentityDeclaration : Boolean; { Флажок выбора символа '=' или ':='
                                      для представления аннотации}
  constructor Init (is_id: Boolean; id : string; c : PConstruct);
  function Show : PChar; virtual;
  procedure Run; virtual;
 end;
{ Список тегов в описаниях тождеств }
 PTagList = ^TTagList;
 TTagList = object (TCollection)
  procedure Run; virtual;
  function Show : PChar; virtual;
 end;
 { Расширенное описание тождеств }
 PIdentityDeclaration = ^TIdentityDeclaration;
 TIdentityDeclaration = object (TConstruct)
  mode : string;
```

```
  TagList : PTagList;
  constructor Init (md : string; tl : PTagList);
  procedure Run; virtual;
  function Show : PChar; virtual;
 end;
 implementation
 { ВСПОМОГАТЕЛЬНЫЕ СИНТАКСИЧЕСКИЕ СТРУКТУРЫ }
 { Теги в описаниях тождеств, в частности, описаниях переменных }
 constructor TTag.Init (is_id : Boolean; id : string; c : PConstruct);
 begin is_IdentityDeclaration := is_id; identifier := id; Source := c end;
 function TTag.Show : PChar;
 var Bf: array [0..511] of char;
 begin StrPCopy (Bf, identifier);
       if Source <> Nil
      then begin if is_IdentityDeclaration then StrCat(Bf,' = ')
                 else StrCat(Bf,':= ');
                 StrCat(Bf, Source^.Show) end;
       Show := Bf
 end;
 procedure TTag.Run;
 var Couple : PCouple;
 begin
  if Source <> Nil
  then begin Source^.Run;
             Couple := New (PCouple, Init (identifier, UV));
              Couple^.PutValue (UV)
       end
   else Couple := New (PCouple, Init (identifier, Nil));
   CurrentEnviron^.AddCouple (Couple);
 end;
   { Список тегов в описаниях тождеств }
 procedure TTagList.Run;
   procedure ExecItem (Item : PTag); far;
   begin Item^.Run end;
 begin ForEach (@ExecItem)end;
 function TTagList.Show : PChar;
 var Bf : array [0..511] of char; l : Longint;
   procedure PrintItem (Item : PTag); far;
   begin StrCat (Bf, Item^.Show); StrCat (Bf, ', ') end;
 begin Bf[0] := #0; ForEach (@PrintItem); l := StrLen (Bf)-2;
 if Bf[l] = ',' then Bf[l] := #0; Show := Bf
 end;
   { Расширенное описание тождеств }
 constructor TIdentityDeclaration.Init (md : string; tl : PTagList);
 begin mode := md; TagList:= tl end;
 procedure TIdentityDeclaration.Run;
 { Выкладывает пары (индикатор, указатель на значение основы тега)
   на участок текущего блока }
 begin TagList^.Run end;
 function TIdentityDeclaration.Show : PChar;
 var Bf, B : array [0..511] of char;
 begin StrPCopy (Bf, mode + ' ');
       B := TagList^.Show;
       StrCat (Bf,B);
       Show := Bf
 end;
end.
```

Заметим, что определение этого модуля не зависит от вида конструкции *описание тождества* по существу.

Рассмотрим простейший пример программы, использующей конструкцию *описание тождества*:

```
.begin .int one = 001, two = 02; one + two .end
```

Связь между *определяющими* вхождениями индикаторов **one** и **two** в этом описании тождества и их *использующими* вхождениями в формуле **one + two** устанавливается на стадии идентификации при контекстном анализе.

Планирование размещения индикаторов на участке блока – задача конвертера. Для этого в тексте программы достаточно информации: уровень блока определяется как число блоков, объемлющих данный, а номер определяющего вхождения индикатора в данном блоке устанавливается по тексту входной программы на стадии бесконтекстного синтаксического анализа.

Значения обоих операндов формулы **one + two** в момент их использования операцией ‘+’ находятся в участке блока уровня 0 под индексами 0 и 1 и доступны через указатели этих индикаторов.

Пары (0, 0) и (0, 1) – статические адреса индикаторов **one** и **two** соответственно. Они используются для доступа к значениям, которыми эти индикаторы обладают в текущий момент исполнения программы, то есть при исполнении упомянутой формулы.

Конструкция типа **TAddr**, как и другие средства создания и управления динамикой текущего окружения исполняемой программы, описываются в модуле **ENVIRON**.

## 4. РЕАЛИЗАЦИЯ ОКРУЖЕНИЙ

Модельное представление понятия окружения включает:

– стек участков (**Stack**),

– таблицу (**Dislpay**),

– четыре параметра, характеризующих его текущее состояние:

**CurrentLevel** – текущий блочный уровень,

**CurrentEnviron** – текущий участок стека,

**UCouple** – индикатор значения текущей конструкции,

**UV** – значения текущей конструкции.

Оно описывается в модуле

```
Unit ENVIRON;
{Реализация окружений}
interface
uses objects, Strings, Values;
{ Напоминание: элементы коллекций индексируются от 0 до
  значения поля Count-1 включительно.}
type
   { Представление индикатора }
PCouple = ^TCouple;
TCouple = object (TObject)
 Indicator : string; {Внешнее представление ИНДИКАТОРА во входной программе.}
 Value : PValue; { Указатель на объект-значение. }
 constructor Init (indic : String; val : PValue);
 { Инициализирует индикатор по параметру indic и
   указателю на объект-значение val.
 destructor Done; virtual;
 { ИНТИКАТОР уничтожается, но значение, которым он обладает, сохраняется.}
```

```
 function Show : PChar; virtual;
 { Даёт строку, представляющий данный ИНДИКАТОР.
   Эта строка содержит внешнее представление индикатора и значение,
   которым он обладает.
   Если данная пара представляет переменную, то за
   идентификатором этой переменной в сформированной
   строке следует текущее значение переменной, а не имя
   (в смысле Алгола 68), которым она обладает.}
 function GetValue : PValue; virtual;
 { Даёт указатель на объект-значение, представленного данной парой.}
 function GetIndicator: string; virtual;
 { Даёт внешнее представление индикатора.}
 procedure PutValue (v : PValue); virtual;
 { Помещает указатель на объект-значение v в данную пару.}
end;
   { Представление участка стека (locale) }
PLocale = ^TLocale;
TLocale = object (TCollection)
 Name : string;
 RangeLevel : integer; { Блочный уровень (>= 0) блока,
                         образовавшего данный участок.}
 Prev: PLocale; { Указатель на участок объемлющего блока.}
 constructor Init (lev, max, inc : integer);
 { Инициализация участка блочного уровня lev.}
 destructor Done; virtual;
 { Участок уничтожается вместе со значениями, размещёнными на нём.}
 procedure AddCouple (Couple : PCouple);
 function CoupleNmb : integer;
 function Show : PChar; virtual;
end;
  { Представление стека участков }
PStack = ^TStack;
TStack = object (TCollection)
 constructor Init (NmbOfLocales : integer);
 { Создание стека данных первоначально на NmbOfLocales участков
   с последующим расширением на половину этой величины.}
 destructor Done; virtual;
 { Уничтожение всех участков.}
 procedure AddLocale (Locale : PLocale);
 { Поместить участок Locale на вершину стека.}
 procedure RemoveTopLocale;
 { Удалить участок с вершины стека.}
 procedure RemoveLocale (lr : PLocale);
 { Удалить участок с вершины стека пока его указатель не равен lr.
   Используется при исполнении перехода.
   Display корректируется соответственно.
   Здесь lr характеризует сцену метки.}
 function Show : PChar; virtual;
end;
   { Таблица Dislpay - текущее окружение }
PDisplay = ^TDisplay;
TDisplay = object (TCollection)
 constructor Init (Max : integer);
 { Создание таблицы Display на Max элементов.}
 destructor Done; virtual;
 procedure Fix (Locale : PLocale);
```

```
 { Фиксирует в таблице Display участок, созданный блоком,
   в качестве нового текущего окружения.}
 procedure UpDate (LocaleReference : PLocale);
 { Обновление таблицы Display по ссылке на участок,
   заданный параметром lr }
 function Show : PChar; virtual;
 { В виде строки представляет текущее окружение }
end;
    { Статический адрес значения в стеке данных }
PAddr = ^TAddr;
TAddr = object (TObject)
 RLevelNmb, { Блочный уровень блока, к которому относится данный адрес.}
 ItemNmb { Номер ИНДИКАТОРА на участке уровня RLevelNmb.}: integer;
 constructor Init (l, n : integer);
 { Инициализирует:
   (a) номер блочного уровеня RLevelNmb по параметру l и
   (b) номер ИНДИКАТОРА на участке уровня RLevelNmb
       в стеке данных по параметру n.}
 function Show : PChar; virtual;
 { Готовит строку, представляющую данный статический адрес.}
 function GetCouple : PCouple;
 { По статическому адресу ИНДИКАТОРА даёт
   указатель на него в стеке данных.}
 function GetIndic : string;
 { По статическому адресу элемента стека данных дает
   внешнее представление индикатора - первую компоненту пары.}
 function GetValue : PValue; virtual;
   { По статическому адресу ИНДИКАТОРА дает
     указатель на значение - вторую компоненту пары.}
 procedure PutValue (v : PValue); virtual;
 { Указатель v вставляет в стек данных на место,
   задаваемое данным статическим адресом,
   не меняя внешнего представления индикатора получателя.}
 function GetScope : integer; virtual;
 { Даёт область действия значения,
   размещенного по данному адресу в стеке данных.}
end;

var
 Stack : PStack;           { Стек участков }
 Display : PDisplay;       { Таблица Display }
 CurrentLevel : integer;   { Текущий блочный уровень }
 CurrentEnviron : PLocale; { Текущее окружение }
 UCouple : PCouple; { Индикатор значения текущей конструкции }
 UV : PValue; { Универсальное значение любого вида -
                собственно значение текущей конструкции }
 Jump_elaborated : Boolean; { Флажок прерывание исполнения основ
                              текущего блока.
                             Используется при исполнении переходов. }
implementation
   { РЕАЛИЗАЦИЯ TStack }
constructor TStack.Init (NmbOfLocales : integer);
{ Создание стека данных первоначально на NmbOfLocales участков
  с последующим расширением на половину этой величины }
  begin inherited Init (NmbOfLocales, NmbOfLocales div 2 + 1) end;
destructor TStack.Done;
begin while count > 0 do
```

```
      begin count := count - 1; Free (At (count))end;
       DeleteAll;
       inherited Done
end;
procedure TStack.AddLocale (Locale : PLocale);
{ Добавление нового участка на вершину стека данных }
var CE : PLocale;
  begin
    CE := CurrentEnviron;
    Insert (Locale);
    CurrentEnviron := At (count-1);
    CurrentEnviron^.Prev := CE;
    CurrentLevel:= CurrentLevel + 1;
     CurrentEnviron^.RangeLevel := CurrentLevel;
     Display^.Fix (CurrentEnviron)
  end;
   procedure TStack.RemoveTopLocale;
  { Удалить участок с вершины стека }
  begin
     if count > 0
     then
     begin
       Free (At (count-1));
      Display^.count := CurrentLevel;
      CurrentLevel := CurrentLevel - 1;
       if CurrentLevel < 0
       then CurrentEnviron := nil
       else CurrentEnviron := CurrentEnviron^.Prev
     end
  end;
   procedure TStack.RemoveLocale (lr : PLocale);
  { Удалить участок с вершины стека пока его указатель не равен lr.
    Используется при исполнении перехода.
    Display корректируется соответственно.
    Здесь lr характеризует сцену метки. }
  begin while CurrentEnviron <> lr do RemoveTopLocale end;
  function  TStack.Show : PChar;
  var s : string; i : integer;
  Bf : array [0..400] of char; B : array [0..127] of char;
    procedure PrintItem (Item: PLocale); far;
    begin inc (i); str (i, s); StrPCopy (B, s);
        {s := s + #10#13'Stack [' + s1 + '] = ' + L^.Show}
        StrCat (Bf, #10#13'   Stack ['); StrCat (Bf, B);
        StrCat (Bf, '] = '); StrCat (Bf, Item^.Show)
   end;
  begin i := -1; Bf[0] := #0; ForEach (@PrintItem);
        if Bf[0] = #0 then  StrPCopy (Bf, '   EMPTY'); Show := Bf end;
     { РЕАЛИЗАЦИЯ TDislpay }
   constructor TDisplay.Init (Max : integer);
  { Сознание таблицы Display на Max элементов }
  begin inherited Init (Max, 0) end;
  function  TDisplay.Show : PChar;
  var s : string; i : integer;
      Bf : array [0..4095] of char; B : array [0..4095] of char;
      procedure PrintItem (Item: PLocale); far;
     begin
        inc (i); str (i, s); StrPCopy (B, s);
```

```
      {s := s + #10#13'Display [' + s1 + '] :: ' + L^.Show }
      StrCat (Bf, #10#13'   Display ['); StrCat (Bf, B);
      StrCat (Bf, '] :: '); StrCat (Bf, Item^.Show)
   end;
begin i := -1; Bf[0] := #0; ForEach (@PrintItem);
      if Bf[0] = #0 then StrPCopy (Bf, #13#10'   ПУСТО'); Show := Bf
end;
procedure TDisplay.Fix (Locale : PLocale);
{ Фиксирует в таблице Display участок, созданный блоком,
  в качестве нового текущего окружения. }
var cl : integer;
begin
  cl := Locale^.RangeLevel;
  Insert (Locale);
  CurrentLevel := cl; CurrentEnviron := Locale
end;
procedure TDisplay.UpDate (LocaleReference : PLocale);
var cv : integer; e : PLocale; { Параметр цикла продвижения по Display.}
begin
  CurrentEnviron := LocaleReference; { Установка текущего окружения
                                       по параметру процедуры.}
  CurrentLevel := CurrentEnviron^.RangeLevel; { Определение уровня
                                                нового окружения.}
  count := CurrentLevel+1;
  e := CurrentEnviron;
  cv := CurrentLevel;
  while cv > 0 do
  begin Display^.AtPut (cv-1, e^.Prev);
          e := e^.Prev; cv := cv - 1
  end
end;
destructor TDisplay.Done;
begin inherited Done end;
   { РЕАЛИЗАЦИЯ TLocale }
constructor TLocale.Init (lev, max, inc : integer);
{ Создание нового участка блочного уравня lev >= 0
  на max элементов и приращением на inc элементов }
begin
  RangeLevel := lev ; Prev := CurrentEnviron;
  inherited Init (max, inc)
end;
destructor TLocale.Done;
begin DeleteAll end;
function TLocale.CoupleNmb : integer;
begin CoupleNmb := count end;
procedure TLocale.AddCouple (Couple : PCouple);
begin Insert (Couple) end;
function  TLocale.Show : PChar;
var s : string; n : integer;
    Bf : array [0..4095] of char; B : array [0..511] of char;
  procedure PrintItem (Item: PCouple); far;
  begin inc (n); str (n, s); StrPCopy (B, s);
      StrCat (Bf, #10#13'   ['); StrCat (Bf, B); StrCat (Bf, '] ');
      StrCat (Bf, Item^.Show)
  end;
begin n := -1; Bf[0] := #0; ForEach (@PrintItem); Show := Bf end;
  { РЕАЛИЗАЦИЯ TCouple }
```

```
  constructor TCouple.Init (indic : String; val : PValue);
  begin Indicator := indic; Value:= val end;

  destructor TCouple.Done;
  begin Dispose (Value, Done)end;
  function TCouple.GetValue : PValue;
  begin GetValue := Value end;
  function TCouple.GetIndicator : string;
  begin GetIndicator := INDICATOR end;
  procedure TCouple.PutValue (v : PValue);
  begin Value := v end;
  function TCouple.Show : PChar;
  var Bf : array [0..127] of char;
  begin StrPCopy (Bf, INDICATOR); StrCat(Bf,' => ');
    if Value = nil then StrCat(Bf, 'nil')
                   else StrCat(Bf, Value^.Show);
      Show := Bf
  end;
     { РЕАЛИЗАЦИЯ TAddr }
constructor TAddr.Init (l, n : integer);
begin RLevelNmb := l; ItemNmb := n end;
function TAddr.Show : PChar;
var s, s1, s2 : string;
     s1x, s2x : array [0..127] of char;
     Bf : array [0..63] of char;
begin str (RLevelNmb, s1); s1 := s1 + ', ';
      str (ItemNmb, s2);
      StrPCopy (s1x, s1);  StrPCopy (s2x, s2);
      StrPCopy (Bf, '< '); StrCat (Bf, s1x);
      StrCat (Bf, s2x);  StrCat (Bf, ' >');
      Show := Bf
end;
function TAddr.GetCouple : PCouple;
{ По статическому адресу элемента стека данных
  дает указатель на пару в стеке данных.}
begin GetCouple := PLocale(Display^.At(RLevelNmb))^.At(ItemNmb) end;
function TAddr.GetIndic : string;
{ По статическому адресу элемента стека данных дает
  внешнее представление индикатора - первую компоненту пары.}
var Couple : PCouple;
begin Couple := GetCouple; GetIndic := Couple^.GetIndicator end;
function TAddr.GetValue : PValue;
{ По статическому адресу элемента стека данных дает
  указатель на значение - вторую компоненту пары.}
var Couple : PCouple;
begin Couple := GetCouple; GetValue := Couple^.GetValue end;
procedure TAddr.PutValue (v : PValue);
{ Указатель v вставляет в стек данных на место,
  задаваемое данным статическим адресом,
  не меняя внешнего представления индикатора получателя.}
var Couple : PCouple; { Пара-получатель }
begin Couple := GetCouple; Couple^.PutValue (v) end;
function TAddr.GetScope : integer;
begin GetScope := RLevelNmb end;
begin CurrentLevel := -1; CurrentEnviron := nil; UCouple := nil end.
```

Статические адреса индикаторов используются для инициализации объекта-конструкции типа **TAddr**, который при её исполнении преобразует статический адрес значения в стеке в указатель на собственно значение в реальной куче. Фактически, тип **TAddr** представляет конструкцию *применённое вхождение индикатора*. Семантически она открывает доступ к значению, которым обладает соответствующий индикатор для той конструкции, в которой он используется.

А теперь самое время вспомнить о способе передачи статической информации в объект-конструкцию при создании посредством *конструктора* **Init** и использовании этой информации в методе **Run**, реализующем исполнение этой конструкции.

Рассмотрим использование статических адресов индикаторов для доступа к значениям, которыми они обладают, в качестве значений операндов формул.

Заметим, что в [3] описан способ передачи значений операндов в формулу, который метафорически уместно назвать «из рук в руки», поскольку результат конструкции-операнда передаётся в конструкцию-формулу через локальные поля объектов **Loperand** и **Roperand**, минуя стек.

Если в программе используются описания тождеств, то способ обмена значениями «из рук в руки» не достаточен и придётся чуть изменить описание формул, а именно, ввести статические адреса операндов **LAddr**, **RAddr**.

```
PFormula = ^Tformula;
TFormula = object (TConstruct)
{ Наследуемые поля данных:
  Representation :  PChar ; }
  Loperand, Roperand : PConstruct { конструкции-операнды };
  LAddr, RAddr : PAddr { статические адреса значений операндов в стеке };
  function Show : PChar; virtual;
  procedure Run; virtual;
end;
```

с реализацией методов

```
function TFormula.Show : PChar;
begin abstract end;
procedure TFormula.Run;
begin abstract end;
```

При создании конкретного экземпляра формулы её конструктор (**Init**) получает обе пары ссылок на операнды, то есть ссылки на конструкции в роли операндов формулы и в виде статических адресов значений операндов в стеке данных.

Метод (**Run**), реализующий исполнение формулы, сам анализирует фактическую ситуацию: формула унарная или бинарная и какой способ доступа к значениям операндов применять.

Для примера, покажем описание формул с операциями вида **(integral, integral) integral**.

```
constructor Tintegral_integral_integral_Formula.Init
              (r : PChar; { Внешнее обозначение операции }
              op : Tintegral_integral_integral_Routine;
              { Стандартная операция, реализующая формулу }
               lo, ro : PConstruct;
              { Конструкции в роли операндов формулы }
              la, ra : PAddr { Статические адреса значений операндов } );
begin
   Representation := r;
   Routine := op;
   if lo <> nil
```

```
    then LOperand := lo
    else if la <> nil then LAddr := la;
    if ro <> nil
    then ROperand := ro
    else RAddr := ra;
end;
function Tintegral_integral_integral_Formula.Show : PChar;
var Bf, Bf1 : array [0..512] of char;
begin if LOperand <> nil then strPCopy (Bf, LOperand^.Show);
       if ROperand <> nil
       then begin StrPCopy (Bf1, Representation);
                  StrCat (Bf1, ROperand^.Show); StrCat (Bf, Bf1)
            end;
{ Если операнд формулы представлен статическим адресом, то при создании кон-
струкции «формула» стеком пользоваться нельзя, ибо он ещё не существует!}
        if LAddr <> nil
        then begin StrPCopy (Bf, LAddr^.Show);
                   if RAddr <> nil
                   then begin StrPCopy (Bf1, Representation);
                              StrCat (Bf1, RAddr^.Show); StrCat (Bf, Bf1)
                        end
             end;
  Show := Bf
end;
procedure Tintegral_integral_integral_Formula.Run;
begin if LOperand <> nil
       then begin LOperand^.Run;
                  LOperandValue := PIntegralValue (UV)
            end;
       if ROperand <> nil
       then begin ROperand^.Run;
                  ROperandValue := PIntegralValue (UV)
            end;
       if LAddr <> nil
        then begin LAddr^.GetValue; LOperandValue := PIntegralValue (UV) end;
       if RAddr <> nil
        then begin RAddr^.GetValue; ROperandValue := PIntegralValue (UV) end;
        UV := Routine (LOperandValue, ROperandValue)
end;
```

Здесь учитывается число операндов формулы и какой метод используется для доступа к их значениям. Если адрес **LOperand** или **ROperand** не равны **nil**, то используется метод «из рук в руки», а иначе значения операндов формулы достаются из стека данных с использованием статических адресов.

Демонстрация вышесказанного приведена в листинге I и на рис. 1.

В этом примере доступ к значениям операндов формулы **one + two** производится с использованием статических адресов (0, 0) и (0, 1), поскольку они соответствуют нулевому и первому вхождению индикаторов на участке уровня 0. Эти адреса преобразуются в указатели на значения, которыми обладают индикаторы **one** и **two** (*см.* метод **TAddr.GetValue** в модуле **ENVIRON**). Значение формулы вычисляется, но нигде не используется.

## 5. ПРИСВАИВАНИЕ

Рассмотрим ещё один пример, в котором показывается, как значение формулы замещает прежнее значение в стеке. Однако для этого потребуется описать реализацию ещё одной конструкции, а именно, *присваивания*.

**Листинг I.** Программа построения семантического дерева входной программы на Алголе 68: **.begin .int one = 001, two = 02; one + two .end** и её интерпретации

```
program  Itentity_Declaration_test;
 { Тестирование программы Алгола 68: .begin .int one = 001, two = 02; one + two .end }
 uses CRT, objects, Strings,
      VALUES, PLAIN_VALUES, STANDART, CONSTRUCTS, CLAUSES, ENVIRON, DECLARATIONS;
 var id1s, id2s, ones, twos, Addr00s, Addr01s, fs : string;
     id1, id2 : PIntegralDenotation;    cl0 : PConstructList;
     Range0 : PRange;     one, two : PTag;    Addr00,  Addr01 : PAddr;
     TagList : PTagList;  IdentityDeclaration : PIdentityDeclaration;
     Routine : Tintegral_integral_integral_Routine;
     f : Pintegral_integral_integral_Formula;
     main : PClosedClause;    var0, var1 : PVariable;
 begin ClrScr;
   writeln (#13#10'     Тестирование программы Алгола 68:');
   writeln ('     .begin .int one = 001, two = 02; one + two .end'#13#10);
   writeln ('   ПРОСТРАНСТВО ДАННЫХ:'#10#13);
 { СОЗДАНИЕ СТЕКА ДАННЫХ }
   Stack := New (PStack, Init (1));
   writeln ('   Стек на ', Stack^.Limit, ' участка');
 { СОЗДАНИЕ ТАБЛИЦЫ DISPLAY }
   Display := New (PDisplay, Init (1));
   writeln ('   Display на ', Display^.Limit,' участка'#13#10);
   writeln ('======= СОЗДАНИЕ СЕМАНТИЧЕСКОГО ДЕРЕВА ПРОГРАММЫ =======');
 { СОЗДАНИЕ КОНСТРУКЦИЙ ПРОГРАММЫ }
 id1s := '001';   id1 := New (PIntegralDenotation, Init (@id1s[1], 1));
 writeln ('   Изображение целого ', id1^.Show);
 writeln ('   Создание Tag''a one ');
 one := New (PTag, Init (true, 'one', id1));
 writeln ('   Tag ', one^.Show);
 id2s := '02';  id2 := New (PIntegralDenotation, Init (@id2s[1], 2));
 writeln ('   Изображение целого ', id2^.Show);
 writeln ('   Создание Tag''a two ');
 two := New (PTag, Init (true, 'two', id2));
 writeln ('   Tag ', two^.Show);
 { Создание списка тегов }
 writeln ('   Создание списка Tag''ов ...');
 TagList := New (PTagList, Init (2, 0)); TagList^.Insert (one); TagList^.Insert (two);
 writeln ('   Список Tag''ов создан: ', TagList^.Show); writeln;
 { Создание конструкции описание тождества: .int one = 001, two = 02  }
 IdentityDeclaration := New (PIdentityDeclaration, Init ('.int', TagList));
 writeln ('   Конструкция описание тождества создана: '); writeln;
 writeln ('    ', IdentityDeclaration^.Show); writeln;
 Routine := @PlusRoutine;  fs := ' + ';
 Addr00 := New (PAddr, Init (0, 0));
 writeln ('   Создан адрес Addr00: ', Addr00^.Show);
 Addr01 := New (PAddr, Init (0, 1));
 writeln ('   Создан адрес Addr01: ', Addr01^.Show);
 { Создание формулы с операцией вида (.int,.int).int }
 f := New (Pintegral_integral_integral_Formula,
           Init (@fs[1], Routine, NIL, NIL, Addr00, Addr01));
 writeln ('   Создана формула вида (integral,integral)integral: ', f^.Show);
 { Создание списка конструкций блока 0 }
 cl0 := New (PConstructList, Init (2, 0));
 cl0^.Insert (IdentityDeclaration); { Описание тождества }
 cl0^.Insert (f);                   { Формула }
```

```
 { Создание последовательного предложения - блока уровня 0 }
 Range0 := New (PRange, Init (0, 10, cl0));
 { Создание замкнутого предложения - собственно программы }
 main := New (PClosedClause, Init (Range0));
 writeln (#13#10'    СЕМАНТИЧЕСКОЕ ДЕРЕВО ПРОГРАММЫ СОЗДАНО:', main^.Show);
 writeln (#13#10'    Программа начала ... ');
 main^.Run;
 writeln ('    Программа выполнена!');
 writeln ('    UV = ', UV^.Show); writeln ('    СТОП !!!'); readln
end.
```

```
  Тестирование программы Алгола 68:
  .begin .int one = 001, two = 02; one + two .end

ПРОСТРАНСТВО ДАННЫХ:

Стек на 1 участка
Display на 1 участка

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

Изображение целого 001
Создание Tag'а one
Tag one = 001
Изображение целого 02
Создание Tag'а two
Tag two = 02
Создание списка Tag'ов ...
Список Tag'ов создан: one = 001, two = 02

Конструкция описание тождества создана:

.int one = 001, two = 02

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

constructor Tintegral_integral_integral_Formula.Init начал ...
Representation =  +
LAddr = < 0, 0 >
RAddr = < 0, 1 >
constructor Tintegral_integral_integral_Formula.Init закончил!

Создана формула вида (integral,integral)integral: < 0, 0 > + < 0, 1 >

СЕМАНТИЧЕСКОЕ ДЕРЕВО ПРОГРАММЫ СОЗДАНО:
BEGIN(0)
  [0] .int one = 001, two = 02
  [1] < 0, 0 > + < 0, 1 >
END(0)

Программа начала ...
ConstructList:
  [0] .int one = 001, two = 02
  [1] < 0, 0 > + < 0, 1 >

ТЕКУЩЕЕ ОКРУЖЕНИЕ ПОСЛЕ ИСПОЛНЕНИЯ TTag.Run
Display [0] ::
[0] one => 1

ТЕКУЩЕЕ ОКРУЖЕНИЕ ПОСЛЕ ИСПОЛНЕНИЯ TTag.Run
Display [0] ::
[0] one => 1
[1] two => 2

ТЕКУЩЕЕ ОКРУЖЕНИЕ ПОСЛЕ ВЫХОДА ИЗ БЛОКА ТЕКУЩЕГО УРОВНЯ
ПУСТО
Программа кончила!
UV = 3
СТОП !!!
```

**Рис. 1.** Протокол исполнения программы на Алголе 68:
`.begin .int one = 001, two = 02; one + two .end`

Эффект исполнения присваивания состоит в замещении значения получателя (**Destination**) значением источника (**Source**). Реализация этой конструкция во многом аналогична реализации формул.

Конструкция *присваивание* представляется как объект типа **TAssignment**:

```
PAssignment = ^TAssignment;
TAssignment = object (TConstruct)
(* Наследуемые поля данных:
 Representation : PChar; *)
 Destination { Аналог LOperand в формулах },
 Source { Аналог ROperand в формулах } : PConstruct;
 DestinationAddr { Аналог LAddr в формулах },
 SourceAddr { Аналог RAddr в формулах } : PAddr;
 function Show : PChar; virtual;
 procedure Run; virtual;
end;
```

от которого наследуются присваивания конкретных видов, в частности, логического вида:

```
PBooleanAssignment = ^TBooleanAssignment;
TBooleanAssignment = object (TAssignment)
(* Наследуемые поля данных:
 Representation : PChar;
 Destination, Source: PConstruct;
 DestinationAddr, SourceAddr: PAddr;
*)
 DestinationValue, SourceValue : PBooleanValue;
 constructor Init (r : PChar; D, S : PConstruct; da, sa : PAddr);
 function Show : PChar; virtual;
 procedure Run; virtual;
end;
```

Реализации абстрактных присваиваний в разделе **implementation** модуля **CONSTRUCTS** имеет вид:

```
function TAssignment.Show : PChar;
begin abstract end;
procedure TAssignment.Run;
begin abstract end;
```

а конкретных вида **boolean**:

```
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;
 if Source <> nil
 then begin Source^.Run;
            SourceValue := PbooleanValue (UV) end;
 if DestinationAddr <> nil
 then begin DestinationAddr^.GetValue;
            DestinationValue := PbooleanValue (UV) end;
 if SourceAddr <> nil
 then begin SourceAddr^.GetValue;
            SourceValue := PbooleanValue (UV) end;
 if DestinationAddr <> nil
 then DestinationAddr^.PutValue (SourceValue);
 if DestinationValue^.Scope >= SourceValue^.Scope
 then UV := DestinationValue
 else { Недопустимое отношение областей действия получателя и источника }
        Halt (1);
end;
```

И вот обещанный пример (см. листинг II, рис. 2).

Поясним дополнительно, что конструкция описания тождества **.bool b** в этой программе определяет логическую переменную. Это выясняется на этапе синтаксического анализа программы с учётом расширенного варианта грамматики языка Алгол 68. При этом пропущенная основа логического вида после идентификатора **b** в описании тождества воспринимается как генератор, исполнение которого даёт имя неопределённого логического значения. Поскольку из двух возможных значений **false** или **true**, согласно семантике языка Алгол 68, можно выбрать любое, то в нашей реализации выбрано **false**, в соответствии с бытующей традицией реализации алгоритмических языков (см. Листинг II).

Генераторы и имена – значения, выдаваемые этими конструкциями, – тема следующей публикации.

## 5. ЗАКЛЮЧЕНИЕ

Подведём краткие итоги.

1. Описания дают ещё один способ передачи значений между конструкциями в программе. При этом стек играет роль канала связи. Не будь описаний, не возможен был бы обмен значениями между конструкциями иначе как «из рук в руки». Рекурсивные процедуры также не могли бы существовать.

2. Средством передачи значений между конструкциями через стек служат статические адреса индикаторов. Они применяются при создании конструкций, использующих значения в стеке. Фактически, тип **TAddr** представляет конструкцию *применённое вхождение индикатора*. Семантически она открывает доступ к значению, которым обладает соответствующий индикатор для той конструкции, в которой он используется.

**Листинг II.** Программа построения семантического дерева входной программы **.begin .bool b; b := ~ b .end** и её интерпретации

```
program Itentity_Declaration2xx;
{ Тестирование программы Алгола 68: .begin .bool b; b := ~ b .end }
uses CRT, objects, Strings,
     VALUES, PLAIN_VALUES, STANDART, CONSTRUCTS,
     CLAUSES, ENVIRON, DECLARATIONS;
var bgs, Addr00s, fs : string;
    bg : PBooleanDenotation;  cl0 : PConstructList;
    Range0 : PRange;  b : PTag;  Addr00 : PAddr;
    TagList : PTagList;
    IdentityDeclaration : PIdentityDeclaration;
    Routine : Tboolean_boolean_Routine;
    f : Pboolean_boolean_Formula;
    Assignment : PAssignment;
    main : PClosedClause;
begin ClrScr;
      writeln (#13#10'     Тестирование программы Алгола 68:');
      writeln ('     .begin .bool b; ~ b .end'#13#10);
      writeln ('    ПРОСТРАНСТВО ДАННЫХ:'#10#13);
 { СОЗДАНИЕ СТЕКА ДАННЫХ }
   Stack := New (PStack, Init (1));
   writeln ('   Стек на ', Stack^.Limit, ' участка');
 { СОЗДАНИЕ ТАБЛИЦЫ DISPLAY }
   Display := New (PDisplay, Init (1));
   writeln ('   Display на ', Display^.Limit,' участка'#13#10);
   writeln ('======= СОЗДАНИЕ СЕМАНТИЧЕСКОГО ДЕРЕВА ПРОГРАММЫ ======'#13#10);
 { СОЗДАНИЕ СЕМАНТИЧЕСКОГО ДЕРЕВА ПРОГРАММЫ }
 writeln ('   Создание Tag''a b ');
 bgs := 'false';
 { Создание конструкции генератор логического .loc.bool }
 bg := New (PBooleanGenerator, Init (@bgs[1]));
 { Создание тега 'b = .loc.bool' }
 b := New (PTag, Init (false, 'b', bg));
 writeln ('   Tag ', b^.Show);
 { Создание списка тегов }
 writeln ('   Создание списка Tag''ов ...');
 TagList := New (PTagList, Init (1, 0));
 TagList^.Insert (b);
 writeln ('   Список Tag''ов создан: ', TagList^.Show); writeln;
 { Создание конструкции описание тождества '.bool b := false' }
 IdentityDeclaration := New (PIdentityDeclaration, Init ('.bool', TagList));
 writeln ('   Конструкция описание тождества создана: '); writeln;
 writeln ('    ', IdentityDeclaration^.Show); writeln;
 { Создание стандартной операции вида .op (.bool).bool  ~ = @NotRoutine }
 Routine := @NotRoutine;
 fs := ' ~ ';
 { Создание статического адреса переменной b }
 Addr00 := New (PAddr, Init (0, 0));
 writeln ('   Создан адрес Addr00: ', Addr00^.Show);
 { Создание логической формулы ~ b }
 f := New (Pboolean_boolean_Formula,
            Init (@fs[1], Routine, NIL, NIL, NIL, Addr00));
 writeln (#13#10'   Создана формула вида (boolean)boolean: ', f^.Show);
 { Создание присваивания b := ~ b }
```

```
Assignment := New  (PBooleanAssignment, Init (':=', nil, f, Addr00, nil));
{ Создание списка конструкций блока 0 }
cl0 := New (PConstructList, Init (2, 0));
 cl0^.Insert  (IdentityDeclaration);
cl0^.Insert  (Assignment);
{ Создание последовательного предложения - блока уровня 0:
  .bool b; b := ~ b }
Range0 := New (PRange, Init (0, 10, cl0));
{ Создание замкнутого предложения - собственно программы:
  .begin .bool b; b := ~ b .end }
main := New (PClosedClause, Init (Range0));
writeln (#13#10'    СЕМАНТИЧЕСКОЕ ДЕРЕВО ПРОГРАММЫ СОЗДАНО:', main^.Show);
writeln (#13#10'    Программа начала ... '#13#10);
{ ЗАПУСК ИНТЕРПРЕТАЦИИ СЕМАНТИЧЕСКОГО ДЕРЕВА ПРОГРАММЫ }
main^.Run;
writeln (#13#10'    Программа кончила!');
writeln (#13#10'    СТОП !!!'); readln
end.
```

```
 Тестирование программы Алгола 68:
 .begin .bool b; b := ~ b .end

ПРОСТРАНСТВО ДАННЫХ:

Стек на 1 участка
Display на 1 участка

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

Создание Tag'а b
Tag b:= false
Создание списка Tag'ов ...
Список Tag'ов создан: b:= false

Конструкция описание тождества создана:

.bool b:= false

Создан адрес Addr00: < 0, 0 >

Создана формула вида (boolean)boolean:  ~ < 0, 0 >

СЕМАНТИЧЕСКОЕ ДЕРЕВО ПРОГРАММЫ СОЗДАНО:
BEGIN(0)
  [0] .bool b:= false
  [1] < 0, 0 >:= ~ < 0, 0 >
END(0)

Программа начала ...

ConstructList:
  [0] .bool b:= false
  [1] < 0, 0 >:= ~ < 0, 0 >

После окончания TBooleanAssignment.Run

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

ТЕКУЩЕЕ ОКРУЖЕНИЕ ДО ВЫХОДА ИЗ БЛОКА ТЕКУЩЕГО УРОВНЯ

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

ТЕКУЩЕЕ ОКРУЖЕНИЕ ПОСЛЕ ВЫХОДА ИЗ БЛОКА ТЕКУЩЕГО УРОВНЯ

ПУСТО

Программа кончила!

СТОП !!!
```

**Рис. 2. Протокол исполнения программы: `.begin .bool b; b := ~b.end`**

3. Преобразование статических адресов индикаторов в указатели на значения, которыми они обладают во время исполнения программы, и детали реализации динамики окружений программы, описаны в модуле **ENVIRON**.

4. Ясно также, что конструкции *присваивание* и *описание тождества* лучше всего рассматривать совместно с окружением программы, что и было сделано.

### Литература

1. *Мартыненко Б.К.* Учебный исследовательский проект реализации алгоритмических языков // Компьютерные инструменты в образовании, 2008. № 5. 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, 2009. № 1. С. 10–25.

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

**Abstract**

The article describes representation of plain mode declarations as object-oriented constructs, and implementation of environments in the form of hierarchy of stack frames. The indicators stored in a stack frame are used for the program to have access to valid values. The access method to the values in question is described.

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