электронный научно-технический журнал
ИНЖЕНЕРНЫЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51036. ISSN 2307-0595

Важные этапы изучения основ программирования

 # 12, декабрь 2014

Семин В. М., Русакова З. Н.

УДК: 004.423.4

Россия, МГТУ им. Н.Э. Баумана
a398vm@mail.ru

Введение

Изучение программирования начинается с понятия алгоритма и изучения конструкций языка, позволяющих создать простую линейную программу. Для этого достаточно, например, использовать:

- числовые типы данных real и integer;
- арифметическое выражение;
- оператор присваивания;
- операторы ввода-вывода

Конструкции языков программирования в отличие от естественных языков однозначно определены синтаксисом и семантикой языка и не могут быть произвольно изменены. Синтаксис определяет правила образования конструкций языка, семантика – правила толкования (смысл) синтаксически правильных конструкций.

Линейная программа, операторы в которой выполняются последовательно, не вызывает проблем даже у начинающих программировать студентов, так как с синтаксисом можно согласиться и исправить синтаксические ошибки на этапе компиляции программы. Семантика же конструкций в них простая.

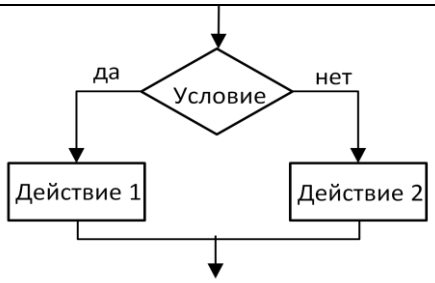
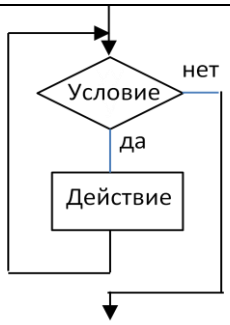
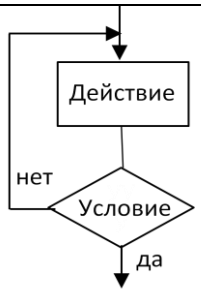
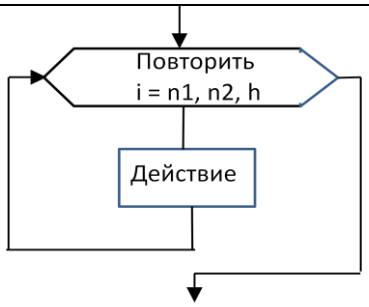
Реальные алгоритмы, содержат циклические и разветвляющиеся фрагменты, которые могут быть вложены друг в друга, образуя сложные логические связи. Для программирования таких алгоритмов при использовании принципов структурного программирования используются управляющие операторы. Они представляют собой конструкцию, позволяющую разветвляющиеся и циклические фрагменты алгоритма описывать фактически с помощью одного управляющего оператора, чаще всего включающего в себя другие управляющие операторы. Умение анализировать последовательность выполнения операторов программы по шагам, даже при знании алгоритма, является важным при тестировании и поиске логических ошибок в программе.

1. Управляющие операторы

Наиболее часто используемыми управляющими операторами «Табл. 1» являются [1,3]:

- оператор условной передачи управления;
- операторы цикла:
  - цикл с предусловием (цикл - пока);
  - цикл с постусловием (цикл - до);
  - цикл с известным числом повторений.

Таблица 1. Структура операторов и их запись на языке Borland Pascal

Стуктура операторов	**Запись операторов на языке Borland Pascal**
	Оператор условной передачи управления ***if*** *Условие* ***then*** *Действие 1* ***else*** Действие 2**;**
	Цикл – пока w***hile*** *Условие* ***do*** *Действие* **;**
	Цикл – до ***repeat*** *Действие* ***until*** *Условие* **;**
	Цикл с известным числом повторений ***for*** *i:=n1* ***to*** *n2* ***do*** *Действие* **;** ***for*** *i:=n2* ***to*** *n1* ***downto*** *Действие* **;**

Условие записывается в виде логического выражения.

Действие, – обработка данных, описывается одним или группой операторов, которые необходимо выполнить для его реализации. Если *Действие* после ключевых слов ***then else do*** содержит группу операторов, то они заключаются в операторные скобки ***begin*** <*Группа операторов*> ***end***.

В цикле с известным числом повторений параметр цикла *i* изменяется от начального значения до конечного в первом случае с шагом +1, а во втором – с шагом -1.

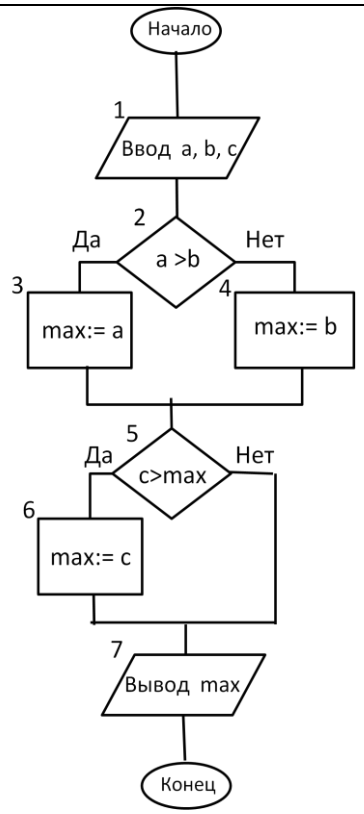
В отличие от цикла с известным числом повторений цикл-до и цикл-пока используются для программирования циклов с неизвестным числом повторений и итерационных циклов.

В состав операторов, описывающих *Действие*, могут входить любые операторы языка, в том числе и управляющие операторы.

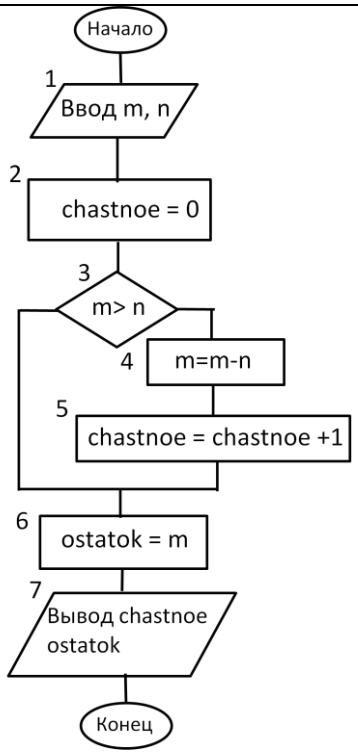
Приведенные несложные алгоритмы иллюстрируют применение управляющих структур при переходе от алгоритма к программе «Табл. 2».

Таблица 2. Использование управляющих структур в программе

Алгоритм и программа нахождения наибольшего значения из трех чисел	

	**program** prim_max;
	Var a,b,c,max:integer;
	begin
	readln(a,b,c);
	if a>b then max:=a else max:=b;
	if c>max then max:=c;
	writeln('max= ',max);
	end. // *конец программы*
	
	Каждый из блоков алгоритма 2,3,4 и 5, 6 в программе представлены одним оператором условной передачи управления.

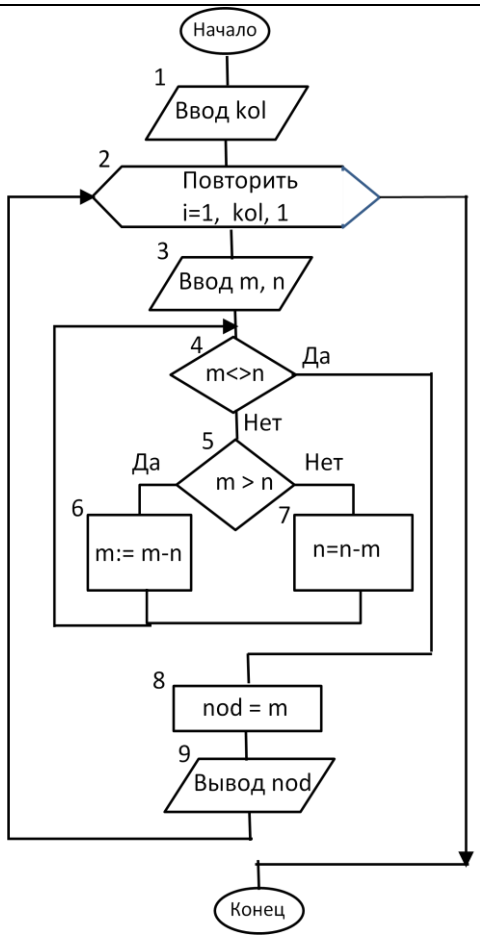
Алгоритм и программа вычисления частного и остатка от деления двух целых неотрицательных чисел.




```
program prim_DivMod;
  var   m,n, Chastnoe, Ostatok:integer;
begin
  readln( m,n);
  Chastnoe:=0;
  While m>=n do
    begin          m:= m - n;
                   Chastnoe:=Chastnoe+1;
    end; // конец While
  Ostatok:=m;
  writeln( 'Chastnoe =',Chastnoe:3,'  ','Ostatok
  =', Ostatok:3);
end.  // конец программы
```

Блоки алгоритма 3,4,5 в программе представлены одним оператором – циклом с предусловием (циклом -пока).

Алгоритм и программа вычисления наибольшего общего делителя NOD двух целых положительных чисел с использование алгоритма Евклида




```
program Prim_NOD;
  var m,n,i,kol,nod :integer;
begin
  readln(kol);
  for i:=1 to kol do
    begin
      readln(m,n);
      While m<>n do
        if m>n then m:=m-n
               else n:=n-m; // конец While
      nod:=m;
      writeln('nod= ',nod:3)
    end; //  конец for i
end  //конец программы.
```

В приведенной схеме алгоритма цикл с известным числом повторений (блок 2) определяет, для скольких пар чисел m и n будет вычисляться NOD. В состав этого цикла входят как линейные действия для ввода и вывода данных (блоки 3, 8, 9), так и блоки, реализующие сам алгоритм Евклида (блоки 4, 5, 6 ,7). Он реализуется с использованием цикла с предусловием (цикла - пока), который включает в себя оператор условной передачи управления (блоки 5,6,7).

Как видно, текст программы существенно снижает восприятие алгоритма и требует от студента не только знания правил работы конструкций, но и умения выделять эти конструкции в программе, как на этапе создания программы, так и при ее отладке.

2. Тесты для контроля знаний работы управляющих конструкций языка

На начальных стадиях освоения языка, даже разобравшись в алгоритме, часто не удается создать программу и воспроизвести последовательность выполнения операторов в ней. Говорить о понимании программы и тем более об осмысленном поиске логических ошибок в программе без умения правильно выполнять операторы программы по шагам не приходится. Конечно, можно использовать отладчик, но он не подскажет, почему ваша последовательность выполнения операторов отличается от его маршрута.

Понимание студентом на начальных этапах обучения правил выполнения операторов, формирует у него общее представление об изучаемом курсе и постепенно формирует осознанное восприятие текста программы.

Для тестов можно использовать:

- тексты программ формальных вычислительных алгоритмов;
- тексты программ, рассмотренных ранее типовых схем алгоритмов.

Само тестирование включает пошаговое выполнение программы студентом вручную на бумаге и ее сравнение с распечаткой пошагового выполнения, полученной программным способом. Программа пошагового выполнения получается из тестовой программы путем вывода значений переменных программы на каждом шаге ее выполнения.

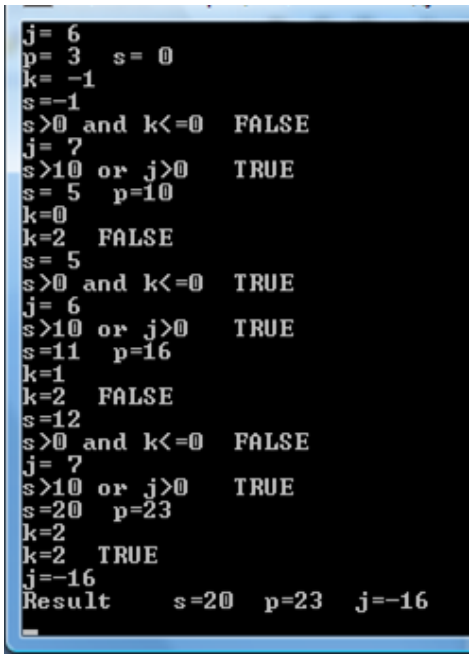
Тесты могут быть использованы как для контроля и оценки знаний студентов, так и для подготовки студентов к контрольным мероприятиям.

В «Табл. 3» приведены тесты для одного формального вычислительного процесса с использованием разных циклических операторов.

Таблица 3. Тесты для контроля пошагового выполнения программы

Оператор Repeat

Программа для тестирования	**Результаты пошагового выполнения операторов**

program Variant_Repeat;
var j,k,p,s:integer;
begin
j:= 6;
p:=3; s:=0; k:=-1;
repeat
s:=s+k;
if (s>0) and (k<=0) then j:=j-1 else j:=j+1;
if (s>10) or (j>0) then **begin** s:=s+k+j ; p:=p+j
end;
k:=k+1;
until k=2; // *Конец repeat*
j:=j-p ;
Writeln('Result ',' s=',s:2,' p=', p:2,' j=',j);
end.
	

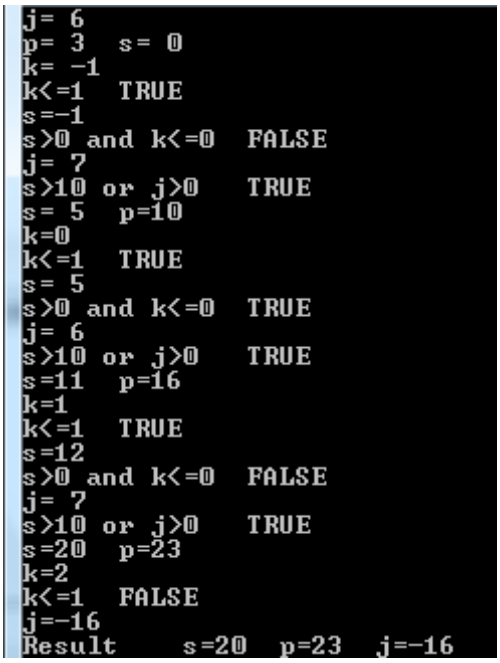
Программа вывода результатов пошагового выполнения операторов

```
program Variant_ Repeat;
        var j,k,p,s:integer;
begin
j:=6;                                   Writeln('j=',j:2);
p:=3; s:=0;                             Writeln('p=',p:2, '  s=',s:2);
k:=-1;                                  writeln('k=', k:3);
repeat

        s:=s+k;                         Writeln('s=',s:2);
                                        Writeln('s>0 and k<=0  ',(s>0) and (k<=0));
        if (s>0) and (k<=0) then j:=j-1  else  j:=j+1;
                                        Writeln('j=',j:2);
                                        Writeln('s>10 or j>0   ',(s>10) or (j>0) );
        if (s>10) or (j>0) then begin s:=s+k+j ; p:=p+j;
                                        Writeln('s=',s:2,'  p=',p:2); end;
        k:=k+1;             Writeln('k=',k) ;  writeln('k=2  ', K=2);
until k=2;
j:=j-p;                                  Writeln('j=',j:2);
Writeln('Result   ','s=', s:2,'  p=',p:2, '  j=',j);
end.
```

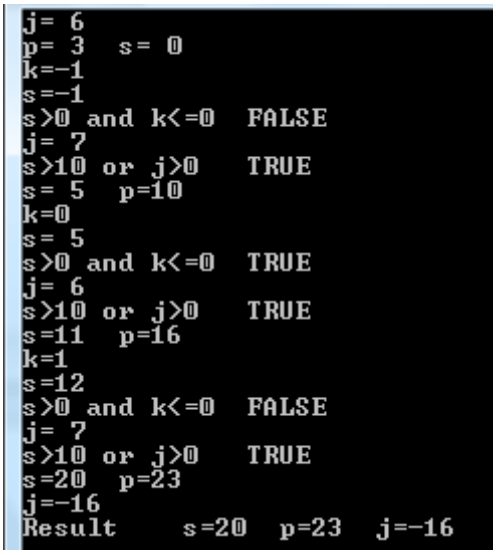
Оператор While

Программа для тестирования	Результаты пошагового выполнения операторов

program Variant_While;
var j,k,p,s:integer;
begin
j:= 6;
p:=3; s:=0; k:=-1;
while k<=1 do
begin s:=s+k;
if (s>0) and (k<=0) then j:=j-1 else j:=j+1;
if (s>10) or (j>0) then **begin** s:=s+k+j ;
p:=p+j
end;
k:=k+1
end; *// Конец while*
j:=j-p ;
Writeln('Result ',' s=',s:2,' p=', p:2,' j=',j);
end.
	

Оператор For

Программа для тестирования	Результаты пошагового выполнения операторов

program Variant_For
var j,k,p,s:integer;
begin
j:= 6;
p:=3; s:=0;
for k:=-1 to 1 do
begin s:=s+k;
if (s>0) and (k<=0) then j:=j-1 else j:=j+1;
if (s>10) or (j>0) then **begin** s:=s+k+j ;
p:=p+j
end;
end; *// Конец for*
j:=j-p ;
Writeln('Resulat ',' s=',s:2,' p=', p:2,' j=',j);
readln
end.
	

Предложенный способ позволяет проводить само тестирование и подготовку к нему в аудитории. Подготовку к тестированию можно проводить и с использованием отладочных средств Турбо Delphi, предварительно выполнив пошаговое выполнение программы вручную и сравнив ее с результатами, полученными в среде отладчика.

3. Отладчик как средство обучения практике поиска логических ошибок

Основное назначение встроенного в Турбо Delphi отладчика – значительно упростить процесс отладки программы. Под отладкой понимается поиск логических ошибок, как на начальном этапе отладки программы, так и при ее тестировании.

Конечно, для нахождения таких ошибок в программе необходимо понимать (представлять) семантику конструкций языка и знать алгоритм решаемой задачи. Приобретение же студентом навыков поиска таких ошибок закрепляет ранее полученные знания и выводит его на новый уровень понимания изучаемого курса.

На начальном этапе работы с Турбо отладчиком достаточно освоить основные инструменты отладки:

- установка точек прерывания (останова) выполнения программы;
- просмотр значений переменных;
- выполнение программы построчно/по шагам (трассирование программы).

Существуют разные возможности выполнения этих действий [3]. Не отвлекая внимание студента на их многообразие, целесообразно остановиться на использовании команды Debug всплывающего меню, вызываемого правой клавишей, и кнопок панели инструментов Debug.

Для установки /удаления точки прерывания программы необходимо щелкнуть правой клавишей по выбранной строке программы и выполнить команду *Debug / TuggleBreakepoint*. Аналогичным образом «Рис. 1» с помощью команды *Debug / Add Watch at Curcor* производится занесение переменных в окно *Watch List* для наблюдения за их изменением «Табл. 4». Курсор при этом должен быть предварительно установлен на имени переменной. Просматривать значения переменных можно просто устанавливая в режиме отладки курсор на имя переменной «Рис. 2».

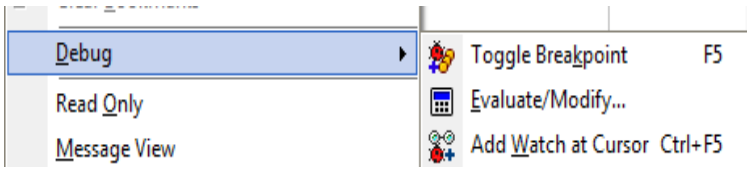


Рис. 1. Установка точки прерывания и занесение переменных в окно наблюдений

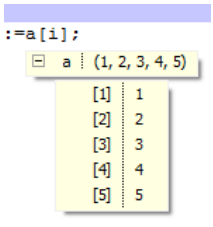


Рис. 2. Просмотр значений переменных

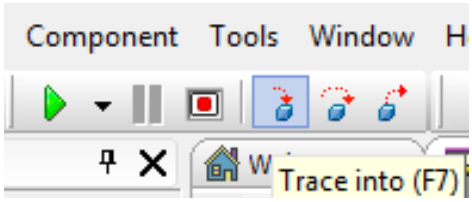


Рис. 3. Панель Debug

Панель инструментов Debug «Рис. 3» удобно использовать для пошагового выполнения операторов программы. Вызов панели инструментов производится с помощью команды *View / Toolbars*.

Кнопка *Trace Into* позволяет трассировать операторы программы включая имеющиеся в программе функции и процедуры. Расположенная правее этой кнопки кнопка *Trace Over* позволяет избежать трассировки уже отлаженных функций и процедур.

Для повышения эффективности работы программы компиляция ее производится с оптимизацией. Оптимизация сокращает выполняемый файл, но при этом уменьшает объем информации, предоставляемой отладчиком, удаляя из кода откомпилированной программы результаты выполнения некоторых простых операторов, не позволяя просматривать изменения переменных.

В «Табл. 4» представлены изменения переменных при включенном ключе оптимизации. Как видно перед началом выполнения цикла переменные ‘i’ и ‘n’ недоступны, даже при выполненном операторе n:=0. При первом и всех последующих повторениях цикла переменная ‘n’ остается недоступной, а параметр цикла ‘i’ производит обратный отсчет количества оставшихся повторений цикла 5, 4, 3, 2, 1.

Таблица 4 Просмотр значений переменных в окне *Watch List*

<table>
<tr><th colspan="2">Ключ оптимизации включен</th></tr>
<tr><td>

```
program Project2;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var m,n,i:integer;
begin
  m:=0;  n:=0;
  For i:=1 to 5 do
              m:=m+1;
  n:=m+1;
  Writeln('n= ',n);
  readln
end.
```

</td><td>

Watch List

Watch Name	Value
i	E2171 Variable 'i' inaccessible here due to optimization
m	0
n	E2171 Variable 'n' inaccessible here due to optimization

Watch List

Watch Name	Value
i	4
m	2
n	E2171 Variable 'n' inaccessible here due to optimization

Watch List

Watch Name	Value
i	E2171 Variable 'i' inaccessible here due to optimization
m	E2171 Variable 'm' inaccessible here due to optimization
n	6

</td></tr>
<tr><td>Ключ оптимизации выключен</td><td>

Watch List

Watch Name	Value
i	3
m	2
n	0

</td></tr>
</table>

Для получения более полной информации об изменении переменных перед началом работы с отладчиком необходимо изменить установку компилятора:

1. Выполнить *Projekt / Options / Compiler*;
2. На панели *Code Generation* сбросить флажок *Optimization*

Поиск логических ошибок по своему характеру является творческой работой. Ошибка находится как вследствие целенаправленных действий, так и на основании опыта. Важным условием является знание алгоритма программы и связи переменных и модулей программы между собой. Существуют различные способы поиска логических ошибок от анализа текста программы, использования отладочных печатей, обратного отслеживания выполнения программы по операторам до использования дампа памяти [7]. Самым распространенным способом для небольших программ и несложных структур данных является выполнение программы по шагам с использованием отладочных средств.

Практика проведения занятий показывает, что студенты, даже имея неплохие знания, часто пытаются устранить ошибки методом проб и ошибок или только путем анализа текста программы. Первый метод неприемлем, второй требует хорошего знания языка и алгоритма, хотя алгоритм может быть тоже неверным. Все это чаще приводит к внесению дополнительных ошибок в программу.

Работа с отладчиком начинается с освоения рассмотренных инструментов отладки на отлаженной программе при выключенном режиме оптимизации компиляции программы. Чтобы выполнить каждый оператор по шагам, необходимо каждый из них разместить на отдельной строке. Если в строке будет несколько операторов, они выполнятся за один шаг.

Методика использования отладчика приходит с опытом. Для приобретения же начальных практических навыков поиска логических ошибок предлагается использовать программы известных студенту алгоритмов с внесенными в них ошибками. Входные данные для программ с целью проявления ошибок могут быть заданы с помощью константных значений переменных и структур данных или предложены преподавателем.

Студент, зная содержание запрограммированной задачи и даже алгоритм, должен быть ориентирован на нахождение ошибки только с использованием отладочных средств. Подтверждением нахождения причин ошибки должны служить значения переменных на конкретном шаге выполнения программы и их обоснованная интерпретация.

Отладка начинается с того фрагмента программы, где первый раз встретился неправильный результат или произошло прерывание программы. При простых алгоритмах отладку лучше начинать сразу после ввода входных данных с контролем правильности их ввода средствами отладчика.

Для иллюстрации такого подхода показаны примеры простых типовых программ с внесенными в них ошибками «Табл. 5», которые не так уж редки в самом начале изучения курса.

Таблица 5 Примеры программ для поиска логических ошибок

<table>
<tr><th colspan="2">Программа нахождения максимального элемента массива.</th></tr>
<tr><td>Ошибка внесена в 18 строку программы – b[i]:=max. Результат работы программы max=1.10.</td><td>Ошибка проявляется при i=3, когда b[3]>max, но max остается равным 1.1</td></tr>
<tr><td>

```
program Project1;

{$APPTYPE CONSOLE}

uses
  SysUtils;

const a:array [1..5]of real=(1.1, 0.5, 5.0, 3.5, 1.0
 Var  max:real;   i:integer;
      b: array [1..5]of real;
begin

  for i:=1 to 5 do b[i]:=a[i];

  max:=b[1];
    for i:=2 to 5 do
    if b[i]>max then
                 b[i]:=max;
      writeln('max=  ',max:5:2);
      readln
end.
```

</td><td>

Watch List

Watch Name	Value
a	(1,1, 0,5, 5, 3,5, 1)
b	(1,1, 0,5, 1,1, 3,5, 1)
i	4
max	1,1

</td></tr>
<tr><td>Ошибка внесена в 13 строку программы – if a[i]>a[1] Результат работы программы max=3.50</td><td>Ошибка проявляется на последнем шаге при i=5, когда предыдущее значение max=5 становится равным max=3.5</td></tr>
<tr><td>

```
program Otladka;

{$APPTYPE CONSOLE}

uses
  SysUtils;

const a:array [1..5]of real=(1.1, 0.5, 3.5, 5.0, 3.5)
 Var  max:real;   i:integer;
begin
    max:=a[1];
    for i:=2 to 5 do
    if a[i]>a[1] then
                 max:=a[i];
      writeln('max=  ',max:5:2);
      readln
end.
```

</td><td>

Watch List

Watch Name	Value
a	(1,1, 0,5, 3,5, 5, 3,5)
max	3,5
i	6

</td></tr>
<tr><td>Программа вычисления суммы элементов массива до первого a[i]>0. Результат работы программы sum= -5.</td><td>Ошибка проявляется при i=5, когда к верному значению полученной суммы добавляется a[5]=5</td></tr>
<tr><td>

```
program Kontroly;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const a: array [1..6] of integer =(-1,-2,-3,-4,
 var       sum,i:integer;

begin
    sum:=0;
    i:=0;
   repeat     i:=i+1;
              sum:=sum+a[i];
    until (a[i]>0) or (i=6);
   writeln('sum= ', sum);
   readln;
end.
```

</td><td>

Watch List

Watch Name	Value
a	(-1, -2, -3, -4, 5, 6)
i	5
sum	-10

</td></tr>
</table>

Можно подобрать много других простых примеров, например, написание программы преобразования квадратной матрицы по правилу $A[i,j] = {A[i,j]}/{A[i,i]}$,где кажущаяся простота задачи чаще не приводит сразу к получению правильных результатов и требует отладки.

Заключение

Практика проведения лабораторных работ с использованием предлагаемых тестов и методик способствует освоению языка программирования, формирует представление о взаимодействии блоков и структур данных программы между собой, вырабатывает у студента понятие необходимости применения и расширения арсенала средств отладки. Полученный первый опыт далее используется при отладке программ с процедурами и новыми структурами данных и постепенно повышает заинтересованность студента (интерес охотника) в доведении отладки до логического конца.

Список литературы

1. Иванова Г.С. Основы программирования.–М.: Изд-во МГТУ им. Н.Э. Баумана, 2007. – 416 с.
2. Алексеев В.Е., Ваулин А.С., Куров А.В. Практикум по программированию. Обработка числовых данных. М.: Изд-во МГТУ им. Н.Э. Баумана, 2008, 288 с.
3. Парфилова Н.И., Пруцков А.В., Пылькин А.Н., Трусов Б. Г. Информатика и программирование. Том 2. Алгоритмизация и программирование.– М.: Издательский центр «Академия», 2012. – 343 с.
4. Криницына Л.Ф., Мартынюк Н.Н. Сложные типы данных. Массивы. Методические указания. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2000.
5. Блог Rouse:Изучаем отладчик, часть первая. Режим доступа: http://alexander-bagel.blogspot.ru/2012/10/debugger-1.html.html (дата обращения 21.11.2014)
6. Интегрированный отладчик Delphi. Режим доступа: http://www.helloworld.ru/texts/comp/lang/delphi/delphi4/DebugerDelphi.htm (дата обращения 21.11.2014).
7. Отладка и тестирование программ. Режим доступа: http://exinform09.narod.ru/otveti/65_66_67.htm (дата обращения 21.11.2014).