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

Автоматизация тестирования студенческих программ в среде Turbo Delphi

 # 12, декабрь 2014

Алексеев Ю. Е., Куров А. В.

УДК: 81.3.069

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

Переход к новым программам обучения студентов сопровождается сокращением количества учебных занятий, проводимых непосредственно в аудиториях и лабораторных залах. Однако существенно сократить объем рассматриваемых знаний не представляется возможным, поэтому насущной потребностью является интенсификация всего учебного процесса, требующая более эффективной организации занятий, что требует и большей отдачи как от преподавателей, так и студентов. Помочь решить поставленную задачу может широкое внедрение в процесс преподавания информационных технологий, прежде всего в преподавание такой дисциплины, как Информатика.

Значительная часть этой учебной дисциплины связана с изучением основ программирования. Изучение программирования студентами первого курса сопровождается целым рядом проблем. Эти проблемы обусловлены тем, что значительная часть студентов первого курса не имеет предшествующего опыта программирования и, соответственно, навыков по разработке алгоритмов, их программной реализации, тестированию и отладке создаваемых программ. Часто, представляя к защите написанную и выдающую некоторые результаты программу, студенты не задумываются о том, что не всякий результат, выданный программой, может являться правильным.

Информация о синтаксических ошибках, выдаваемая транслятором, заставляет студента ознакомиться с сообщениями транслятора и внести исправления в свою программу. В случае же алгоритмических ошибок или ошибок, связанных с неправильной записью и, соответственно, с неверным выполнением вычислительных операций, не препятствующих выполнению программы полностью, студент никаких сообщений о допущенных ошибках, естественно, не получает. В этом случае необходимо сравнить полученные и правильные результаты. Однако уже при написании программ линейной структуры, в которых может производиться вычисление значения довольно громоздкого выражения, возникают определенные трудности, связанные с отсутствие у студентов навыков быстрого (практически устного) оценивания правильности полученного результата. Задача в этом случае связана и с выбором таких исходных данных, при которых действительно практически в уме мож-

но вычислить требуемое значение, и с вычислением значений элементарных функций при определенных значениях аргументов. Сложности при решении этих двух взаимосвязанных задач обусловливаются недостаточно прочными знаниями, полученными студентами еще при изучении математики в средней школе.

Подобная ситуация достаточно часто возникает на занятиях по информатике, несмотря на то, что уже на первых семинарах и лабораторных работах студенты знакомятся со средствами отладки, имеющимися в составе используемой среды программирования, и учатся использовать эти средства в своей практической деятельности. Преподаватель, в силу указанных обстоятельств, приступая к приему очередной лабораторной работы, прежде всего должен убедиться в правильности работы программы. В одних случаях это легко сделать, изучив только текст представленной программы, в других случаях необходимо проверить работу программы на определенных наборах исходных данных. В ряде случаев ошибки, допущенные в программе, могут проявляться только при определенных значениях исходных данных. Однако во всех рассмотренных случаях преподаватель и студент затрачивают дополнительное время, что, в конечном счете, снижает эффективность проводимых занятий.

Для повышения эффективности проводимых занятий следует предложить технологию, которая позволяла бы с минимальными затратами времени обнаруживать ошибки в студенческих программах. Предлагаемая технология позволяет путем тестирования программы выявить наличие ошибок в студенческой программе. Под тестированием в данном случае понимается проверка выполнения программой своих функций на некотором заранее подготовленном или генерируемом при выполнении наборе данных. В общем случае подобное тестирование не гарантирует выявление всех ошибок, допущенных в программе. Оно также не позволяет решить некоторые задачи обучения студентов. В частности, изучение основ программирования предполагает освоение определенных алгоритмов и приемов решения задач, поэтому выполнение заданий предусматривает использование рассматриваемых алгоритмов и методов решения сформулированных задач. Понятно, что осуществить проверку использования в программе определенного алгоритма решения задачи не представляется возможным.

Согласно предлагаемой технологии тестирования студенческих программ предлагается организовать работу программы в двух режимах – без тестирования (в обычном режиме) и с тестированием. Здесь рассматривается тестирование определенных частей программы, выполняющих вычислительные операции по обработке данных, и некоторые элементы ввода данных. При реализации поставленных целей на структуру программы накладывались некоторые ограничения, соответствующие принципам структурного программирования. К числу этих ограничений отнесены следующие: части программы должны быть функционально законченными, части программы должны иметь только одну точку входа и одну точку выхода, вычислительные части программы не должны каким-либо образом менять входные данные, должны иметь вычисленные значения в точке выхода,

не должны каким-либо образом, кроме как путём преобразования, менять переменные, являющиеся и входными, и выходными.

Кроме того, имена всех переменных, используемых на входе и выходе в тестируемых частях программы, должны по, типу и назначению совпадать с указанными в задании. Более того, допускается их описание, в том числе с начальными значениями, в шаблонах (о них речь пойдёт ниже), подготавливаемом для разработки программ по учебным заданиям.

Рассмотрим реализацию предлагаемой технологии тестирования на примере создания консольного приложения в среде программирования Turbo Delphi. Это обусловлено тем, что согласно учебным программам на большинстве специальностей университета изучается язык программирования Pascal и используется среда программирования Turbo Delphi, причем в течение всего первого семестра студентами разрабатываются консольные приложения. Это обусловлено тем, что в первую очередь студенты должны освоить основы программирования и научиться реализовывать основные алгоритмы и разрабатывать программы, в которых используются эти алгоритмы. Таким образом, основное внимание на начальных этапах уделяется вопросам алгоритмизации инженерных задач и отладке программ, реализующих эти алгоритмы, а не вопросам разработки интерфейса, которые в большей степени рассматриваются уже во втором семестре.

При создании нового проекта в консольном режиме в среде Turbo Delphi основная программа – файл с именем проекта и расширением .dpr изначально содержит стандартный шаблон, а разработка программы состоит в добавлении в стандартный шаблон операторов, объявлений, директив и проч.

В предлагаемой технологии для автоматизации тестирования требуется заменить стандартный шаблон новым, специально разработанным для каждого задания. В новом шаблоне в минимальной реализации должны присутствовать:

- директива компилятора ($define};
- одна или несколько директив компилятора вида {%File 'pr1.txt'}, где строка 'pr1.txt' определяет имя файла, хранящего код, добавляемый в исходный шаблон;
- одна или несколько директив компилятора вида{$I pr1.txt}, позволяющая указать место вставки кода, содержащегося в файле с именем pr1.txt;
- директивы условной компиляции {$ifdef test}, {$endif};
- комментарии, сопровождающие эти директивы, или имеющие иное назначение.

Для создания шаблона в среде программирования Turbo Delphi надо сначала подготовить текстовые файлы, содержащие добавляемые программные коды. Для этого в меню надо выбрать пункт File, затем New, Other, а затем в раскрывшемся окне выбрать Text. После этого в пункте меню Project выбрать подпункт Add to Project и добавить в проект созданные текстовые файлы.

Для нового шаблона, как и для стандартного, запрещается редактировать, удалять и изменять положение любых частей шаблона. Исключением является директива #define,

которая изначально закомментирована, то есть представлена в виде однострочного комментария, что соответствует обычному, без тестирования режиму работы программы

Директива вида {$I pr1.txt} позволяет включить в код программы содержимое файла, имя которого содержится в директиве. Эти директивы делят разрабатываемый код программы на части, реализующие отдельные подзадачи задания, которые явно формулируются в задании, если требуется соблюсти определённые требования (например, использовать определённый метод вычисления или определённым образом организовать ввод, вывод или задание значений данным), или по умолчанию, когда нет специальных требований (например, если в задании формулируется только, какие вычисления должна выполнять программа, то ввод исходных данных должен выполняться с клавиатуры и вывод результатов - в окно программы).

В новом шаблоне могут присутствовать отдельные комментарии, не связанные с директивами, обозначающие место начала и/или конца части кода программы. Таким образом, новый шаблон может дополнять задание.

Рассмотрим общий вид шаблона для задания, требующего ввода исходных данных – матрицы.

```
program progwwod;
{$APPTYPE CONSOLE}
uses  SysUtils;
{%File 'p1.txt'}
{%File 'p2.txt'}
{%File 'p3.txt'}
        {$define test}


{$ifdef test}
        {$I p1.txt}
{$endif}
//здесь разместите объявление необходимых констант, типов и переменных

begin
//здесь разместите код для ввода количества строк и столбцов матрицы

{$ifdef test}
        {$I p2.txt}
{$endif}
//здесь разместите код для ввода элементов матрицы по строкам

$ifdef test}
        {$I p3.txt}
```

```
{$endif}
readln;
end.
```

С помощью рассматриваемого шаблона предлагается контролировать правильность организации ввода элементов матрицы: элементы должны вводиться построчно. Контроль основан на том, что случайные числа, генерируемые датчиком, записываются в соответствии с предъявляемыми требованиями в файл, а также выводятся для образца студенту, который должен ввести именно такие же числа. Затем сравниваются введенные значения элементов матрицы с записанными в файл. В случае ошибки будет выдало сообщение об этом.

Файл p1.txt содержит объявление дополнительных, необходимых для проведения контроля, переменных: f - файловая переменная, flag – признак наличия ошибки при вводе. Его содержание выглядит следующим образом:

```
var   f:textfile;       flag:boolean;
```

Файл p2.txt содержит программный код, позволяющий подготовить файл к работе, сгенерировать и записать в файл случайные числа, осуществляя запись элементов матрицы строго по строкам (в виде матрицы). Его содержимое приведено ниже:

```
//формирование предложения для ввода и запись в файл
writeln(Rus('ВВЕДИТЕ МАТРИЦУ ТАК, КАК ОНА ПРЕДСТАВЛЕНА НИЖЕ'));
assignfile(f,'t2.txt');
rewrite(f);
randomize;
// генерация элементов матрицы, вывод матрицы для пользователя
// и запись элементов в файл
for i := 1 to m do
begin
        for j := 1 to n do
        begin
                k:= random(9);
                write(f,k:3);
                write(k:3);
        end;
        writeln;
        writeln(f);
end;
closefile(f);
```

В файле p3.txt содержится программный код, который и обеспечивают контроль правильности ввода элементов матрицы путем сравнения значений, считанных из файла, со значениями матрицы.

```
// Тестирование ввода матрицы
assignfile(f,'t2.txt');
reset(f);
flag:=true;
for i := 1 to m do
begin
  for j := 1 to n do
  begin
    read(f,k);
    write(k:3);
    if mx[i,j]<>k then
    begin
      flag:=false;
      break;
    end;
    if not flag then break;
  end;
  readln(f);
  writeln;
end;
closefile(f);
ERASE(f);
if flag then
  writeln(Rus('НЕ ОБНАРУЖЕНО ОШИБКИ!'))
else
  writeln(Rus('ОШИБКА ВВОДА МАТРИЦЫ!'));
```

В качестве второго примера рассмотрим шаблон программы для проверки правильности программы, выполняющей вычисление значения некоторой функции.

Для примера взята первая часть задания 3.3.2, сстраница 51 из практикума по программированию [4]: требуется составить программу вычисления с помощью минимального числа операторов if else значения функции Y(X) по формуле

$$Y(X) = \begin{cases} 0 \text{ при } x <- 1, \\ 1 \text{ при } -1 \le X < 0, \\ -1 \text{ при } 0 \le X < 2, \\ 1 \text{ при } 2 \le X. \end{cases}$$

К этому условию добавим, что тестирование следует проводить только после разработки части программы, вычисляющей Y(X).

```
program progfun;
{$APPTYPE CONSOLE}
uses  SysUtils;
```

```
{%File 'tst1.txt'}
{%File 'tst2.txt'}
{%File 'tst3.txt'}
{$define dbg=1} // 1 или 2
{$define dbg2=1}

{$ifdef dbg=1}
  {$I tst1.txt}
{$endif}
//Здесь разместите объявление констант, типов и переменных

begin
{$ifndef dbg1 }
//здесь разместите код для вывода подсказки пользователю и код для
//обращения к процедуре ввода значения переменной x
{$endif}

{$ifdef dbg1}
  {$I tst2.txt}
{$endif}

//здесь разместите код для вычисления значения функции Y(X)

{$ifdef dbg1}
  {$I tst3.txt}
{$endif}

{$ifdef dbg1}
  {$I tst3.txt}
{$endif}
  readln;
end.
```

После копирования нового шаблона в файл главной программы с расширением .dpr, в её текст следует добавить:

* в первую часть программы вывод подсказки для ввода и обращение к процедуре ввода значения переменной x:

```
writeln(Rus('Введите  x: '));
  readln(x);
```

* во вторую часть – операторы вычисления Y(X) и вывод полученного значения функции :

```
if(x<-1) then
  y:=0
else
  if(x<0) then
    y:=1
  else
        if(x<2) then
          y:=-1
        else
          y:=1;
writeln(Rus('При x='),x:5:2,Rus('  Функция y='),y:5:2);
```

При определении значения dbg работа программы происходит в режиме тестирования, а при определенном значении dbg1 значение аргумента может принимать случайное значение.

Файл p1.txt содержит объявление переменных, которые должны быть добавлены в программу в режиме тестирования. Его содержимое следующее:

```
var
  xx:array[1..7] of real=(-3, -1,  -0.5,  0, 1, 2, 4);
  yy:array[1..7] of real=( 0, 1, 1, -1, -1, 1,  1);
  r:real;
  i:integer;
  flag:boolean;
```

Массивы xx и yy содержат соответственно тестовые наборы значений аргумента и соответствующие им значения функции, r – случайная величина, I – параметр цикла, flag – признак наличия (отсутствия) ошибки вычислений.

Файл p2.txt содержит операторы, позволяющие вносить случайное изменение в значения аргумента, а также заголовок цикла для тестирования программы при значениях аргумента, занесенных в массив. При внесении случайных изменений значения аргумента остаются принадлежащими тому же интервалу, т.е. значение функции при этом не изменяется.

Его содержимое имеет вид:

```
{$ifdef test1}
 randomize;
r:= random();
if(0<=r)and(r<=0.14) then
  xx[1]:=xx[1]-r;
if (0.14<r)and(r<=0.28) then
  xx[2]:=xx[2]+r;
if (0.28<r)and(r<=0.42) then
  xx[4]:=xx[4]+r;
```

```
if (0.42<r)and(r<=0.56) then
  xx[5]:=xx[5]+r;
if (0.56<r)and(r<=0.7) then
  xx[6]:=xx[6]+r;
if (0.7<r)and(r<=0.86) then
  xx[7]:=xx[7]+r;

{$endif}
flag:=true;
for i:=1 to 7 do
begin
  x:=xx[i];
writeln('x=',x:5:2);
```

Файл p3.txt содержит операторы, оканчивающие цикл, которые как раз и осуществляют проверку правильности вычисленных значений:

```
if(x=xx[i])and (yy[i]<>y) then
begin
  writeln(Rus('Ошибка при x = '),x:6:2,'  y=',y:5:2);
  flag:=false;
end;
if flag then      writeln (Rus('Ошибок нет!'));
```

В случае обнаружения ошибки в окно программы будет выведен текст ОШИБКА ПРИ x=, значение x, при котором ошибка обнаружена, и вычисленное значение функции. В случае отсутствия ошибок после выполнения цикла тестирования будет выдано сообщение Ошибок нет!

Файл шаблона и файл тестирования следует при выдаче задания поместить в папку проекта, в которой расположен файл основной программы с расширением .dpr. Это упростит выдачу задания, а для ссылки на файл тестирования в директивах {$I имя файла} достаточно будет указать только его имя.

В общем случае файлов тестирования может быть несколько, а заложенные в них коды и алгоритмы препроцессорной обработки могут определять как произвольные, так и вполне определённые последовательности разработки частей программы. Эти вопросы требуют отдельного рассмотрения.

Список литературы

1. Канер С., Фолк Дж., Нгуен Е. К. Тестирование программного обеспечения. Пер. с англ. -Киев: "ДиаСофт", 2001. - 544 с.
2. Винниченко И. В. Автоматизация процессов тестирования. – СПб.: Питер, 2005.- 203 с.

3. Дал У., Дейкстра Э., Хоор К. Структурное программирование Пер. с англ. - М.: Мир, 1975. - 247 с.
4. Алексеев Ю. Е., Ваулин А. С., Куров А. В. Практикум по программированию. Обработка числовых данных : учеб. пособие для вузов / Под ред. Б.Г. Трусова. - М. : Изд-во МГТУ им. Н. Э. Баумана, 2008. – 288 с.
5. Алексеев Ю. Е., Куров А. В Автоматизация тестирования студенческих программ. Инженерный журнал: наука и инновации, 2013, вып. 6. URL:http://engjournal.ru/catalog/it/hidden/768.html