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

Основные отличия и особенности преподавания курса информатики при переходе от языка Паскаль к языку С

09, сентябрь 2016

Алексеев Ю. Е.[1], Куров А. В.[1,*]

УДК: 681.3.069

[1]Россия, МГТУ им. Н.Э. Баумана
*avkur7@mail.ru

Введение

Основу курса информатики, преподаваемого кафедрой ИУ7 студентам первого курса университета, составляет изложение основ алгоритмизации и программирования. С момента основания кафедры ИУ7, совпавшего по времени с широким использованием в учебном процессе персональной вычислительной техники и появлением новых языков и сред программирования, студентам большинства специальностей университета преподавался и до сих пор преподается язык Паскаль (конкретно его объектно-ориентированная версия ObjectPascal, реализованная в среде программирования Turbo-Delphi). В первоначальном варианте язык Паскаль по замыслу его создателя профессора Цюрихского университета Н. Вирта был предназначен для обучения основам программирования старшеклассников и студентов младших курсов ВУЗов. Многолетний опыт преподавания информатики показал, что использование данного языка программирования в учебном процессе оказалось оправданным. У студентов большого количества специальностей, непосредственно не связанных с программированием, не возникает больших проблем с записью изучаемых алгоритмов на используемом языке программирования, использованием конструкций языка, а также с написанием и отладкой программ в целом. Основные проблемы возникают, как правило, на этапе разработки алгоритма решаемой задачи.

Такому положению дел способствуют основные принципы, заложенные Н. Виртом и реализованные в различных версиях языка Паскаль: жесткая типизация данных, простой синтаксис, наличие ограниченного количества операторов, реализующих базовые управляющие структуры. Наличие готовых подпрограмм, реализующих основные математические функции, а также обеспечивающих обработку данных разных типов (символьных, строковых, файловых, множеств) также позволяет упростить процесс написания программ. Жесткие правила языка, ограничивающие свободу действий программиста, также упрощают жизнь начинающим программистам, не позволяя записывать определенные конструкции (например, переменной целого типа можно присваивать значение только этого типа), что предотвращает возможные ошибки на стадии выполнения программы.

***

При изучении языка Паскаль основные проблемы возникают у студентов, как правило, при освоении двух разделов: механизмы передачи параметров в подпрограммы и работа с динамическими данными. Однако в первом случае существуют простые формальные правила записи заголовков подпрограмм, следование которым позволяет записывать подпрограмму даже в том случае, когда студент не слишком глубоко разобрался в сути методов передачи данных между подпрограммами. Второй из упомянутых разделов в последние годы в связи с сокращением часов, отводимых на преподавание дисциплины, практически не рассматривается в курсе, а использование динамических массивов сводится по сути к использованию готовой подпрограммы, выполняющей все необходимые операции по выделению динамической памяти.

Однако в последние годы от ряда кафедр поступили предложения по использованию языка С в учебном процессе. Одними из первых в их числе оказались кафедры ИУ1 и ИУ3. Это объясняется тем, что в своей практической деятельности после окончания ВУЗа для решения возникающих задач выпускникам ряда кафедр приходится использовать более гибкие и мощные языки программирования, позволяющие в наилучшей степени справиться с решением проблем.

Количество кафедр, высказывающих пожелания о смене преподаваемого языка программирования, возрастает с каждым годом. Авторы накопили опыт преподавания языка С на ряде кафедр университета и считают целесообразным поделиться им в связи с актуальностью замены преподаваемого языка программирования в ближайшие годы.

Проблемы и сложности в процессе изучения языка С могут возникать уже на самых ранних этапах. Это обусловливается большими возможностями языка С, его большей гибкостью и одновременно большей лаконичностью, что, в конечном счете, возлагает и большую ответственность на самого программиста.

В первую очередь, проблемы могут возникнуть в связи с отсутствием жесткой типизации данных в языке С. Поскольку вся информация в ЭВМ представляется в двоичной системе счисления, то, как будет интерпретироваться хранящаяся в памяти последовательность нулей и единиц (двоичное число), зависит от автора программы. Наиболее просто и наглядно это выглядит при обработке символьных данных. Поскольку каждый символ представляется своим кодом (целым числом), а с целыми числами можно выполнять арифметические операции, то язык С позволяет те же операции выполнять и над самими символами. Фактически при изложении языка Паскаль и языка С приходится использовать два разных исходных тезиса.

В первом случае можно опираться на тот факт, что в повседневной практике мы не можем выполнять арифметические операции с символами (непонятно, каков будет результат операции ‘!’/’?’ – деление восклицательного знака на вопросительный). В языке С в качестве исходного положения приходится использовать тезис о кодировании символов. Поскольку символы представляются целыми числами, то в этом случае можно выполнять над ними те же действия, что и с целыми (сложение, вычитание, умножение, деление,

определение остатка от деления). Получаемый результат можно интерпретировать (выводить) двояко: представить в виде целого числа или символа (код которого соответствует полученному результату). Это зависит от спецификации формата, согласно которой производится вывод значения.

При переходе от языка Паскаль к языку С надо учитывать и некоторые особенности синтаксиса, заключающиеся, в частности, в использовании разных символов для записи операций. Наиболее часто встречающаяся ошибка – запись операции отношения “сравнение на равенство”: в Паскале для ее записи используется один знак равенства (=), а в С – два знака равенства (==). Согласно правилам языка С, конструкция if (c==5) a=4; else a=7; и if (c=5) a=4; else a=7; являются допустимыми, следовательно, программа будет выполняться и будет получен результат, который, скорее всего, окажется неверным.

В случае, когда одним из операндов выражения является константа, можно избежать подобной ошибки, записав выражение в виде 5==c (поставить константу на первое место). При неправильной записи 5=c транслятор выдаст сообщение о синтаксической ошибке, т.к. нельзя константе присваивать значение (изменять ее).

Похожая ситуация возникает и при записи логического умножения, требующего также дублирования знака &: a>b&&c>d. В случае использования одного знака & должно выполняться логическое (поразрядное) умножение, а не логическое умножение. Ошибка не будет обнаружена транслятором в том случае, когда операнды являются значениями целого типа. В этом случае может выполняться как операция логического умножения, так и поразрядного логического умножения. Это объясняется тем, что в языке С любое значения, отличное от нуля, трактуется как истина, а нуль трактуется как ложь.

При написании программ необходимо помнить правила обработки тех или иных типов данных (правила выполнения операций). В частности, часто у начинающих программистов возникают ошибки, связанные с выполнением операции деления, которые в аналогичных ситуациях не возникают в программах, написанных на языке Паскаль. Это связано с тем, что тип результата операции деления определяется типом операндов: 3./5.=0.6, а 3/5=0. Подобная ошибка часто возникает у студентов при записи выражений, в частности, при возведении числа в дробную степень, например, при вычислении среднего геометрического нескольких значений: srg=pow(p,1/n); При n>1 (n –целое, количество сомножителей) будет получен неверный результат, т.к. второй аргумент функции будет равен нулю. В Паскале такая ошибка не возникает, т.к. деление, обозначаемое знаком /, дает всегда результат действительного типа, а для получения результата целого типа используется другая операция – деление целых с получением целочисленного результата, обозначаемая div. При попытке использования ее к действительным величинам возникает синтаксическая ошибка.

Несмотря на использование в языках всех допустимых символов, вводимых с клавиатуры, их не хватает для использования только по одному единственному назначению. Один и тот же символ в разных контекстах может иметь разное назначение. Например, символ звездочка * используется для обозначения операции умножения и для обозначения

указателя. Студенты, имеющие недостаточно глубокие знания, не понимают смысла, например, следующей конструкции char *ch и при ответе на соответствующий вопрос отвечают, что звездочка является обозначением операции умножения. Простейшие размышления могут позволить дать совсем другой, правильный, ответ, т.к. тип не может умножаться на значение переменной.

Похожая ситуация складывается и при записи конструкций, в которых встречается знак процента %. В зависимости от контекста этот знак может обозначать операцию определения остатка от деления одного целого числа на другое целое, а может являться управляющим символом в строке управления при обращении к функциям ввода/вывода. Во втором случае символ % показывает, что следующая за ним последовательность символов должна рассматриваться как спецификация формата.

В языке С используется несколько иной смысл понятия «оператор», что отражается в причислении к операторам и тех языковых средств, которые управляют последовательностью вычислений. Полный список операторов, их приоритеты и их влияние на порядок вычислений представлен в [5]. Следует обратить внимание также на то, что есть операторы, для которых естественным является порядок вычисления справа налево. К ним относят унарные операторы, а также операторы присваивания (собственно оператор присваивания «=» и его модификации «+=» «-=» и т.д.). Из них можно строить цепочки, в которых операторы выполняются последовательно от последнего к первому (справа налево).

В языке С [2], в отличие от языка Паскаль [3,6], считается, что оператор присваивания не меняет состояния процессора и присвоенное переменной значение можно использовать для дальнейших вычислений выражения, в котором он находился. Поэтому в цепочках из правосторонних операторов может находиться произвольное число операторов присваивания (например, c*=b+=a=1; или z=y=x=0;), а в «обычных» выражениях, содержащих двуместные операторы, оператор присваивания следует заключать в скобки (например, `if((f=fopen("a.txt","w")) == NULL)`. Эти отличительные свойства операторов присваивания позволяют сделать код программы более компактным, удобочитаемым и представляющим больше информации об алгоритме программы на странице редактора кода.

Одним из важнейших разделов при изучении языка С является раздел, посвященный указателям. В отличие от языка Паскаль здесь без подробного изучения указателей обойтись нельзя. Существует даже тезис, утверждающий, что в языке С все построено на указателях. При рассмотрении данного раздела студенты должны уяснить исходной положение, состоящее в том, что указатель в качестве своего значения хранит адрес некоторой области памяти. Зная адрес памяти, можно получить доступ к содержимому этой области памяти.

При изучении данного раздела необходимо добиться прежде всего четкого представления и понимания студентами того, что в объявлении int *n указателем является переменная n, а для получения доступа к содержимому области памяти, адрес которой хранит переменная n, надо использовать операцию разадресации: *n=5.

Операции, выполняемые с указателями, также имеют свои особенности в языке С, например, сложение или вычитание указателя с константой означают изменение значения адреса не на указанную константу, а на значение константы, умноженное на объем памяти, занимаемой значением, с которым связан указатель. В языке Паскаль операции над указателями фактически недопустимы, а в С допустимо вычитание указателей, сложение с константой, сравнение, преобразование типов. При вычитании указателей получается не адрес, а целая величина, показывающая фактически смещение одной области памяти относительно другой области, т.е. полученная разность делится на объем памяти, занимаемый одним элементом. В языке Паскаль операции над указателями можно выполнить, используя соответствующие подпрограммы, позволяющие сначала выделить адрес сегмента и смещения, а затем, изменив эти значения, сформировать новый адрес.

В обоих языках операции выполняются над указателями одного типа, т.е. связанными с данными одного типа. Для присваивания указателю значения указателя другого типа в обоих языках используется явное преобразования типов: в С это конструкция вида (базовый-тип-приёмника *)U, а в языке Паскаль конструкция вида тип-указателя-приёмника(U), где U - указатель на базовый тип источника. Однако в языке Паскаль тип-указателя-приёмника должен быть объявлен заранее, например:

```
type pr=^real;
var n: integer; p1:pr;
p1:=pr(@n);
```

Одним из факторов, обусловливающих важность рассмотрения указателей при изучении языка С, является тесная взаимосвязь указателей и массивов. Имя массива представляет собой не что иное, как указатель на первый элемент массива, т.е. хранит адрес первого элемента массива. Данный факт позволяет программисту самому вычислять адрес нужного элемента массива, используя адресную арифметику. Например, к элементам массива можно обращаться, указывая их индекс, а можно сделать это самостоятельно: запись a[i] (обращение к i-ому элементу массива a) означает то же самое, что *(a+i). В случае обработки двумерного массива обращение b[i][j] можно заменить на *(b+i*nn+j), где nn – количество столбцов двумерного массива.

Важным разделом при изучении программирования является раздел, связанный с организацией и использованием подпрограмм. Независимо от изучаемого языка программирования определенные сложности возникают у студентов при использовании механизмов передачи параметров в подпрограмму – по значению и по адресу (указателю). Однако если при использовании языка Паскаль студент может ограничиться знанием нескольких формальных правил, не вдаваясь глубоко в содержание механизмов передачи параметров, то в языке С необходимо четко представлять суть механизмов и уметь работать с указателями. Упомянутые правила сводятся к следующему: по адресу можно передавать только переменную, соответствующий формальный параметр должен быть объявлен как параметр-переменная (var) или выходной параметр (out), для возврата результата надо использовать передачу по адресу.

Те же правила, естественно, используются и в языке С, но все операции с указателями пользователь программирует сам. Во-первых, возвращаемое значение должно объявляться как указатель на переменную (передача по адресу), соответствующее фактическое значение параметра должно представлять собой адрес, т.е. нельзя забывать записывать обращение к адресной операции &. Во-вторых, в подпрограмме программист должен помнить, что переданное значение представляет собой адрес, а работа ведется с содержимым памяти по этому адресу, т.е. должна выполняться разадресация. В качестве примера приведем простейшую функцию сложения и умножения двух переменных:

```
void f1(float a, float b, float *s, float *p)
{*s=a+b;
 *p=a*b;}
```

Обращение к функции f1(a,b,&s,&p);

Некоторые сложности возникают у студентов при передаче в подпрограмму в качестве параметров массивов. Во-первых, в языке С массивы всегда передаются по адресу. Во-вторых, соответствующий формальный параметр можно объявить по-разному.

Например, можно записать заголовок функции как void fun1(float a[],int n), а можно как void fun1(float *a, int n). Дело в том, что для вычисления адреса очередного элемента массива достаточно знать адрес первого элемента и тип элементов массива. Транслятору известен объем памяти, занимаемый данными каждого типа. Количество элементов массива при этом роли не играет, но программист сам должен следить за тем, из скольких элементов действительно состоит массив.

Несколько сложнее обстоит дело с передачей в функции двумерных массивов. Обычно используют три варианта объявления соответствующих формальных параметров. В первом случае заголовок функции может выглядеть следующим образом:
void fun2(float a[][nn], int m, int n). Здесь важно отметить, что для правильного вычисления адреса элемента двумерного массива транслятор должен знать максимальное количество элементов строки массива, что и задается в заголовке функции значением константы nn. Данная константа заранее объявляется как глобальная константа, что несколько снижает степень универсальности подпрограммы.

Второй вариант записи заголовка функции может выглядеть следующим образом:
void fun2(float (*a)[nn], int m, int n). Первый параметр представляет собой указатель на массив из nn элементов, т.е. указатель на строку матрицы, что также позволяет транслятору правильно вычислять адреса элементов двумерного массива. Здесь, как и в первом случае nn является глобальной константой. В обоих случаях в заголовке подпрограммы можно не указывать количество элементов по первому измерению массива, т.к. это не требуется для корректного вычисления адресов элементов массивов, обрабатываемых в подпрограмме. Программист сам несет ответственность за то, сколько элементов он будет рассматривать в качестве элементов массива. С этой целью и передается в подпрограмму фактическое количество элементов по каждому измерению.

Элементы передаваемых массивов могут принадлежать одному из ранее объявленных типов. Обращение к подпрограмме для представленных случаев может выглядеть следующим образом: fun2(a, m, n); До этой строки текста программы должны быть объявлены передаваемые параметры, например:

```
const int nn=20; //должна быть объявлена как глобальная
const int mm=15;
float a[mm][nn];
```

Третий вариант является более универсальным, т.к. он не требует предварительного объявления глобальной константы: void fun2(float *a, int m, int n, int nn). В этом случае программист работает в подпрограмме с многомерным массивом как с одномерным, т.е. самостоятельно вычисляет адрес каждого нужного элемента массива, например, для двумерного массива адрес вычисляется согласно выражению *(a+i*nn+j). В этом случае для вычисления адреса и корректной работы с массивом в подпрограмму надо передать фактическое количество элементов по каждому измерению (для матрицы – количество строк и столбцов) и максимальное количество элементов по каждому измерению, начиная со второго (для матрицы – максимальное количество столбцов, заданное при объявлении статического массива). Обращение к функции в третьем случае может выглядеть следующим образом; fun2(&a[0][0],m,n,nn);

Подобный способ работы с матрицами произвольных размеров и измерений, основанный не на вычислении указателей, представляющих элементы матрицы, а на вычислении соответствующих индексов одномерного массива, совмещенного с матрицей, возможен и в языке Паскаль [3]. Правда в этом случае придется использовать нетипизированный параметр, представляющий массив, и отказаться от автоматического контроля за выходом за пределы массива во время выполнения программы.

Наряду с преобразованием типов указателей и массивов разной размерности следует рассмотреть более общий вопрос преобразования данных разных типов. В языке С для решения этой задачи может использоваться операция явного преобразования типов (type), т.е. перед преобразуемым выражением в скобках записывается новый тип. В языке Паскаль преобразование типов требует больших знаний и усилий. Преобразуемое значение передается в подпрограмму, где соответствующий параметр должен быть объявлен как нетипизированный, в самой подпрограмме осуществляется преобразование типа путем указания имени нового типа, за которым в скобках записывается преобразуемое значение (переданное в подпрограмму). Пример иллюстрирует обработку матрицы как одномерного массива с одновременным преобразованием типа элементов:

```
{$R-}
const mm=20;nn=20;
type      tm=array[1..1] of real;
     matr=array[1..mm,1..nn] of integer;
  var  m1:matr;
```

```
procedure pech(var m1;m,n,nn:integer);
var i,j,k:integer;
begin
  for i := 1 to m do
 begin
   for j := 1 to n do
   begin
     k:=(i-1)*nn+j;
     write(tm(m1)[k]:5);
   end;
   writeln;
  end;
end;
```

Разный уровень знаний и умений требуется также при работе с динамическими массивами. Если в языке Паскаль достаточно научиться правильно использовать библиотечную функцию SetLength, которая выделяет память в соответствии с указанным количеством элементов массива (изменяет размер массива), то в языке С дело обстоит несколько сложнее. Здесь можно использовать библиотечные функции malloc, calloc, realloc или операции выделения памяти new и освобождения памяти delete. В обоих случаях требуется достаточно глубокое понимание программируемой последовательности действий и умения работать с указателями. Например, выделение памяти для хранения динамической матрицы может быть выполнено следующим образом:

```
float **a; //объявляется переменная, являющаяся указателем на массив указателей
a=new float*[m]; //выделение памяти для хранения указателей на строки матрицы
  for (i=0;i<m;i++) //выделение памяти для храниения элеметов строк матрицы
        a[i]=new float[n];

for (i=0;i<m;i++)
        delete []a[i]; //освобождение памяти, занимаемой элементами строк
                                    // матрицы
  delete []a;      //освобождение памяти, занимаемой массивом указателей на
                          //строки матрицы
```

Заключение

Таким образом, авторы рассмотрели основные отличительные особенности записи и использования конструкций языков программирования, которые надо принимать во внимание преподавателям, осуществляющим переход от обучения студентов языку Паскаль к языку С. Учет этих особенностей, позволит улучшить качество преподавания и сократить количество ошибок, допускаемых студентами при написании программ, особенно теми, кто в предшествующий период изучал язык Паскаль.

Список литературы

[1]. 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.

[2]. Алексеев Ю.Е., Куров А.В. Практикум по программированию на языке С в среде VS C++. Ч.1. Электронное учебное издание. М.: МГТУ им. Н. Э. Баумана. 2011. 100 с. Режим доступа: http://wwwcdl.bmstu.ru/iu7/PraktikumC.html (дата обращения: 3.09.2016)

[3]. Алексеев Ю.Е., Кучеренко И.К., Тассов К.Л. Турбо Паскаль – файлы и модули пользователей: Методические указания к практикуму по программированию / Под ред. Б.Г. Трусова. М.: Изд-во МГТУ им. Н. Э. Баумана. 1995. 56 с.

[4]. Ваулин А.С., Семин В.М. Алгоритмические и психологические аспекты при изучении дисциплины “Информатика”. // Инженерный вестник. Эл. журнал. МГТУ им. Н.Э. Баумана. 2016. № 5. Режим доступа: http://engsi.ru/doc/842870.html (дата обращения: 3.09.2016)

[5]. Керниган Б.В., Ритчи Д.М. Язык программирования С. 2-е изд., пер. и доп. / Пер. с англ. М.: Вильямс. 2016. 288 с. [Brian W. Kernighan, Dennis M. Ritchie. C Programming Language (2nd Ed.). 2012. 274 p.]

[6]. Культин Н.Б. Delphi 6. Программирование на Object Pascal. СПб.: БХВ-Петербург. 2001. 528 с.

[7]. Пахомов Ю.И. C/C++ и MS Visual C++ 2012 для начинающих. 2-е изд. СПб.: БХВ-Петербург. 2015. 518 с.