Лекция № 3

Операторы потокового ввода/вывода

В программе С (и С++), как и в программах написанных на других языках программирования особое и важное место занимают операторы ввода/вывода. Стандартный язык **С** содержит целую группу операторов ввода и операторов вывода в стандартный поток, в файлы и строки. Эти функции позволяют вводить и выводить данные согласно формату, указанному программистом.

В объектно-ориентированное расширение языка С++ включено целое семейство классов, позволяющее в программе создавать объекты, выполняющие операции ввода/вывода. Простейшие операторы ввода с клавиатуры и вывода на экран это объекты потокового ввода/вывода `cin` и `cout`. Эти объекты предопределены в системе программирования и готовы к использованию сразу после подключения соответствующего заголовочного файла `iostream.h`.

Использование этих объектов упрощает выполнение операций ввода/вывода. Для использования этих операторов необходимо подключить заголовочный файл:

```
#include <iostream.h>
```

и в дальнейшем перечислять переменные в потоке ввода или вывода соответственно. Примеры использования операторов потокового ввода/вывода:

```
cin>>aux;              // вводится число в переменную aux
cout<<"Привет мир!\n";     // вывод строки текста и переход на новую строку
cout<<aux<<tmp<<ddd<<endl;     // вывод переменных и переход на новую строку
```

Оператор вывода `cout` выводит данные подряд. То есть числа, выведенные в последнем операторе, на экране будут изображены единым набором идущих подряд цифр. Если среди выводимых чисел есть отрицательные и вещественные, то эта цепочка цифр будет разбавлена знаками минус и точка. Но, тем не менее, читать такой результат очень затруднительно. Поэтому вывод данных требуется “форматировать”. Естественно, что такой сложный агрегат как потоковый ввод/вывод имеет средства форматирования. Но в простейшем случае, не прибегая к средствам форматирования, можно улучшить вид вывода результата, разделяя вывод чисел выводами символов *пробел* или *табуляция*. То есть результат вывода последнего оператора будет выглядеть лучше, если его записать как:

```
cout<<aux<<'␣'<<tmp<<'␣'<<ddd<<endl; // вывод переменных aux, tmp,ddd через пробел
```

Для принудительного перехода на новую строку при выводе большого количества данных можно использовать символ ‘`\n`’ или эквивалентную константу `endl`.

Переменные производных типов. Массивы и строки

При всей мощи современной компьютерной техники и разнообразии развитых систем программирования сегодня задачи приходится решать с помощью ручки и бумаги. При наличии достаточных навыков работы в математических пакетах такие задачи могут решаться с использованием этих математических пакетов, как на эффективных заменителях бумаги. И только когда такие вычисления приходится проводить многократно над различными данными, а тем более единообразной обработке должны подвергаться целые наборы данных или набор данных должен обрабатываться как единое целое, то следует нести затраты на разработку программ. В последних случаях такой наборы данных требуется хранить в оперативной памяти, для доступа операторов программы к ним. Таким средством в языке программирования С (или С++) являются массивы.

Массивы

Языки программирования, в том числе и С (или С++), помимо переменных базовых типов имеют переменные производных типов, являющие агрегированием базовых переменных на различных принципах. Так, например, набор данных одного типа можно объединить под одним именем и обращаться к ним по упрощенной схеме и подвергать однообразной обработке, используя, например, операторы повторения. Такими переменными являются массивы.

Определение. Под *массивом* понимают набор данных, состоящий из переменных одного типа, имеющих единое имя. Обращаться к каждой из ячеек (индивидуально к каждой переменной) необходимо по ее номеру.

Описание массива выполняется следующим образом:

```
<тип элемента>    имя [количество];
```

При этом `<тип элемента>` представляет собой описание типа ячейки массива, то есть представляет собой один из базовых типов, а `количество` — константа или константное выражение целого типа, определяющее количество элементов в массиве или *длину* этого массива. Необходимо помнить, что индекс первого элемента в массиве есть 0, то есть если массив описан как массив длиной из N элементов, то первый элемент имеет индекс 0, а последний N–1. Пример:

```
float mass [100];
```

описывает массив `mass` из 100 элементов, каждая ячейка которого имеет тип `float`, при этом элементы индексируются в диапазоне от 0 до 99.

Обращение к ячейкам массива выполняется по имени массива и индексу элемента. В качестве индекса может использовать константа, константное выражение, переменная или выражение целого типа, например, `mass [i]`.

В математике основными объектами являются векторы и матрицы. Массивы являются программным средством реализации векторов. Средством реализации и хранения матриц в программах написанных на языке С (или С++), являются двумерные массивы. Описание двумерного массива и обращение к его элементам выполняется аналогично. Описание двумерного массива выполняется следующим образом:

```
<тип элемента>    имя [количество строк][количество колонок];
```

Обращение к элементу двумерного массива выполняется следующим образом:

```
имя [номер строки][номер колонки].
```

Пример описания двумерного массива:

```
double      matr [100][200];
```

описывает двумерный массив `matr` из 100 строк и 200 элементов в каждой строке (или 200 колонок), каждая ячейка которого имеет тип `double`, при этом строки индексируются в диапазоне от 0 до 99, а элементы в стоке от 0 до 199. Обращение к ячейкам такого массива, аналогично одномерному, выполняется по имени массива и индексу строки и колонки (или элемента). В качестве индекса так же может использовать константа, константное выражение, переменная или выражение целого типа, например, `matr [i][j]`.

Для обработки элементов массивов часто используют операторы повторения или даже несколько таких операторов, вложенных друг в друга.

Строки

Язык программирования С (или С++) не имеет специальных типов переменных для хранения текстовой информации. Для хранения и обработки таких данных используют одномерные массивы, ячейка которых имеет тип `char`. Описание такого массива выполняется по общим правилам:

```
char  имя [количество];
```

Обращаться к любому символу в таком массиве можно аналогично по индексу элемента: `имя [индекс]`.

Специфичной особенностью таких массивов является то, что полезная информация, хранящаяся в массиве (то есть текст), заканчивается ячейкой с нулевым кодом ‘\0’. Это связано с тем, что в отличие от обычных массивов, текст, хранящийся в массиве, не занимает весь массив, а признаком завершения полезной информации является символ с кодом ‘\0’. Отсюда при работе со строками следует придерживать следующих правил:

- массив-строку описывают размером, исходя из максимального размера текста, который будет помещен в этот массив в ходе работы программы; при этом длина массива должна быть больше максимального размера текста;
- минимальная длина массива-строки должна равняться длине максимального текста плюс один; то есть зарезервировать место под хранение текста максимальной длины и под завершающий символ ‘\0’;
- при обработке массивов-строк необходимо четко соблюдать, чтобы текст завершался кодом ‘\0’.

Для обработки строк текста, в том числе и копирования, в языке программирования С (или С++) применяют соответствующие функции.

Оператор повторения

Для реализации повторяющихся процессов в языках программирования используются *операторы повторения* или *операторы цикла*. В языке С (и С++) используются операторы повторения трех типов: `while`, `do…while`, `for`.

Оператор while

Оператор повторения `while` имеет вид:

```
while (выражение) оператор;
```

Порядок работы такого оператора: вычисляется *выражение* и проверяется — если его значение отлично от нуля (не равно **0**), выполняется *оператор*. Этот процесс повторяется до тех пор, когда значение *выражения* станет равным **0**.

Как и в условном операторе `if` так и в операторе повторения `while` *выражение* может быть логическим, например `a>b`, или состоять из нескольких логических выражений соединенных операциями ‘и’ или ‘или’. Если *значение выражения* становится равным ‘ложь’, то выполнение оператора повторения `while` прекращается и управление передается на оператор, следующий за оператором `while`. То есть по-другому, работу оператора `while` можно сформулировать следующим образом: выполнять *оператор* пока *выражение* ‘истинно’.

Также *выражение* может быть арифметическим выражением с целочисленным результатом. Равенство значения такого выражения нулю прекращает работу оператора. То есть в этом случае работу оператора `while` можно сформулировать следующим образом: выполнять *оператор* пока *выражение* отлично от нуля.

Иногда *оператор*, указанный в цикле, называют *телом цикла*, а конструкцию `while(выражение)` — *заголовком цикла*. Тело цикла, аналогично условному оператору, может быть составным оператором:

```
while (выражение)
 {оператор_1;
  оператор_2;
  …
  оператор_3;
 }
```

Графическое отображение оператора `while` в виде блок-схемы приведено на рис. 1. Из графического изображения можно сделать вывод, что условный оператор `if` в сокращенной форме очень похож по действию с оператором `while`. То есть оператор, тело цикла, выполняется, если выражение ‘истинно’ или отлично от нуля, как и в операторе `if`. Отличие в том, что тело цикла

выполняется многократно, пока не добьется от *выражения* значения ‘ложь’ или равенства нулю. И упаси нас бог, чтобы этого не произошло — получим бесконечный цикл.

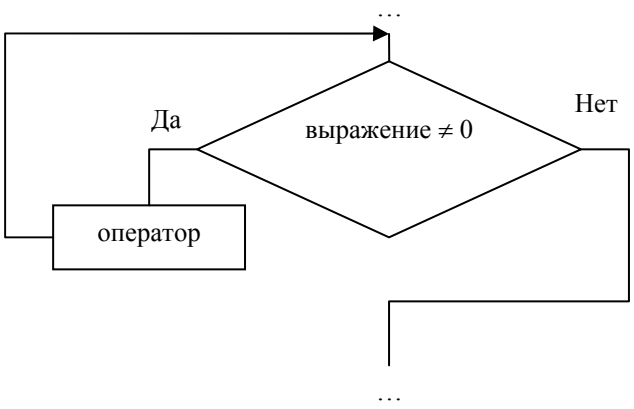

Рис. 1. Блок-схема оператора `while`

Замечания.

1. Необходимо помнить, что сначала вычисляется *выражение*, а затем уже выполняется *тело цикла*.
2. Необходимо помнить, что тело цикла может не выполняться. Так, если при первом вычислении значение `выражения` есть ‘ложь’ (или равно нулю), то тело цикла не будет выполнено ни разу, а управление сразу передается на оператор, следующий за оператором повторения `while`.
3. Тело цикла (операторы, перечисленные в нем) должно влиять на выражение, то есть изменять его значение. Иначе получится бесконечный цикл.

Пример.

```
a = 5; b = 6; c = 100;
while (a < c)
 {a += b;  // увеличивается значение ‘a’ пока оно меньше ‘c’
 }  // тело цикла один оператор – скобки можно не писать
```

Оператор do…while

Оператор `do…while` в общем то эквивалентен оператору `while`, но проверка условия выполняется после выполнения тела цикла. Таким образом, в цикле

```
do оператор
while (выражение);
```

сначала выполняется оператор, а затем проверяется значение выражения. Используется реже, но иногда более приемлем, чем оператор `while`.

Графическое отображение оператора `do…while` в виде блок-схемы приведено на рис. 2.

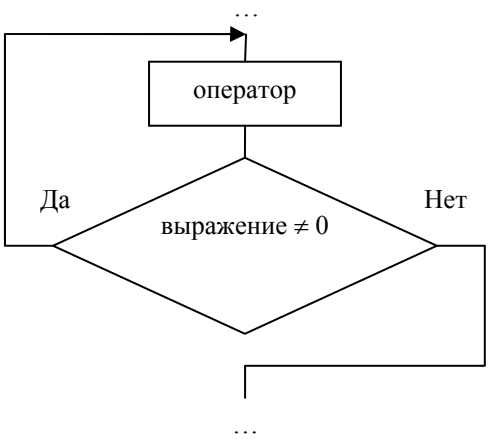

Рис. 2. Блок-схема оператора `do…while`

Замечания.

1. Необходимо помнить, что сначала выполняется *тело цикла*, а затем уже вычисляется *выражение*.
2. Необходимо помнить, что тело цикла выполняться хотя бы раз. Так как первоначально выполняется тело оператора, то, естественно, ничто не может этому воспрепятствовать. А уж после вычисляется значение выражения. И если значение `выражения` есть ‘ложь’ (или равно нулю), то тело цикла не будет более выполняться, а управление передается на оператор, следующий за оператором повторения `do…while`.
3. Тело цикла (операторы, перечисленные в нем) должно влиять на выражение, то есть изменять его значение. Иначе получится бесконечный цикл.

Пример.

```
a = 5.0, b = 1.0, c = 100.0;
do
 {a += b;  // такой же пример, но проверка после выполнения тела цикла
 } while (a <= c);
```

Оператор for

Оператор `for` удобен для записи циклов, в которых задаются начальная установка, шаг и условие выхода. Оператор повторения `for` имеет вид:

```
for (выражение1; выражение 2; выражение3) оператор;
```

Оператор повторения `for` можно развернуть в эквивалентную последовательность с использованием оператора `while`:

```
выражение1;
while (выражение2)
{оператор;
выражение3;}
```

Пример. Программа подсчета суммы элементов

```
const N = 15;
double sou [N], summ;
int i, j;
for (i = 0; i < N; i++)
  cin >> sou [i];
summ = 0;
for (i = 0; i < N; i++)
  summ += sou [i];
cout << "Сумма = " << summ << endl;
cout << "Среднее = " << summ / N << endl;
```

Графическое отображение оператора `for` в виде блок-схемы приведено на рис. 3. Из графического изображения можно сделать вывод, что условный оператор `if` в сокращенной форме очень похож по действию с оператором `while`. То есть оператор, тело цикла, выполняется, если выражение ‘истинно’ или отлично от нуля, как и в операторе `if`. Отличие в том, что тело цикла выполняется многократно, пока не добьется от *выражения* значения ‘ложь’ или равенства нулю. И упаси нас бог, чтобы этого не произошло — получим бесконечный цикл.

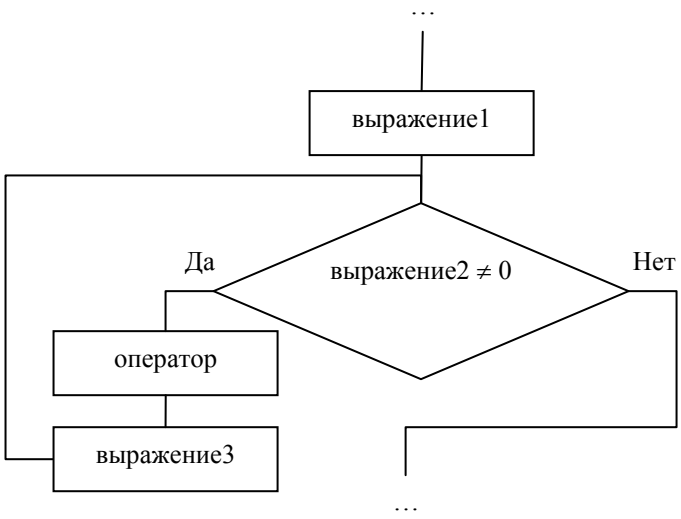

Рис. 3. Блок-схема оператора `for`

Любое из трех `выражений` оператора `for` или даже *все* могут быть опущены. При этом позиция пропущенного выражения отделяется знаком ‘;’. Если `выражение2`, по которому осуществляется выход из цикла, пропущено, то его значение считается не равным нулю и цикл будет повторяться бесконечно. Так, оператор

```
for ( ; ; ) {операторы}
```

представляет собой бесконечный цикл, выход из которого может выполняться с помощью операторов `break, goto` или `return`.

В отличие от других языков программирования, например, Фортрана и Ада, переменная цикла сохраняет свое значение независимо от того, каким образом завершился цикл. Возможно изменение параметров цикла в теле цикла. На `выражения` в операторе `for` не накладывается почти никаких ограничений, они могут быть даже не арифметические. Как и в других операторах, каждое из `выражений` может состоять из нескольких выражений, разделенных запятыми. Такая группа `выражений`, разделенных запятой, является единым выражением и выполняется слева направо, а тип и значение такого выражения совпадают с типом и значением самого правого оператора.

Замечания.

1. Необходимо помнить, что сначала вычисляется *выражение*, а затем уже выполняется *тело цикла*.
2. Необходимо помнить, что *тело цикла* и `выражение3` могут не выполняться. Так, если при первом вычислении значение `выражения2` ‘ложь’ (или равно нулю), то тело цикла не будет выполнено ни разу, а управление сразу передается на оператор, следующий за оператором повторения `for`.
3. Тело цикла (операторы, перечисленные в нем) и/или `выражение3` должно влиять на `выражение2`, то есть изменять его значение. Иначе получится бесконечный цикл.
4. Любое из трех `выражений` оператора `for` или даже *все* могут быть опущены.
5. Если `выражение2` опущено, то его значение считается *не равным* нулю и цикл будет повторяться бесконечно. Выход из цикла необходимо осуществлять с помощью операторов `break, goto` или `return`.

Тело цикла операторов повторения `while`, `do…while` и `for` может содержать любые операторы, в том числе и другие операторы повторения. В этом случае их называют вложенными операторами повторения, отражая тот момент, что один оператор повторения, является составной частью другого оператора повторения.

Оператор переключения (выбора)

Оператор *выбора* или *переключения* обеспечивает ветвление по нескольким направлениям и имеет вид:

```
switch (выражение)
   оператор;
```

Порядок работы оператора выбора следующий: вычисляется *выражение*, тип значения которого должно быть целое. *Оператор* обычно является составным оператором (то есть группой операторов, заключенных в блоковые скобки), в котором отдельные операторы могут быть помечены ключевым словом `case` и константным выражением:

```
case <константное выражение> : оператор;
```

То есть в развернутом виде оператор выбора записывается следующим образом:

```
switch (выражение)
 {
  case <константное выражение1> :  оператор1;
                                   оператор2;
                                   …
                                   операторM;
  case <константное выражение2> :  операторM+1;
                                   операторM+2;
                                   …
                                   операторN;
  case <константное выражение3> :  операторN+1;
                                   операторN+2;
                                   …
                                   операторK;
 {
```

Константное выражение — это *целая* или *символьная* константа (например, 1 или ‘A’) либо *выражение*, *составленное из* таких *констант* с использованием операций. *Оператор*, помеченный константным выражением, и *операторы*, следующие за ним, выполняются, если целое значение *выражения* в `switch` равно значению этого константного выражения. При этом выполняются все операторы до окончания *составного оператора* или до одного из операторов перехода `break, goto` или `return`, если такие есть в составном операторе и выполняются. Если же значение выражения *не равно ни одному* из константных выражений, управление передается на оператор, следующий за оператором выбора `switch`, то есть ни один из операторов, входящих в составной оператор, не исполняется.

Полная форма оператора выбора `switch` имеет вид:

```
switch (выражение)
 {
  case <константное выражение1> :  оператор1;
                                   оператор2;
                                   …
                                   операторM;
  case <константное выражение2> :  операторM+1;
                                   операторM+2;
                                   …
                                   операторN;
  default :                        операторN+1;
                                   операторN+2;
                                   …
                                   операторK;
 {
```

Отличие от сокращенной формы оператора выбора: один из операторов помечен меткой `default`.

Замечание. Меткой `default:` можно пометить не более одного переключателя.

В отличие от неполной формы оператора выбора `switch`, здесь, если значение выражения *не равно ни одному* из константных выражений, управление передается на оператор, помеченный меткой `default`.

Пример. Использование оператора переключения

```
i = random (21) - 10;  // генерирование числа в диапазоне -10÷10
switch (i)  // печать в зависимости от значения переменой 'i'
 {
  case 0 : cout << i << " : Ноль" << endl;
  case 1 : cout << i << " : Единица" << endl;
  case -1 : cout << i <<" : Минус единица" << endl; break;
  default : if (i < 0) cout << i << " : Отрицательное" << endl;
            else cout << i << " : Положительное" << endl;
 }
```

Оператор завершения и перехода

В некоторых случаях, при написании текста программы, необходимо осуществить переход не на следующий оператор, а на оператор, расположенный достаточно далеко. Примерами такого перехода являются: а) прерывание цикла, по результатам некоторой проверки внутри тела цикла, хотя *выражение*, указанное в заголовке цикла, еще истинно; б) завершение выполнения функции. Операторами завершения и перехода являются:

- `break;` — прекращает исполнение объемлющего оператора цикла или оператора переключения. Управление передается на оператор, следующий за оператором цикла или оператором переключения.
- `continue;` — осуществляет переход на начало объемлющего цикла, то есть *прерывает* выполнение *тела цикла* и передает управление на выполнение следующей итерации. При этом есть небольшое отличие в поведении для различных операторов повторения:
  - для операторов `while` и `do…while` исполнение оператора `continue` означает передачу управления на проверку условия, то есть вычисление `выражения` и проверку его значения;
  - для оператора `for` — передачу управления на переадресацию, то есть на вычисление `выражения3` и вычисление `выражения2` и проверку его значения.
- `return;` — завершает исполнение функции и управление передается в вызывающую программу. Если в вызывающую программу должно быть возвращено значение, то используется оператор: `return (выражение);`. В объектно-ориентированном расширении языка C++ так же допускается конструкция: `return выражение;`. Выражение должно иметь тот тип, который указан в описании заголовка функции, как тип возвращаемого значения.
- `exit (выражение);` — завершение программы и возврат кода завершения в операционную систему. Выражение имеет целый тип. Если программа завершается нормально, то должно выполнение программы должно завершаться выражением с нулевым значением или просто с кодом ноль, то есть, например оператором `exit (0);`.
- `goto метка;` — оператор перехода на оператор, помеченный *меткой*. Этот используется редко, но иногда бывает очень полезен. Метка представляет собой идентификатор, завершающийся знаком ':', располагающийся перед соответствующим оператором.

Использование операторов перехода позволяет строить достаточно сложные алгоритмы более простыми конструкциями и правильно оформлять тело функции.