Урок 7. Функции

Цель работы: приобрести навыки в решении задач с использованием функций.

Определение функции. Обращение к функции. В C/C++ используется лишь один тип подпрограмм — функция. Функция является основной программной единицей в C, минимальным исполняемым программным модулем. Всякая программа обязательно включает в себя основную функцию с именем main.

Если в программе используются и другие функции, то они выполняют роль подпрограмм.

Рассмотрим пример. Требуется составить программу нахождения наибольшего значения из трех величин — max(a, b, c). Для ее решения можно использовать вспомогательный алгоритм нахождения максимального значения из двух, поскольку справедливо равенство: max(a, b, c) = max(max(a, b), c). Программа решения этой задачи с использованием вспомогательной функции.

Пример 1.

```
#include  <iostream>
#include  <clocale>

using namespace std;

//Определение  вспомогательной  функции
int  MAX(int x, int y)
{
if(x > y) return x;
else  return y;
}

//Основная  функция
int main( )
{
setlocale(LC_ALL,"RUS");
int a, b, c, d;
cout << "Введите a, b, c:";
cin >> a >> b >> c;
d = MAX(MAX(a, b), c);
cout << "\nmax (a, b, c) = " << d;
}
```

Формат определения функции следующий:

```
тип имя_функции  (спецификация_параметров)
{тело_ функции}
```

Тип функции — это тип возвращаемого функцией результата. Если функция не возвращает никакого результата, то для нее указывается тип void.

Имя функции — идентификатор, задаваемый программистом или main для основной функции.

Спецификации параметров — это либо «пусто», либо список имен формальных параметров функции с указанием типа для каждого из них.

Тело функции — это либо составной оператор, либо блок.

Здесь мы впервые встречаемся с понятием блока. Признаком блока является наличие описаний программных объектов (переменных, массивов и т.д.), которые дейст-

вуют в пределах этого блока. Блок, как и составной оператор, ограничивается фигурными скобками.

В C/C++ действует правило: тело функции не может содержать в себе определения других функций. Иначе говоря, недопустимы внутренние функции. Из всякой функции возможно обращение к другим функциям, однако они всегда являются внешними по отношению к вызывающей.

Оператором возврата из функции в точку ее вызова является оператор return. Он может использоваться в функциях в двух формах:

```
return;  или  return выражение;
```

В первом случае функция не возвращает никакого значения в качестве своего результата. Во втором случае результатом функции является значение указанного выражения. Тип этого выражения должен либо совпадать с типом функции, либо относиться к числу типов, допускающих автоматическое преобразование к типу функции.

Оператор return может в явном виде отсутствовать в теле функции. В таком случае его присутствие подразумевается перед закрывающей тело функции фигурной скобкой. Такая подстановка производится компилятором. Формат обращения к функции (вызова функции) традиционный:

```
имя_функции(список_фактических_праметров)
```

Однако в C/C++ обращение к функции имеет своеобразную трактовку: обращение к функции — это выражение. В этом выражении круглые скобки играют роль знака операции, для которой функция и фактические параметры (аргументы) являются операндами. Приоритет операции «скобки» самый высокий, поэтому вычисление функции в выражениях производится раньше других операций.

Между ***формальными*** и ***фактическими*** параметрами при вызове функции должны соблюдаться правила соответствия по последовательности и по типам. **Фактический** параметр — это выражение того же типа, что и у соответствующего ему формального параметра. Стандарт языка C допускает автоматическое преобразование значений фактических параметров к типу формальных параметров. В C++ такое преобразование не предусмотрено. Поэтому в дальнейшем мы будем строго следовать принципу соответствия типов.

Необходимо усвоить еще один важнейший принцип, действующий в C: передача параметров при вызове функции происходит только по значению. Поэтому выполнение функции не может изменить значения переменных, указанных в качестве фактических параметров.

В C/C++ возможны функции с переменным числом параметров. Примером таких функций являются библиотечные функции printf( ) и scanf( ).

Прототип функции. Оказывается, совсем не обязательно было в предыдущем примере помещать полное определение функции MAX( ) перед основной частью программы. Вот другой вариант программы, решающей ту же самую задачу.

Пример 2.

```
#include  <iostream>
#include  <clocale>

using namespace std;

//Прототип функции MAX
```

```
int MAX(int, int);

//Основная функция
int main( )
{
setlocale(LC_ALL,"RUS");
int a, b, c, d;
cout << "Введите  a, b, c: ";
cin >> a >> b >> c ;
d = MAX(MAX(a, b), c);
cout << "\nmax(a, b, c) = " << d ;
}
//Определение функции MAX
int MAX(int x, int y)
{
if(x > y) return x;
else return y;
}
```

Здесь использован прототип функции. Прототипом называется предварительное описание функции, в котором содержатся все необходимые сведения для правильного обращения к ней: имя и тип функции, типы формальных параметров. В прототипе имена формальных параметров указывать необязательно, хотя их указание не является ошибочным. Можно было написать и так, как в заголовке определения функции:

```
int  MAX(int x, int y);
```

Точка с запятой в конце прототипа ставится обязательно!

Можно было бы записать прототип и в теле основной функции наряду с описаниями других программных объектов в ней. Вопрос о размещении описаний связан с понятием области видимости, который будет обсужден немного позже.

В следующей программе приводится пример использования функции, которая не имеет параметров и не возвращает никаких значений в точку вызова. Эта функция рисует на экране строку, состоящую из 80 звездочек.

Пример 3.

```
#include  <iostream>
#include  <clocale>

using namespace std;


//Прототип функции  line
void  line();

//Основная  функция
int main()
{
setlocale(LC_ALL,"RUS");
line( );  //Вызов функции line
}
//Определение  функции line
void  line()
{
int i;
for(i = 0; i < 80; i++) cout << "*";
}
```

Программа для вычисления наибольшего общего делителя для суммы, разности и произведения двух чисел на C++.

Пример 4.

```
#include  <iostream>
#include  <cmath>
#include  <clocale>

using namespace std;

int NOD2(int, int);

int  main()
{
      setlocale(LC_ALL,"RUS");
      int a, b, Rez;
      cout << "a = ";  cin >> a;
      cout << "b = ";  cin >> b;
      Rez = NOD2(NOD2(a + b, abs(a - b)), a * b);
      cout << "NOD равен " << Rez;
}
      int NOD2(int M, int N)
      {
      while(M != N)
            {
            if(M > N)  M = M - N;
            else  N = N - M;
            }
      return  M;
      }
```

Может возникнуть вопрос: если основная часть программы является функцией, то кто (или что) ее вызывает? Ответ состоит в следующем: программу вызывает операционная система при запуске программы на исполнение. Функция main( ) не обязательно должна иметь тип void. Гораздо чаще она имеет тип int. Например, она может возвращать операционной системе целое значение 1 в качестве признака благополучного завершения программы и 0 — в «аварийном» случае. Обработка этих сообщений будет осуществляться системными средствами.

Использование библиотечных функций. Библиотечными называются вспомогательные функции, хранящиеся в отдельных файлах. Стандартные библиотеки входят в стандартный комплект системы программирования на C/C++. Кроме того, программист может создавать собственные библиотеки функций. Ранее уже говорилось о том, что для использования стандартных функций необходимо подключать к программе заголовочные файлы соответствующих библиотек. Делается это с помощью директивы претранслятора #include с указанием имени заголовочного файла. Например, #include <stdio.h>. Все заголовочные файлы имеют расширение h (от английского header). Теперь должно быть понятно, что эти файлы содержат прототипы функций библиотеки. На стадии претрансляции происходит подстановка прототипов перед основной функцией, после чего компилятор в состоянии контролировать правильность обращения к функциям. Сами программы, реализующие функции, хранятся в форме объектного кода и подключаются к основной программе на стадии редактирования связей (при работе компоновщика).

Рассмотрим программу решения следующей задачи: зная декартовы координаты вершин выпуклого четырехугольника, вычислить его площадь (рис. 1).

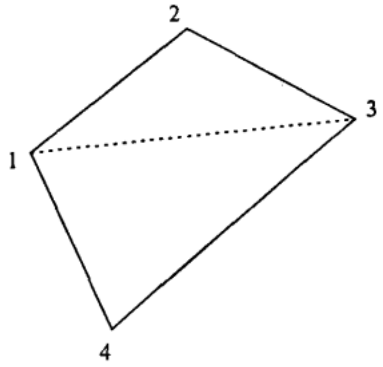

Рис.1

Математическое решение этой задачи следующее. Обозначим координаты вершин четырехугольника так: (x1, y1), (x2, y2), (x3, y3), (x4, y4). Площадь четырехугольника можно вычислить как сумму площадей двух треугольников. В свою очередь, площадь каждого треугольника вычисляется по формуле Герона. Для применения формулы Герона нужно найти длины сторон. Длина стороны между первой и второй вершинами вычисляется по формуле:

$$L_{12} = \sqrt{(x1 - x2)^2 \quad + \quad (y1 - y2)^2}$$

Аналогично вычисляются длины других отрезков.

Таким образом, для решения основной задачи — вычисления площади четырехугольника — требуется вспомогательный алгоритм вычисления площади треугольника, для которого, в свою очередь, необходим вспомогательный алгоритм вычисления длины отрезка по координатам концов.

Ниже приведена программа решения поставленной задачи.

Пример 5.

```
//Площадь выпуклого  четырехугольника
#include <iostream>
#include <cmath>
#include <conio.h>
#include <clocale>
typedef double D;

using namespace std;

      // Переименование  типа double
D Line(D, D, D, D);           // Прототип функции Line
D Geron(D, D, D, D, D, D);    // Прототип функции Geron

//Основная функция
int main()
{
setlocale(LC_ALL,"RUS");
D x1, y1, x2, y2, x3, y3, x4, y4, S1234;
void clrscr( );
cout << "y1 = ";  cin >> y1;  cout << "x1 = ";  cin >> x1;
cout << "y2 = ";  cin >> y2;  cout << "x2 = ";  cin >> x2;
cout << "y3 = ";  cin >> y3;  cout << "x3 = ";  cin >> x3;
cout << "y4 = ";  cin >> y4;  cout << "x4 = ";  cin >> x4;
```

```
S1234 = Geron(x1, y1, x2, y2, x3, y3) + Geron(x1, y1, x3, y3, x4, y4);
cout << "Площадь четырехугольника = " << 31234;
}
//Определение функции Line
D Line(D a, D b, D c, D d)
{
return  sqrt((a - c)*(a - c)+(b - d)*(b – d));
}
//Определение функции Geron
D Geron(D a1, D a2, D b1, D b2, D c1, D c2)
{
D p, ab, bc, ca;
ab = Line(a1, a2, b1, b2);
bc = Line(b1, b2, c1, c2);
ca = Line(c1, c2, a1, a2);
p = (ab + bc + ca)/2;
return sqrt(p*(p - ab)*(p - bc)*(p - ca));
}
```

В этой программе используются функции из трех стандартных библиотек с заголовочными файлами iostream, cmath и conio.h. С первыми двумя мы уже встречались раньше. Третья библиотека (файл conio.h) содержит функции, предназначенные для управления выводом на экран в символьном режиме. В программе из этой библиотеки используется функция clrscr( ) — очистка экрана.

Еще одним новым элементом в приведенной программе является строка

```
typedef double D;
```

Служебное слово typedef представляет собой спецификатор типа, позволяющий определять синонимы для обозначения типов. В результате в рассматриваемой программе вместо длинного слова double для обозначения того же самого типа можно употреблять одну букву D. Данное описание действует глобально и распространяется как на основную, так и на вспомогательные функции.

Обратим внимание на еще одно обстоятельство. В функции Geron имеются обращения к функции Line, а в основной функции — обращение только к функции Geron.

Для компилятора важно, чтобы перед вызывающей функцией присутствовал или прототип, или определение вызываемой функции.

Поэтому если из данной программы убрать прототип функции Line, то ошибки не будет. Но если одновременно с этим поменять местами определения функций Line и Geron, то компилятор выдаст сообщение об ошибке.

Рекурсивные определения функций. В языках C/C++ допускается рекурсивное определение функций. Проиллюстрируем определение рекурсивной функции на классическом примере вычисления факториала целого положительного числа.

```
long  Factor(int n)
{
if  (n < 0) return 0;
if  (n == 0) return 1;
return  n * Factor(n - 1);
}
```

В случае если при вызове функции будет задан отрицательный аргумент, она вернет нулевое значение — признак неверного обращения.

Передача значений через глобальные переменные. Областью действия описания программного объекта называется часть программы, в пределах которой действует (учитывается) это описание. Если переменная описана внутри некоторого блока, то она локализована в этом блоке и из других блоков, внешних по отношению к данному, «не видна». Если описание переменной находится вне блока и предшествует ему в тексте программы, то это описание действует внутри блока и называется глобальным. Глобальная переменная «видна» из блока. Например:

```
double  x;
int  func1( )
{
int  y; ...
}
int  main( )
{
float  y; ...
}
```

Переменная x является глобальной по отношению к функциям func1 , main и, следовательно, может в них использоваться. В функциях func1 и main имеются локальные переменные с одинаковым именем y. Однако, это разные величины, никак не связанные друг с другом. Поскольку переменная x является общей для обеих функций, то они могут взаимодействовать через x друг с другом.

При обращении к функции передача значений возможна как через параметры, так и через глобальные переменные. В следующей программе решается уже рассматриваемая нами задача получения наибольшего из трех значений.

Пример 6.

```
int  z; //Описание глобальной  переменной
void  MAX(int x, int y)
{
if  (x > y) z = x; else z = y;
}
#include  <iostream>
int  main( )
{
int a, b, c;
cout << "a = ";  cin >> a;
cout << "b = ";  cin >> b;
cout << "c = ";  cin >> c;
MAX(a, b);
MAX(z, c);
cout << "max = " << z;
}
```

Результат выполнения функции MAX заносится в глобальную переменную z, которая «видна» также и из основной функции. Поэтому при втором обращении эта переменная играет одновременно роль аргумента и результата.

Классы памяти. Под всякую переменную, используемую в программе, должно быть выделено место в памяти ЭВМ. Выделение памяти может происходить либо на стадии компиляции (компоновки) программы, либо во время ее выполнения. Существуют 4 класса памяти, выделяемой под переменные:

- автоматическая (ключевое слово auto);
- внешняя (extern);
- статическая (static) ;
- регистровая (register).

Под глобальные переменные выделяется место во внешней памяти (не нужно думать, что речь идет о магнитной памяти; это оперативная память класса extern). Глобальную переменную можно объявить либо вне программных блоков, либо внутри блока с ключевым словом extern. Обычно это делается в тех случаях, когда программный модуль хранится в отдельном файле и, следовательно, отдельно компилируется.

Пусть, например, основная и вспомогательная функции хранятся в разных файлах.

Пример 7.

Файл 1:

```
...
int  var
int  main( )
{
var = 5;
func( );
cout << var;
}
```

Файл 2:

```
...
void  func( )
{
extern int  var;
var = 10 * var;
}
```

Здесь обмен значениями между основной и вспомогательной функцией func( ) происходит через общую глобальную переменную var, для которой во время компиляции выделяется место во внешнем разделе памяти. В результате выполнения данной программы на экран выведется число 50.

Локальные переменные, объявленные внутри блоков, распределяются в автоматической памяти, работающей по принципу стека.

Выделение памяти происходит при входе выполнения программы в блок, а при выходе из блока память освобождается. Ключевое слово auto писать необязательно (подразумевается по умолчанию).

Статическая память выделяется под переменные, локализованные внутри блока, но в отличие от автоматической памяти не освобождается при выходе из блока. Таким образом, при повторном вхождении в блок статическая переменная сохраняет свое прежнее значение. Пример объявления статической переменной:

```
f( )
{
static int  schet = 10;
  ...
}
```

Инициализация статической переменной происходит только при первом вхождении в блок. Если инициализация явно не указана, то переменной автоматически присваивается нулевое начальное значение. Статические переменные можно использовать, например, для организации счетчика числа вхождений в блок.

Регистровая память выделяется под локальные переменные. Регистры процессора — самый быстрый и самый маленький вид памяти. Они задействованы при вы-

полнении практически всех операций в программе. Поэтому возможность распоряжаться регистровой памятью лучше оставить за компилятором.

Наконец рассмотрим пример, в котором используются различные способы описания переменных.

Пример 8.

```
int  var;          //описана внешняя переменная var
int  main( )
{
extern int  var;  // та же внешняя переменная var
}
func1( )
{
extern int  var1;//новая внешняя переменная var1
//внешняя var здесь также видна
}
func2( )
{
...                //здесь переменная var видна, а переменная
}                  // var1 не видна
int  var1;         //глобально описана переменная var1
func3( )
{
int  var;          //здесь  var  —  локальная  переменная,
//видна  внешняя  переменная  var1
}
func4( )
{
auto  int  var1;  //здесь  var1  —  локальная  переменная,
//видна  внешняя  глобальная  var
}
```

Использование указателей для передачи параметров функции. Рассматривая ранее правила использования функций, мы обращали внимание на то, что в языке С возможна только односторонняя передача значений фактических параметров из вызывающей программы к формальным параметрам вызываемой функции. Возвращаемое значение несет сама функция, используемая в качестве операнда в выражении. Отсюда, казалось бы, следует неукоснительное правило: в процессе выполнения функции не могут изменяться значения переменных в вызывающей программе. Однако это правило можно обойти, если в качестве параметров функции использовать указатели.

В следующем примере функция swap( ) производит обмен значениями двух переменных величин, заданных своими указателями в аргументах.

```
void  swap(int *a, int *b)
{
int  c;
c = *a; *a = *b; *b = c;
}
```

Если в основной программе имеется следующий фрагмент:

```
int  x = 1,  y = 2;
swap(&x, &y);
printf("x = %d  y= %d", x, y);
```

то на экран будет выведено:

```
x = 2  y = 1
```

т. е. переменные x и y поменялись значениями.

Передача параметров здесь происходит по значению, только этими значениями являются указатели. После обращения к функции указатель a получил адрес переменной x, указатель b — адрес переменной y. После этого переменная x в основной программе и разадресованный указатель *a в функции оказываются связанными с одной ячейкой памяти; так же — y и *b.

Таким образом, можно сделать вывод о том, что использование указателей в параметрах функции позволяет моделировать работу процедур.

Массив как параметр функции. Обсудим эту тему на примерах.

Пример 1. Составим программу решения следующей задачи. Дана вещественная матрица A[M][N]. Требуется вычислить и вывести евклидовы нормы строк этой матрицы. Евклидовой нормой вектора называют корень квадратный из суммы квадратов его элементов:

$$L = \sqrt{\sum_{i=1}^{n} x_i^2} .$$

Если строку матрицы рассматривать как вектор, то данную формулу надо применить к каждой строке. В результате получим U чисел.

Определение функции вычисления нормы произвольного вектора:

```
double  Norma(int  n,  double  X[])
{
int  i;
double  S = 0;
for(i = 0;  i < n;  i++)  S += X[i] * X[i];
return  sqrt(S);
}
```

Заголовок этой функции можно было записать и в такой форме:

```
double  Norma(int  n,  double  *X)
```

В обоих случаях в качестве второго параметра функции используется указатель на начало массива. Во втором варианте это более очевидно, однако оба варианта тождественны. При вызове функции Norma( ) в качестве второго фактического параметра должен передаваться адрес начала массива (вектора).

Рассмотрим фрагмент основной программы, использующей данную функцию для обработки матрицы размером 5x10.

```
int  main( )
{
double A[5][10];
int  i;
//Ввод матрицы
//Вычисление и вывод нормы строк
for(i = 0; i < 5; i++)
cout << "Норма" << i <<" M строки =" << Norma(10, A[i]);
}
```

В обращении к функции второй фактический параметр A[i] является указателем на начало i-й строки матрицы A.

Пример 2. Заполнить двумерную матрицу случайными целыми числами в диапазоне от 0 до 99. Отсортировать строки полученной матрицы по возрастанию значений. Отсортированную матрицу вывести на экран.

```
#include <iostream>
#include <iomanip>
#include <conio.h>
#include <stdlib.h>
#include <clocale>

using namespace std;

const n = 5;                    //Глобальное  объявление константы

//Прототипы функций
void  Matr(int M[][n]);
void  Sort(int, int X[]);

//Основная  программа
int  main( )
{
setlocale(LC_ALL,"RUS");
int  i, j, A[n][n];
void clrscr( );
cout << "\n" << "Матрица  до  сортировки:" << "\n";
Matr(A);
for(i = 0; i < n; i++)  Sort(n, A[i]);
cout << "\n" << "Матрица a после  сортировки:  " << "\n";
for(i = 0;  i < n;  i++)
{
for(j = 0; j < n; j++)
cout << setw(6) << A[i][j];
cout << endl;
}
}
//  Функция  сортировки  вектора
void  Sort (int k, int X[])
{
int i, j, Y;
for(i = 0;  i < k - 1;  i++)
for(j = 0;  j < k - i - 1;  j++)
if (X[j] > X[j + 1])
{
Y = X[j];
X[j] = X[j + 1];
X[j + 1] = Y;
}
}
//Функция  заполнения  матрицы  и  вывода  на  экран
void  Matr(int M[][n])
{
int  i, j;
void randomize( );            //Установка  датчика  случайных  чисел
for( i = 0; i < n; i++)
{
```

```
        for (j = 0; j < n; j++)
        {
        M[i][j] = rand( )%100;
        cout << setw(6) << M[i][j];
        }
cout << endl;
}
}
```

Обратите внимание на прототип и заголовок функции Matr( ). В них явно указывается вторая размерность параметра матрицы. Первую тоже можно указать, но это необязательно. Как уже говорилось выше, двумерный массив рассматривается как одномерный массив, элементами которого являются массивы (в данном случае — строки матрицы). Компилятору необходимо «знать» размер этих элементов. Для массивов большей размерности (3, 4 и т.д.) в заголовках функций необходимо указывать все размеры, начиная со второго.

При обращении к функции Matr( ) фактическим параметром является указатель на начало двумерного массива A, а при обращении к функции Sort( ) — указатели на начало строк. В итоге тестирования программы получен следующий результат.

Матрица до сортировки:

46	23	57	35	18
8	48	68	4	70
56	98	16	71	40
70	84	66	67	11
20	44	37	57	38

Матрица после сортировки:

18	23	35	46	57
4	8	48	68	70
16	40	56	71	98
11	66	67	70	84
20	37	38	44	57

Упражнения

1. Найти ошибку в программе:

```
#include <iostream.h>
int main( )
{
int  a = 1, b = 2, c;
c = sum(a, b);
cout << c;
}
int  sum(int x, int y)
{
return x+y;
}
```

2. Определить результат выполнения программы:

```
#include  <iostream>
void  mul(int, int);
int  S;
```

```
int  main( )
{
int  a = 2, b = 3;
mul(a, b);
a = 2 * S;
mul(a, b);
cout << S;
}
void  mul(int  x, int  y)
{
S = x * y;
}
```

3. Составить программу для вычисления площади кольца по значениям внутреннего и внешнего радиусов, используя функцию вычисления площади круга.
4. Даны три целых числа. Определить, сумма цифр которого из них больше. Подсчет суммы цифр организовать через функцию.
5. Составить функцию, определяющую, является ли ее целый аргумент простым числом. Использовать эту функцию для подсчета количества простых чисел в последовательности из десяти целых чисел, вводимых с клавиатуры.
6. Описать рекурсивную функцию stepen(x, n) от вещественного x ($x \neq 0$) и целого n, которая вычисляет величину x^n согласно формуле

$$x^n = \begin{cases} 1 \text{ при } n = 0 \\ \frac{1}{x^{-n}} \text{ при } n < 0 \\ x * x^{n-1} \text{ при } n > 0 \end{cases}$$

7. Даны натуральные числа n и m; найти НОД(n, m). Составить рекурсивную функцию вычисления НОД, основанную на соотношении НОД(n, m) = НОД (m, r), где r — остаток от деления n на m(n > m).

Информационные источники:

1. **Семакин И.Г., Шестаков А.П.** Основы программирования: Учебник.- М.: Мастерство, 2002.- 432 с.