ФУНКЦИИ ПОЛЬЗОВАТЕЛЯ

В отличие от языка *Pascal* в Си нет разделения на ***подпрограммы-процедуры и подпрограммы-функции***, здесь вся программа строится только из ***функций***, т.е. все написанные подпрограммы создаются в одном стиле и являются функциями, представляющими отдельный программный модуль, к которому можно обратиться, чтобы передать исходные данные и получить результаты работы.

Минимальная программа, написанная на Си, содержит, как известно, функцию ***main***.

Декларация функции

Как любой объект программы на языке Си, пользовательские функции необходимо ***декларировать***.

Хорошим стилем программирования является выполнение следующих действий:

1) ***описание*** функции в форме ***прототипа***, которое выполняется обычно в глобальной области, или до первого ее использования;

2) ***определение*** (реализация) функции, т.е. запись полного текста функции, которое может располагаться в исходном файле в любом месте (до использования, или после).

Прототип функции сообщает компилятору о том, что далее будет ее реализация.

В ***прототипе*** задаются свойства функции – тип возвращаемого результата (если он имеется), имя функции, список типов параметров.

Общий вид декларации прототипа функции:

Тип возвращаемого результата ***Имя функции*** (***Список параметров***);

В ***Списке*** перечисляются типы параметров, а имена переменных можно не указывать, т.к. компилятор их не обрабатывает.

Описание прототипа дает возможность компилятору проверить соответствие типов и количества параметров при вызове этой функции.

Пример:

double Fun (int, int, int, double);

Определение функции – это ее полный текст, включающий ***заголовок*** (первая строка) и код:

Тип возвращаемого результата ***Имя функции*** (***Список параметров***)

{ – Начало функции

Код функции

return Выражение;

} – Конец функции

Декларация функции в проекте простой структуры может быть выполнена до ее вызова ***только Определением***.

Тип возвращаемого результата определяет тип ***Выражения***, значение которого возвращается в точку ее вызова при помощи оператора

return Выражение; (*return* - возврат).

Выражение преобразуется к указанному в заголовке функции ***Типу*** и передается в точку вызова.

Тип возвращаемого функцией значения может быть любым базовым типом, или созданным ранее типом Пользователя.

Если функция не возвращает значения, указывается ***void***. При досрочном выходе из функции *void* используется оператор ***return***;

В конце кода функции *void* оператор *return*; можно не ставить.

Если тип функции не указан, то по умолчанию устанавливается тип ***int***.

Список параметров в заголовке функции состоит из перечня не только типов, но и идентификаторов объектов, которые требуется передать в функцию при ее вызове.

В ***объявлении*** и в ***определении*** одной и той же функции типы и порядок следования параметров должны совпадать.

Тип возвращаемого результата и типы параметров определяют свойство функции.

Функция может не иметь параметров, но ***круглые скобки необходимы***.

Если у функции отсутствует список параметров, то можно указать тип ***void***, например, простейшая форма основной функции

```
void  main ( void )
{
    . . .
}
```

В функции может быть несколько операторов ***return***, но может и не быть ни одного, например в функции ***void*** (это определяется потребностями алгоритма). В последнем случае возврат в вызывающую программу происходит после выполнения последнего оператора функции.

Пример реализации функции поиска наименьшего из двух ***int*** значений:

```
int  Min (int x, int y)
{
    return (x < y) ? x : y;
}
```

Ее прототип: **int Min (int, int);**

Эту функцию можно реализовать иначе с использованием оператора ***if*** :

```
int  Min (int x, int y)
{
    if (x < y) return x ;
    else return  y ;
}
```

Или:

```
int  Min (int x, int y)
{
    int  res;
    if (x < y) res = x ;
    else res = y;
    return res;
}
```

Вызов функции

Каждая функция – это ***отдельный блок***, вход в который возможен только через ее вызов.

Для вызова функции нужно указать ее имя, за которым в круглых скобках через запятую перечислить список передаваемых ей ***аргументов***.

Вызов функции может находиться в любом месте программы, где по синтаксису допустимо выражение того типа, который она возвращает.

Общий вид вызова функции

Имя функции (***Список аргументов***)

В ***Списке аргументов*** могут быть константы, переменные и выражения, которые перед вызовом функции вычисляются.

Аргументы должны совпадать со списком параметров вызываемой функции по количеству и порядку следования.

Типы аргументов при передаче в функцию преобразуются, если это возможно, к типу параметров в заголовке функции, иначе – сообщение об ошибке.

Связь между функциями выполняется через ***аргументы*** и ***возвращаемые функциями значения***.

Связь можно осуществить и через ***глобальные переменные***, но это не рекомендуется, т.к. нужно стремиться к тому, чтобы функции в программе были максимально независимыми.

Все величины, переданные в функцию как аргументы и описанные внутри функции – ***локальны***. Областью их действия является функция.

При вызове функции выделяется память под локальные переменные, которая при выходе из функции освобождается, поэтому значения локальных переменных между вызовами одной и той же функции не сохраняются.

Передача данных в функцию и получение ее результатов

В языке Си аргументы при вызове функции обычно передаются с помощью имен переменных (***по значению***). Для них выделяется память, в которую заносятся значения фактических аргументов при вызове функции.

При передаче аргументов проверяется соответствие типов, выполняются преобразования, после чего функция использует и может изменять эти значения.

Но при выходе из функции измененные значения теряются, т.к. время жизни и зона видимости локальных параметров определяется только кодом функции.

Для получения результатов из функции используется оператор ***return***, позволяющий получить ***только одно значение***, указанного в заголовке типа, или воспользоваться передачей параметров ***по адресу***.

При передаче данных по адресу (с помощью указателей, или ссылок) в память заносятся ***копии адресов*** аргументов и функция, осуществляя доступ к ячейкам памяти по этим адресам, дает возможность не только изменить переданные (исходные) значения аргументов, но и сделать их доступными в точке вызова.

Исходя из вышесказанного следует, что

– параметры, переданные ***по значению***, т.е. с использованием их идентификаторов, могут быть только ***входными***;

– параметры, переданные ***по адресу***, т.е. с использованием указателей, или ссылок (копии адресов) являются как ***входными*** так и ***выходными*** (возвращаемыми в точку вызова).

Функции, возвращающие значение, желательно использовать в правой части выражений, иначе возвращаемый результат будет утерян.

Например стандартная функция **_getch();**

1) если мы используем **только** ее вызов

_getch();

то она выполняет задержку выполнения программы до нажатия любой клавиши;

2) если мы используем возвращаемый ею результат, то она выполнит не только ожидание нажатия клавиши, но и вернет символ (код) нажатой клавиши для дальнейшей обработки, т.е. например

char code; code = **_getch( ) ;**

if (code == 'N') break;

Пример 1. Вычисление суммы двух ***int*** величин:

```
int  Sum ( int x,  int y ) {          - Реализация функции
     return x + y;
}
```

Часть кода с обращением к функции ***Sum***:

```
int  a = 2, b = 3, s;
s = Sum (a, b);          - Сохранив значение
cout   << “ Sum = " << s << endl;
cout   << " Sum = " << Sum (2*a, 3*b) << endl;
```

- Вызов функции без сохранения ее результата

Иначе функцию ***Sum*** можно записать:

```
int  Sum ( int x,  int y )
{
 int  s;
 s = x + y;
 return s;
}
```

Пример 2. Функция вычисления суммы и разности двух ***int*** значений (3-й параметр – указатель):

```
int  Fun ( int x,  int y,  int *r )  {        - Реализация
    *r = x – y;
    return x + y;
}
```

Часть кода с обращением к функции ***Fun***:

```
int a = 2, b = 3, summa, razn;
summa = Fun ( a, b, &razn );
cout   << " A + B = "  << summa
     << "  A – B = " << razn << endl;
```

Третий параметр передается **по адресу с помощью указателя,** поэтому он является и выходным значением, т.е. его измененное значение **будет известно и в точке вызова этой функции.**

Пример 3. Функция вычисления суммы и разности двух ***int*** значений (3-й параметр – ссылка):

```
int  Fun ( int x,  int y,  int  &r )  {      - Реализация
    r = x – y;
    return x + y;
}
```

Часть кода с обращением к функции ***Fun***:

```
int a = 2, b = 3, summa, razn;
summa = Fun ( a, b, razn );
cout   << " A + B = "  << summa
     << "  A – B = " << razn << endl;
```

Третий параметр передается **по ссылке** (копия адреса), поэтому он является и выходным значением, т.е. его измененное значение **будет известно и в точке вызова этой функции.**

С помощью **адресов** (по указателю и по ссылке) из функции в точку ее вызова можно передать столько результатов, сколько необходимо по алгоритму.

Передача массивов

1. Найти минимум в одномерном массиве (статич.)
Функция может иметь следующий вид

```
int  Min1 ( int size, double x[ ] ) {          //  double *x
int m = 0;                  - Поиск по индексу
for ( int i = 1; i < size; ++i )
     if ( x[i] < x[m] )  m = i;
return m;
}
```

Вызов этой функции для ***double*** массива ***a*** размером ***n*** (int ***imin*** – искомый ***индекс***)

```
imin = Min1 ( n, a );

cout << “Min = “ << a[imin] << “ index = “ << imin << endl;
```

2. Найти минимум в двумерном массиве (динамич.)

Функция может иметь следующий вид

```
double Min2 ( int row, int col, double **x ) {
double m = x[0][0];
for(int i = 0; i < row; ++i)
    for(int j = 0; j < col; ++j)
        if(x[i][j] < m) m = x[i][j];
return m;
}
```

- Поиск по значению

Вызов этой функции для ***double*** массива ***a*** размером ***n*** на ***m*** (double ***min*** – искомое ***значение***)

```
min = Min2 ( n, m, a );
cout << “Min = “ << min << endl;
```

Операция typedef

Любому типу данных, как стандартному, так и определенному Пользователем, можно задать новое имя с помощью операции ***typedef***:

typedef Тип Новое имя **;**

Введенный таким образом новый тип используется аналогично стандартным типам, например, введя пользовательские типы:

typedef unsigned int ***UINT***; – ***UINT*** новое имя

typedef double ***Real***;

Декларации объектов введенных типов будут иметь вид

UINT **i, j;** - две переменные типа ***unsigned int***

Real **x, y;** - две переменные типа ***double***

Рассмотренная операция упростит использование указателей на функции, которые рассмотрим далее.

Указатели на функции

В языке Си имя функции является ***константным указателем*** на начало выделенной для нее памяти и не может быть использована в левой части операции присваивания.

Но имеется возможность декларировать указатели на функции, т.е. указатели-переменные.

Рассмотрим методику работы с указателями на функции.

Как и любой объект языка Си, указатель на функцию необходимо объявить:

Тип (*****Имя_Указателя***) (***Список***);

- декларируется ***Указатель***, который можно устанавливать на функции, имеющие ***те же свойства***, т.е. возвращающие результат указанного ***Типа*** и имеющие указанный ***Список параметров***.

Наличие ***первых круглых скобок обязательно***, т.к. без них – это декларация функции, которая возвращает указатель.

Например:

***int fun1* (*int*);**

- прототип функции, имеющей ***int*** результат и один ***int*** параметр;

***int * fun2* (*int*);**

- прототип функции, результат которой – ***указатель*** типа ***int*** и ***int*** параметр.

Декларация ***Указателя***, который можно установить на функции вида ***fun1***

***int* (**p_fun1*) (*int*);**

А ***Указатель***, который можно установить на функции вида ***fun2***

***int ** (**p_fun2*) (*int*);**

Пример

***double* (**p_f*) (*int, double*);**

объявление указателя ***p_f***, который можно устанавливать на функции, возвращающие ***double*** результат и имеющие два параметра: ***int*** и ***double***.

Чтобы установить ***Указатель*** на конкретную функцию, достаточно ему присвоить имя этой функции:

Имя_Указателя = Имя_Функции;

Например, для некоторой функции с прототипом:

***double f1* (*int, double*);**

установим указатель ***p_f*** на функцию ***f1*** , т.е.

p_f = f1;

После чего функцию ***f1*** можно вызвать следующими способами:

f1 (21, 1.5); – по ее имени;

p_f (21, 1.5); – по имени указателя (по адресу).

Основное назначение указателей на функции – это обеспечение возможности передачи конкретных функций в качестве формальных параметров в другие функции.

Классы памяти
и область действия объектов

При объявлении кроме типа можно использовать необязательный атрибут «***Класс памяти***», который определяет время и область действия объекта. Он может принимать значения: ***auto***, ***register*** (динамическая память), ***extern***, ***static*** (статическая память).

Область действия объекта по умолчанию зависит от места их объявления и может быть локальной (внутренней) или глобальной (внешней).

Имеется три основных участка программы, где объявляются переменные:

– внутри функции (блока);

– в заголовке функции (описание параметров);

– вне функции.

Локальные – переменные, объявленные внутри функции (блоке) и описанные в заголовке функции. ***Глобальные*** – переменные, описанные вне функции.

Область действия локальных данных – от места объявления до конца ***функции*** (*блока*), в которой они объявлены, включая все вложенные блоки.

Область действия глобальных данных – от места объявления до конца ***файла***, в котором они объявлены.

Если класс памяти не указан явно, он определяется компилятором по месту объявления объекта.

Время жизни объекта может быть ***постоянным*** – в течение выполнения программы, и ***временным*** – в течение выполнения функции (блока).

Переменные, объявленные внутри функций – локальные и никакая другая функция не имеет прямого доступа к ним.

Каждая локальная переменная существует только в блоке кода, в котором она объявлена, и уничтожается при выходе из него. Эти переменные располагаются в стековой области памяти и называются ***автоматическими***.

Локальные объекты по умолчанию имеют атрибут ***auto.***

Если хотят показать, что переменные не надо искать вне функции, то используют явное описание класса, например:

```
void main(void)  {
        auto int  max,  i,  n;
            ...
}
```

Объекты, размещаемые в статической памяти, объявляются с атрибутом ***static***.

Время их жизни – постоянное.

При описании глобального объекта атрибут ***static*** определяет область применения (действия) этого объекта только в пределах остатка текущего файла.

Описанная вне функции переменная – глобальна и по умолчанию имеет атрибут ***extern*** (*внешняя*).

Если глобальная переменная используется функциями одного файла, то атрибут ***extern*** может быть опущен.

Если глобальная переменная используется функциями разных файлов, то ***extern*** необходимо указать, что позволит использовать внешнюю переменную, даже если она определяется позже в этом или другом файле.

Для внешних и статических переменных гарантируется их обнуление.

Автоматические и регистровые переменные имеют неопределенные начальные значения (*мусор*).

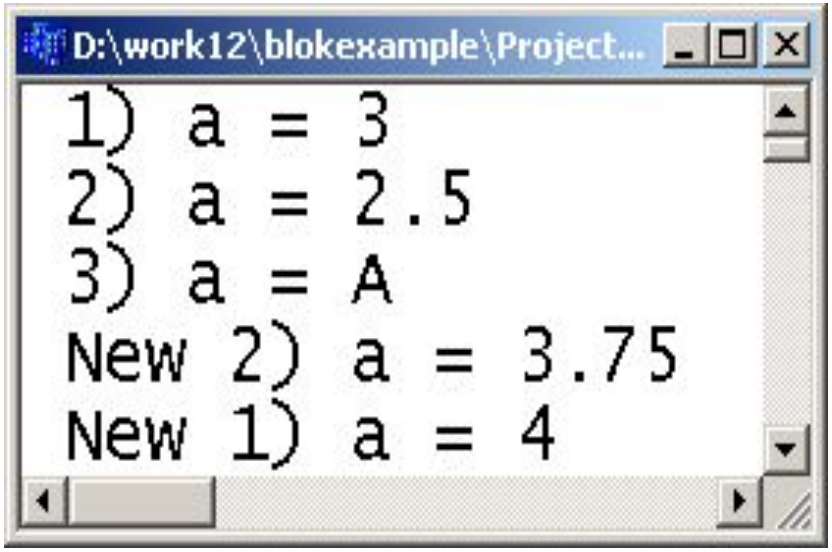
Параметры функции являются локальными объектами, поэтому ***глобальная*** переменная, передаваемая в функцию, становится ***локальной***.

Описанная внутри функции переменная может иметь такое же имя как и глобальная, но она ***локальна*** и не имеет никакой связи с глобальной.

Приведем некоторые примеры.

1. Использование блоков

```
. . .
  int a = 3;
  cout << "1) a =" <<a<<endl;
  {
double a = 2.5;
cout << "2) a =" <<a<<endl;
{
      char a = 'A';
      cout << "3) a =" <<a<<endl;
}
cout << "New 2) a =" <<a+1.25<<endl;
   }
   cout << "New 1) a =" << ++a<<endl;
```



2. Объявление переменных в заголовке оператора ***for*** :

```
for ( int i = 0; i < n; ++i )
        for ( int j = 0; j < m; ++j )
        . . .
```

Дальнейшее использование переменных ***i***, ***j*** приведет к ошибке!

3. Объявление переменных в ***case*** оператора ***switch***:

```
switch (a) {
        case  2:  {                - Необходимо
                double m = 5;          создать
                    . . .              БЛОК
    }  break;
    case  0:   . . .   break;
        . . .
}
```

В связи с рассмотренными примерами, объявление ***всех*** переменных рекомендуется писать в начале функций.