Функции

Лекция 7

Вопрос 1

Объявление и определение функций

Определение

Функция – это именованная последовательность описаний и операторов, выполняющая какое-либо законченное действие. Функция может принимать параметры и возвращать значение.

Особенности

- Любая программа на C++ состоит из функций, одна из которых должна иметь имя **main** (с нее начинается выполнение программы). Функция начинает выполняться в момент *вызова*.
- Любая функция должна быть *объявлена* и *определена*. Как и для других величин, объявлений может быть несколько, а определение только одно.
- Объявление функции должно находиться в тексте раньше ее вызова для того, чтобы компилятор мог осуществить проверку правильности вызова.

Структура функции

Объявление функции (прототип, заголовок, сигнатура) задает ее имя, тип возвращаемого значения и список передаваемых параметров.

Определение функции содержит, кроме объявления, *тело* функции, представляющее собой последовательность операторов и описаний в фигурных скобках:

[класс] тип имя ([список параметров])

{ тело функции }

Класс функции

С помощью необязательного модификатора **класс** можно явно задать область видимости функции, используя ключевые слова extern и static:

extern – глобальная видимость во всех модулях программы (по умолчанию);

static – видимость только в пределах модуля, в котором определена функция.

Тип возвращаемого значения

Тип возвращаемого функцией значения может быть любым, кроме массива и функции (но может быть указателем на массив или функцию). Если функция не должна возвращать значение, указывается тип **void**.

Параметры функции

- Список параметров определяет величины, которые требуется передать в функцию при ее вызове.
- Элементы списка параметров разделяются запятыми.
- Для каждого параметра, передаваемого в функцию, указывается его тип и имя.

Параметры функции

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

На имена параметров ограничений по соответствию не накладывается, поскольку функцию можно вызывать с различными аргументами.

Вызов функции

- Для вызова функции в простейшем случае нужно указать ее имя, за которым в круглых скобках через запятую перечисляются имена передаваемых аргументов.
- Вызов функции может находиться в любом месте программы, где по синтаксису допустимо выражение того типа, который формирует функция.
- Если тип возвращаемого функцией значения не **void**, она может входить в состав выражений или, в частном случае, располагаться в правой части оператора присваивания.

Пример 1

Функция, возвращающая сумму двух целых величин:

```
#include <iostream.h>
int sum(int a, int b);      // объявление функции
int main(){
    int a = 2, b = 3, c, d;
    c = sum(a, b);      // вызов функции
    cin >> d;
    cout<< sum(c, d);      // вызов функции
    return 0;
}

int sum (int a, int b){     // определение функции
    return (a + b);
}
```

Пример 2

Функция, выводящая на экран поля переданной ей структуры:

```
#include <iostream.h>
struct Worker{
    char fio[30];
    int code;
    double zarpl;
};

void print_worker(Worker);          //объявление функции

int main()
{
    Worker mas[10];
    ...  /* формирование массива mas */
```

Пример 2 (продолжение)

```
    for (int i = 0; i<10; i++)
     print_worker(mas[i]);   // вызов функции
    return 0;
}

void print_worker(Worker w)          // определение
    функции
{
    cout<< w.fio << ' ' << w.date << ' ' << w.code
    << ' ' << w.zarpl << '\n';
}
```

Локальные параметры

Все величины, описанные внутри функции, а также ее параметры, являются локальными. Областью их действия является функция.

При выходе из функции значения локальных переменных между вызовами одной и той же функции не сохраняются.

Если этого требуется избежать, при объявлении локальных переменных используется модификатор **static**:

```
static int n = 0;
```

Глобальные переменные

Глобальные переменные видны во всех функциях, где не описаны локальные переменные с теми же именами, поэтому использовать их для передачи данных между функциями очень легко.

Тем не менее это не рекомендуется, поскольку затрудняет отладку программы и препятствует помещению функций в библиотеки общего пользования.

Нужно стремиться к тому, чтобы функции были максимально независимы, а их интерфейс полностью определялся прототипом функции.

Вопрос 2

Возвращаемое значение и параметры функции

Оператор return

Механизм возврата из функции в вызвавшую ее функцию реализуется оператором

return [выражение];

Особенности

- Функция может содержать несколько операторов **return** (это определяется потребностями алгоритма).
- Если функция описана как **void**, выражение не указывается.
- Оператор **return** можно опускать для функции типа **void**, если возврат из нее происходит перед закрывающей фигурной скобкой, и для функции **main**.
- Выражение, указанное после **return**, неявно преобразуется к типу возвращаемого функцией значения и передается в точку вызова функции.

Примеры:

```
int f1(){return 1;}     // правильно

void f2(){return 1;}    // неправильно, f2 не
   должна возвращать значение

double f3(){return 1;}// правильно, 1
   преобразуется к типу double
```

Параметры функции

Параметры, перечисленные в заголовке описания функции, называются *формальными параметрами,* или просто *параметрами,* а записанные в операторе вызова функции – *фактическими параметрами,* или *аргументами.*

Способы передачи параметров

При передаче по значению в стек заносятся копии значений аргументов, и операторы функции работают с этими копиями. Доступа к исходным значениям параметров у функции нет, а, следовательно, нет и возможности их изменить.

При передаче по адресу в стек заносятся копии адресов аргументов, а функция осуществляет доступ к ячейкам памяти по этим адресам и может изменить исходные значения аргументов.

Пример

```
#include <iostream.h>
void f (int i, int* j, int& k);
int main(){
   int i = 1, j = 2, k = 3;
   cout << "i j k\n";
   cout << i << ' ' << j << ' ' << k << '\n';
   f(i, &j, k);
   cout << i << ' ' << j << ' ' << k;
   return 0:
}
```

Пример (продолжение)

```
void f(int i, int* j, int& k){
i++; (*j)++; k++;
}
```

Результат работы программы:

```
i  j  k
1 2 3
1 3 4
```

Пример (продолжение)

- Первый параметр (i) передается по значению. Его изменение в функции не влияет на исходное значение.
- Второй параметр (j) передается по адресу с помощью указателя, при этом для передачи в функцию адреса фактического параметра используется операция взятия адреса, а для получения его значения в функции требуется операция разыменования.
- Третий параметр (k) передается по адресу с помощью ссылки.

Запрет изменения параметров

Если требуется запретить изменение параметра внутри функции, используется модификатор const:

```
int f(const char*);
char* t(char* a, const int* b);
```