Тема 5.1. Функции в C++

Объявление и описание функций, передача параметров, возвращаемые значения, рекурсивные функции, перегрузка функций

Функции

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

```
[класс] тип имя ([список_параметров]) [throw (исключения)]
{
    тело функции
}
```

Объявление и определение функций

Объявление функции:

```
int sum (int a, int b);
```

Определение функции:

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

Обмен информации между функциями

При совместной работе функции должны обмениваться информацией. Это можно осуществить с помощью:

- глобальных переменных;
- через параметры;
- через возвращаемое функцией значение.

Пример функции

```
#include <iostream>
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;
}
```

Возвращаемое значение

Возврат из функции в вызвавшую
ее функцию реализуется оператором return:

```
return [выражение];
```

Примеры:

```
int function_1 {
    return 1;
}

double function_2 {
    return 1; //1 преобразуется к типу double
}
```

Параметры функции

Формальные параметры – параметры, перечисленные в заголовке описания функции.

Фактические параметры (аргументы) – параметры, записанные в операторе вызова функции .

Передача параметров функции

```
#include <iostream>
void f(int i, int* j, int& k); //описание функции
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;
}
//определение функции
void f(int i, int* j, int& k){
	i++;
	(*j)++;
	k++;
}
```

- по значению;
- по адресу:
  - с использованием указателя;
  - по ссылке.

Передача массивов в качестве параметров

```
#include <iostream>
int sum(const int* mas, const int n); //описание функции
int const n = 10;
int main{
	int marks[n] = {3, 4, 5, 4, 4};
	cout << “Сумма элементов массива: “ << sum(marks, n);
	return 0;
}
int sum(const int* mas, const int n){ //определение функции
	int s = 0;
	for (int i = 0; i < n; i++)
		s +=mas[i];
	return s;
}
```

Передача имен функций в качестве параметров

```
void f(int a){ //определение функции
    …
}
void (*pf)(int); //указатель на функцию
…
pf = &f; //указателю присваивается адрес функции
pf(10); //функция f вызывается через указатель pf
```

Параметры со значениями по умолчанию

```
int f(int a, int b = 0);
…
f(100);
f(a, 100); //варианты вызова функции f

void f_1(int, int = 100, char* = 0);
…
f_1(a);
f_1(a, 10);
f_1(a, 10, “Hello!”); //варианты вызова функции f_1
```

Функции с переменным числом параметров

```
int printf(const char*, …);
```

Пример:

```
//один параметр
    int printf(“Введите исходные данные”);
//два параметра
    int printf(“Сумма: ”, sum);
//пять параметров
    int printf(“Оценки: ”, mark_1, mark_2, mark_3, mark_4 );
```

Рекурсивные функции

Рекурсивной называется функция, которая вызывает сама себя.

Рекурсия:

- прямая;
- косвенная.

Вычисление факториала:

```
long fact (long n){
        if (n == 0 || n == 1) return 1;
        return (n * fact(n - 1));
}
```

Перегрузка функций

```
//возвращает наибольшее из двух целых
        int max(int, int);
//возвращает подстроку наибольшей длины
        char* max(char*, char*);
//возвращает наибольшее из первого параметра и длины второго
        int max(int, char*);
//возвращает наибольшее из второго параметра и длины первого
        int max(char*, int);
```

Неоднозначность может проявиться при:

- преобразовании типа;
- использовании параметров-ссылок;
- использовании аргументов по умолчанию.

Правила описания перегруженных функций

- Перегруженные функции должны находиться в одной области видимости, иначе произойдёт закрытие аналогично одинаковым именам переменных во вложенных блоках.
- Перегруженные функции могут иметь параметры по молчанию, при этом значение одного и того же параметра разных функций должны совпадать. В различных вариантах перегруженных функций может быть различное количество параметров по умолчанию.
- Функции не могут быть перегружены, если описание их параметров отличается только модификатором const или использованием ссылки.

Лекция 8. Структуры

Структуры, битовые поля

Структуры (struct)

Структура может содержать элементы разных типов. Элементы структуры называется **полями структуры.**

```
struct [имя_типа] {
    тип_1 элемент_1;
    тип_2 элемент_2;
    …
    тип_n  элемент_n;
} [список_описателей];
```

Определение структуры

```
//определение массива структур и указателя на структуру
struct {
      char fio[30];
      int date, code;
      double salary;
} staff[100], *ps;

struct Worker{ //описание нового типа Worker
      char fio[30];
      int date, code;
      double salary;
}; //конец определения структуры
Worker staff[100], *ps; //определение массива типа Worker и указателя на него
```

Инициализация структуры

Инициализация структуры:

```
struct {
    char fio[30];
    int date, code;
    double salary;
} worker = {“Иванов”, 31, 111, 3400.55};
```

Инициализация массивов структур:

```
struct complex{
    float real, im;
} compl [2][3] = {
    {{1,1}, {1,1}, {1,1}}, //строка 1
    {{2,2}, {2,2}, {2,2}} //строка 2
};
```

Доступ к полям структуры

Доступ к полям структуры при обращении к полю через имя структуры выполняется с помощью операции выбора **.** (точка).

```
Worker worker, staff[100], *ps;
worker.fio = “Иванов”;
staff[8].code = 111;
```

При обращении через указатель с помощью **->.**

```
ps -> salary = 0.12;
```

Битовые поля

Битовые поля – это особый вид полей структуры.

Описание битового поля:

```
struct Options{
    bool centerX:1;
    bool centerY:1;
    unsigned int shadow:2;
    unsigned int palette:4;
};
```