Лекция 4. Функции

Описание и использование функций. Параметры функции. Функции стандартной библиотеки С. Директивы препроцессора. Создание программ, состоящих из нескольких модулей. Области действия имен. Именованные области.

Функции

inline

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

Класс:

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

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

Функции - пример

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

Локальные статические переменные

```
#include <iostream.h>
void f(int a){
   cout << "n  m\n";
   while (a--){
      static int n = 0;
      int m = 0;
      cout << n++ << ' ' << m++ << '\n';
   }
}
int main(){ f(3);}
```

```
n m
0 0
1 0
2 0
```

```
int* f(){
   int a = 5;
   return &a;
// нельзя!
}
```

Параметры функции

```
#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 <<"до     " << i <<' '<< j <<' '<<k<<'\n';
   f(i, &j, k);
   cout <<"после " << i << ' ' << j << ' ' << k;
}
void f(int i, int* j, int& k){
   i++; (*j)++; k++;
}
```

	i	j	k
до	1	2	3
после	1	3	4

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

Пример 1 - сумма элементов массива

```
#include <iostream.h>
int sum(const int* mas, const int n);
int const n = 10;
void main(){
   int marks[n] = {3, 4, 5, 4, 4};
   cout << "Сумма эл-в: " << sum(marks, n);
}
int sum(const int* mas, const int n){
// варианты:int sum(int mas[], int n)
// или   int sum(int mas[n], int n)
// (величина n должна быть константой)
   int s = 0;
   for (int i = 0; i<n; i++) s += mas[i];
   return s;
}
```

Пример 2 - сумма элементов массива

```
...
int sum(const int *a, const int nstr, const int nstb);
void main(){
   int b[2][2] = {{2, 2}, {4, 3}};
   printf(" %d\n", sum(&b[0][0], 2, 2));

   int i, j, nstr, nstb, *a;
   printf("Введите кол. строк и столбцов: \n");
   scanf("%d%d", &nstr, &nstb);
   a = (int *)malloc( nstr * nstb * sizeof(int) );
   for (i = 0; i<nstr; i++)
      for (j = 0; j<nstb; j++)
         scanf("%d", &a[i * nstb + j]);
   printf(" %d\n", sum(a, nstr, nstb));}
```

Пример 2 - сумма элементов массива

```
int sum(const int *a, const int nstr, const int nstb){
   int i, j, s = 0;
   for (i = 0; i<nstr; i++)
      for (j = 0; j<nstb; j++)
         s += a[i * nstb + j];
   return s;
}
```

Пример 3 - сумма элементов массива

```
...
int sum(const int **a, const int nstr, const int nstb);
void main(){
   int nstr, nstb;
   cin >> nstr >> nstb;
   int **a;
   a = new int* [nstr];
   for (int i = 0; i<nstr; i++)
      a[i] = new int [nstb];

/*  формирование матрицы a */
...
   cout << sum(a, nstr, nstb);
}
```

Пример 3 - сумма элементов массива

```
int sum(const int **a, const int nstr, const int nstb){
   int i, j, s = 0;
   for (i = 0; i<nstr; i++)
      for (j = 0; j<nstb; j++)
         s += a[i][j];
   return s;
}
```

Передача имен функций в качестве параметров

```
void f( int a ){ /* ... */ }
void (*pf)( int );  // указатель на функцию

pf = &f;      // или pf = f;
pf(10);       // или (*pf)(10)
```

```
void fun(PF pf){ ... pf(10); ... }
```

```
typedef void (*PF)(int);
PF menu[]={&new, &open, &save};
menu[1](10);
```

Параметры со значениями по умолчанию

```
int f(int a, int b = 0);

void f1(int, int = 100, char* = 0);

void err(int errValue = errno);
...

f(100); f(a, 1);

f1(a); f1(a, 10); f1(a, 10, "Vasia");

f1(a,,"Vasia")   // неверно!
```

Функции с переменным числом параметров

```
int printf(const char* ...);

    printf("Введите исходные данные");
    printf("Сумма: %5.2f рублей", sum);
    printf("%d %d %d %d", a, b, c, d);
```

Для доступа к необязательным параметрам внутри функции используются макросы библиотеки va_start, va_arg и va_end, находящиеся в заголовочном файле <stdarg.h>.

Рекурсивные функции

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

Или:

```
long fact(long n){
   return (n > 1) ? n * fact(n - 1) : 1;
}
```

Перегрузка функций

```
int max(int, int);
char* max(char*, char*);
int max (int, char*);
int max (char*, int);

void f(int a, int b, char* c, char* d){
   cout << max (a, b) << max(c, d)
      << max(a, c) << max(c, b);
}
```

Неоднозначность

Неоднозначность может появиться при:

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

```
#include <iostream.h>
float f(float i){...}
double f(double i){...}
int main(){
   float x = 10.09;
   double y = 10.09;
   cout << f(x) << endl;
   cout << f(y) << endl;
   // cout << f(10) << endl; Неоднозначность!
}
```

Неоднозначность

```
#include <iostream.h>
int f(int a){return a;}
int f(int a, int b = 1){return a * b;}
int main(){
   cout << f(10, 2);
   /* cout << f(10);       Неоднозначность –
что вызывается:
f(int, int) или f(int) ? */
}
```

int f(int a, int b) {…}

int f(int a, int &b) {…}

Правила описания перегруженных функций

- Перегруженные функции должны находиться *в одной области видимости*
- Перегруженные функции могут иметь *параметры по умолчанию*, при этом значения одного и того же параметра в разных функциях 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* (например, int и const int или int и int&).

Шаблоны функций

```
template <параметры> заголовок
    { /* тело функции */ }
```

Параметры:

1. class имя
2. typename имя
3. template < … > class имя
4. описание параметра

```
template<class T1 = int, class T2 = int>
                f(...){...}
```

Шаблоны функций - пример

```
template <class Type>
   void sort_vybor(Type *b, int n){
   Type a;
   for (int i = 0; i<n-1; i++){
      int imin = i;
      for (int j = i + 1; j<n; j++)
         if (b[j] < b[imin])  imin = j;
      a = b[i]; b[i] = b[imin];
      b[imin] = a;
   }
```

Шаблоны функций - пример

```
#include <iostream.h>
template <class Type>
   void sort_vybor(Type *b, int n);
int main(){
   const int n = 20;
   int b[n];
   for (int i = 0; i<n; i++) cin >> b[i];
   sort_vybor(b, n);
   for (int i=0; i<n; i++) cout << b[i] << ' ';

   double a[] = {0.22, 117, -0.08, 0.21, 42.5};
   sort_vybor(a, 5);
   for (int i=0; i<5; i++) cout << a[i] << ' ';
}
```

Явное задание аргументов шаблона

```
template<class X, class Y, class Z> void f(Y, Z);
void g(){
    f<int, char*, double>("Vasia", 3.0);
    f<int, char*>("Vasia", 3.0);
                // Z определяется как double
    f<int>("Vasia", 3.0);
   // Y определяется как char*, а Z определяется как double
    // f("Vasia", 3.0);  ошибка: X определить невозможно
}
```

Шаблоны функций - описание параметра (4)

```
class T { /* ... */ };
int i;

template <class T, T i> void f(T t)
{
T t1 = i;         // template-parameters T and i
::T t2 = ::i;     //global namespace members T and i
}
```

```
template<const X &x, int i>
        void f(){ ... }
```

Описание параметра может быть:

- integral or enumeration type,
- pointer to object or pointer to function,
- reference to object or reference to function

Примеры параметра

```
template <int i> void f1(int a[10][i]);
template <int i> void f2(int a[i][20]);
template <int i> void f3(int (&a)[i][20]);
void g()
{
   int v[10][20];
   f1(v); //  OK:  i  deduced to be  20
   f1<20>(v);                  //  OK
   f2<10>(v);                  //  OK
   f3(v);  //  OK:  i  deduced to be  10
}
```

Специализация шаблона функции

```
void sort_vibor <int> (int *b, int n)
{
   тело специализированного метода
}
```

Функция main()

```
// без параметров:
тип main(){ ... }
// с двумя параметрами:
тип main(int argc, char* argv[]){ ... }
```

```
#include <iostream.h>
void main(int argc, char* argv[]){
for (int i = 0; i<argc; i++) cout << argv[i] << '\n';
}
```

d:\cpp\main.exe one two three

```
D:\CPP\MAIN.EXE
one
two
three
```

Функции стандартной библиотеки

Функции ввода/вывода <stdio.h> или <cstdio>.

```
FILE* fopen(const char* filename, const char* mode);
```

mode: (b/t) и

"r" — файл открывается для чтения;

"w" — открывается пустой файл для записи (если файл существует, он стирается);

"a" — файл открывается для добавления информации в его конец;

"r+" — файл открывается для чтения и записи (файл должен существовать);

"w+" — открывается пустой файл для чтения и записи (если файл существует, он стирается);

"a+" — файл открывается для чтения и добавления информации в его конец

```
FILE *f = fopen("d:\\cpp\\data.txt", "rb+");
```

Предопределенные потоки

стандартный ввод stdin

стандартный вывод stdout

стандартный вывод сообщений об ошибках stderr

стандартный дополнительный поток stdaux

стандартная печать stdprn

Функции ввода/вывода

Чтение и запись потока байтов — fread, fwrite.

Чтение символа из потока — getc, fgetc, из стандартного потока stdin — getchar.

Запись символа в поток — putc, fputc, в стандартный поток stdout — putchar.

Чтение строки из потока — fgets, из стандартного потока stdin — gets.

Запись строки в поток — fputs, в стандартный поток stdout — puts.

Форматированный ввод из потока — fscanf, из стандартного потока stdin — scanf, из строки — sscanf.

Форматированный вывод в поток — fprintf, в стандартный поток stdout — printf, в строку — sprintf.

```
int fclose(FILE*);   int feof(FILE*);

      int ferror(FILE*);
```

Работа с файлами - пример 1

```
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
int main(){
   FILE *fi, *fo;
   if((fi = fopen("d:\\c\\file.txt", "r")) == 0){
      cout << "Ошибка"; return 1;};
   if((fo = fopen("d:\\c\\binfile.out", "w+b"))==0){
      cout << "Ошибка"; return 1;};

   const int dl = 80;
   char      s[dl];
```

Работа с файлами - пример 1

```
struct{
      char  type[20];
      intopt, rozn;
      char  comm[40];
   }mon;
while (fgets(s, dl, fi)){
      // Преобразование строки в структуру:
      strncpy(mon.type, s, 19);                mon.opt =
atoi(&s[20]);
      mon.rozn = atoi(&s[25]);
      strncpy(mon.comm, &s[30], 40);
      fwrite(&mon, sizeof mon, 1, fo);
   }
   fclose(fi);
```

Работа с файлами - пример 1

```
int i; cin >> i;          // Номер записи
   fseek(fo, (sizeof mon)*i, SEEK_SET);
   fread(&mon, sizeof mon, 1, fo);
   cout  << "mon.type " << mon.type << " opt "
      << mon.opt  << " rozn "
      << mon.rozn << endl;
   fclose(fo);
   return 0;
}
```

Другие функции библиотеки

```
isalnum
isalfa
iscntrl
isdigit
isgraph
islower
isprint
ispunct
isspace
isupper
isxdigit
```

```
double atof(const char* p)
int atoi(const char* p)
long atol(const char* p)
```

копирование строк (strcpy, strncpy)
сравнение (strcmp, strncmp),
объединение строк (strcat, strncat)
поиск подстроки (strstr)
поиск вхождения символа (strchr, strrchr, strpbrk)
определение длины строки (strlen) и другие.

Математические

Использование стандартных функций

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main(){
   char s[] = "2, 38.5, 70, 0, 0, 1", *p = s;
   float m[10];
   int i = 0;
   do{
      m[i++] = atof(p);
      if (i>9) break;
   }while(p = strchr(p, ','), p++);

   for( int k = 0; k<i; k++) printf("%5.2f ", m[k]);
   return 0;
}
```

Директивы препроцессора

Директива #include:

#include <имя_файла>

“ “

Заголовочные файлы могут содержать:

- определения типов, констант, встроенных функций, шаблонов, перечислений;
- объявления функций, данных, имен, шаблонов;
- пространства имен;
- директивы препроцессора;
- комментарии.

В заголовочном файле не должно быть определений функций и данных.

Внешние объявления

Чтобы сделать доступной в нескольких модулях переменную или константу, необходимо:

- определить ее ровно в одном модуле как глобальную;

- объявить ее как внешнюю с помощью модификатора extern. Поместить это объявление либо в нужные модули, либо в заголовочный файл, который включить в нужные модули.

```
// my_header.h
extern int a;
extern double b;

// --------------------
// one.cpp
#include "my_header.h"
int a;

// --------------------
// two.cpp
#include "my_header.h"
double b;
```

Все объявления одной переменной должны быть согласованы

Директива #define

Практически каждый макрос свидетельствует о недостатке в языке, программе или программисте

используется для определения:

- *символических констант* :

#define имя текст_подстановки

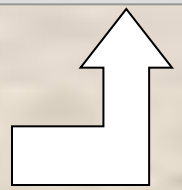
- *макросов*:

#define имя(параметры) текст_подстановки

- *символов, управляющих условной компиляцией:*

#define имя

```
#define M 1000
#define Vasia “Василий Иванович”
#define MAX(x,y) ((x)>(y)?(x):(y))
```



```
#define N(a,b) a##b /*склеить */
…
int N(bam,buk)(); /* => int bambuk(); */
```

Альтернативы макросам в C++:

const, inline, template, namespace

#undef имя

Директивы условной компиляции: #if -- #endif

Назначение:

- исключить компиляцию отдельных частей программы
- временно закомментировать фрагменты кода

```
#if конст_выражение
...
[ #elif конст_выражение
...]
[ #elif конст_выражение
...]
[ #else
]
#endif
```

```
#if OPT == 2
#include “hdr2.h”
#elif OPT == 1
#include “hdr1.h”
#elif
#include <cstdlib>
#endif
```

Директивы условной компиляции: пример

```
#if defined(__BORLANDC__) && __BORLANDC__ == 0x530
                          // Tested with BC5.3:
typedef istream_iterator<int, char, char_traits<char>, ptrdiff_t>
  istream_iter;

#elif defined(__BORLAND__)    // Tested with BC5.2:
typedef istream_iterator<int, ptrdiff_t> istream_iter;

#else                                  // Tested with VC5.0:
typedef istream_iterator<int> istream_iter;
#endif
```

```
#if 0
  int i, j;
  double x, y;
#endif
```

Директивы #ifdef и #ifndef

```
#ifdef символ
// Расположенный ниже код компилируется, если символ определен
#ifndef символ
// Расположенный ниже код компилируется, если символ не определен
```

```
#ifndef HEADER_INCLUDED
#include "myheader.h"
#define HEADER_INCLUDED
#endif
```

Стражи включения

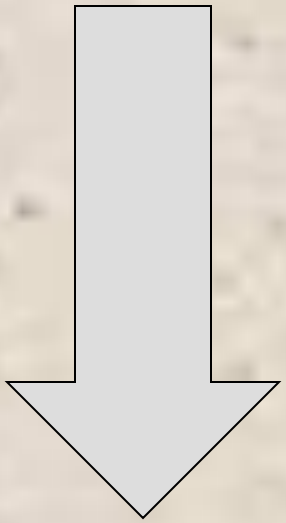
Содержимое каждого заголовочного файла:

```
#ifndef FNAME_H
#define FNAME_H
…
#endif   /* FNAME_H */
```

Директива #define - антипример

```
#define x 3
#define f(a) f(x * (a))
#undef x
#define x 2
#define g f
#define z z[0]
#define h g(~
#define m(a) a(w)
#define w 0,1
#define t(a) a
```

```
f(y+1) + f(f(z)) % t(t(g)(0) + t)(1);
g(x+(3,4)-w) | h 5) & m
(f)^m(m);
```



```
f(2 * (y+1)) + f(2 * (f(2 * (z[0])))) % f(2 * (0)) + t(1);
f(2 * (2+(3,4)-0,1)) | f(2 * (~ 5)) & f(2 * (0,1))^m(0,1);
```

Предопределенные макросы

```
#ifdef __cplusplus
// Действия, специфические для C++
#endif
```

```
  printf(" Дата компиляции — %s \n", __DATE__);
```

```
printf("Ошибка в файле %s\nВремя комп: %s\n",
__FILE__, __TIME__);
```

```
__LINE__
```

Поименованные области

`namespace [ имя_области ]{ /* Объявления */}`

```
namespace demo{
   int i = 1;
   int k = 0;
   void func1(int);
   void func2(int) { ... }
}
namespace demo{     // Расширение
   // int i = 2; Неверно — двойное определение
   void func1(double);// Перегрузка
   void func2(int);//Верно (повторное объявление)
}
```

определение

```
void demo::func1(int) { ... }
```

Обращение к элементам поименованной области

```
demo::i = 100; demo::func2(10);
using demo::i;
using namespace demo;
```

```
namespace DAM =
Department_of_Applied_Mathematics;
```

Пространства имен стандартной библиотеки

```
// stdio.h
namespace std{
int feof(FILE *f);
...
}
using namespace
std;
```

```
// cstdio
namespace std{
int feof(FILE *f);
...
}
```