Программирование с использованием функций на языке C++

Основные понятия

Функция представляет собой именованное объединение группы операторов, которое может быть вызвано из других частей программы.

Аргументом называют единицу данных (например, переменную типа int), передаваемую программой в функцию. Аргументы позволяют функции оперировать различными значениями или выполнять различные действия в зависимости от переданных ей значений.

Для использования всех функций, кроме main обязательно наличие 3-х компонентов функции: объявление, определение, вызов

В C++ при определении функции автоматически происходит ее объявление.

Определение функции в коде программы (описание)

```
//Заголовок функции
тип имя (список формальных параметров) //аргументы функции
//начало определения функции
{
Тело функции – набор инструкций
}//конец определения функции
```

Примеры различных заголовков функций

- типы аргументов функции могут различаться,
- возвращаемое значение у функции может отсутствовать (тип void),
- список аргументов функции может быть пустым,
- некоторые компиляторы позволяют явно не указывать типы в заголовке функции для типа по умолчанию (int).

```
int func1(float x, char c)
void func2(double d)
float func3()
//аналогично float func3(void)

func4(float a, float b)
//int func4(float a, float b)
char func5(a, float b)
//char func5(int a, float b)
func6(a, b, c)
//int func6(int a, int b, int c)
```

Возврат функцией значения

- при возвращаемом типе функции не void, в теле функции обязательно должен содержаться хотя бы один оператор return, причем обязательно с операндом (указание возвращаемого значения),
- тип операнда return обязательно должно совпадать с типом возвращаемого значения функции,
- результат, полученный после работы функции с возвращаемым типом не void может быть использован в другом выражении, если он по типу допустим для выражения,
- функция void может быть вызвана только отдельной инструкцией, результат ее работы не может быть использован в другом выражении,
- оператор return без операнда может быть использован в функции void для досрочного завершения работы функции.

Спецификация функции max

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

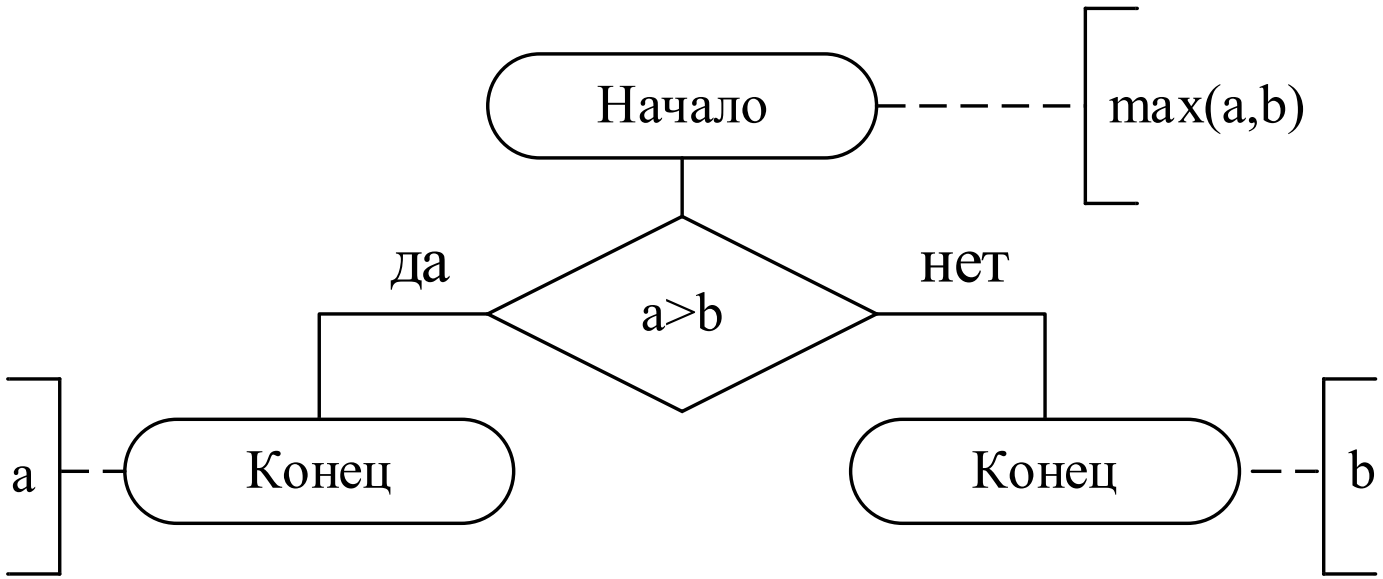
Функция max находит максимальное из двух целых чисел.

Входные аргументы:

— a, b – целые числа;

Возвращаемое значение:

— a или b – целое число.



Пример определения функции нахождения максимума из двух целых чисел

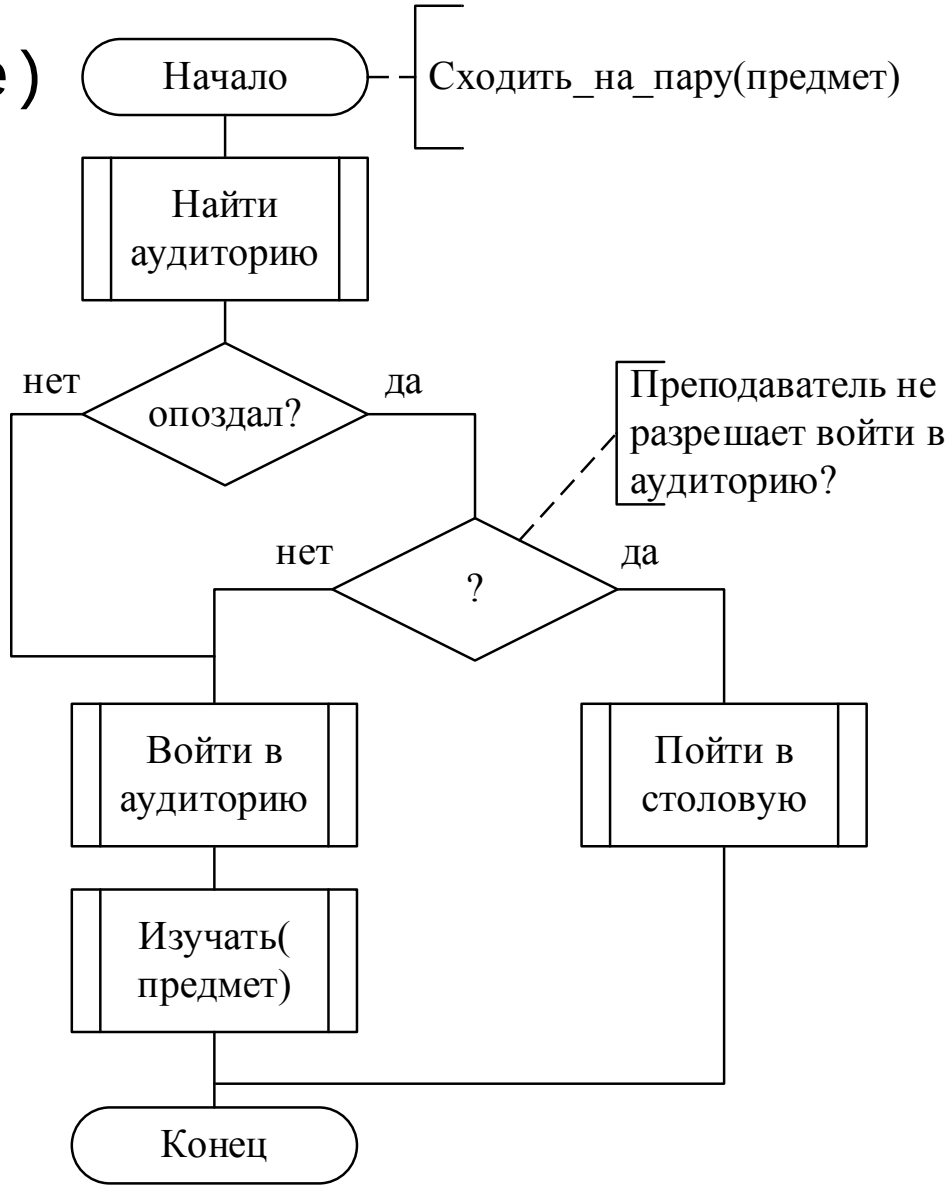
```
int max(int a, int b) {
if (a > b)
   return a;
else
   return b;
}
```

Пример группировки однотипных действий

Прийти в университет
Узнать сколько пар и какие // например, 2 пары: Мат. анализ и Алгебра
Найти аудиторию
Если опоздал
    Если преподаватель не разрешает войти
        Пойти в столовую
Иначе Зайти в аудиторию
Изучать «Математический анализ»
Перемена
Найти аудиторию
Если опоздал
    Если преподаватель не разрешает войти
        Пойти в столовую
Иначе Зайти в аудиторию
Изучать «Алгебра и теория чисел»
Перемена

Пример определения функции "Сходить на пару"

```
void goToClass(std::string className)
{
findClassroom();
if (startTimeIsOver)
  if (dntPerformToEnter){
        goToEatery();
        }
enterClassroom();
teach(className);
}
```

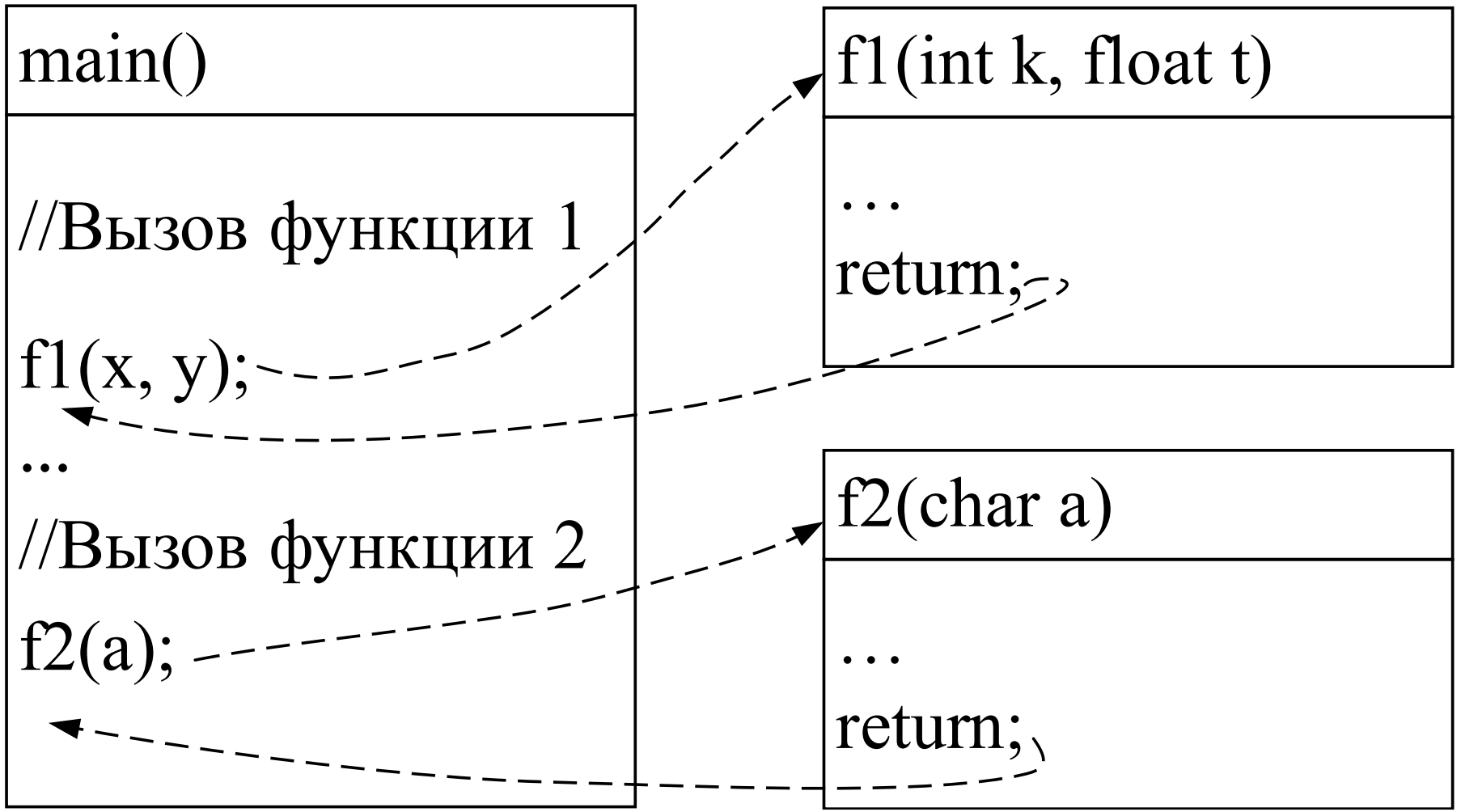


Вызов функции

- при указании значений аргументов важно соблюдать порядок их следования в соответствии с типами

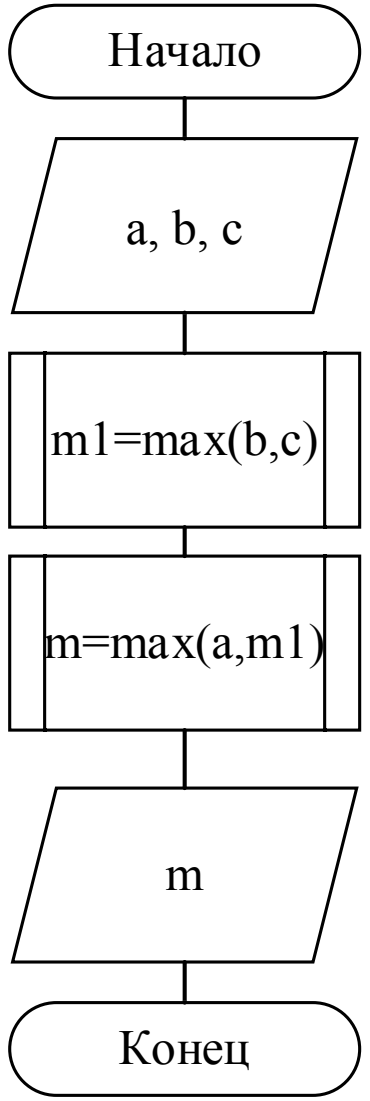
```
int main()//или другая функция
{
//обращение к функции
имя (список фактических параметров);
//действия, выполняемые после вызова функции
}
```

Механизм вызова функции и возврата функцией значения



Пример вызова функции max

```
int main(){
int a, b, c;
cin >> a >> b >> c;
cout << max(a, max(b, c));
}
```



Входные данные алгоритма решения задачи:

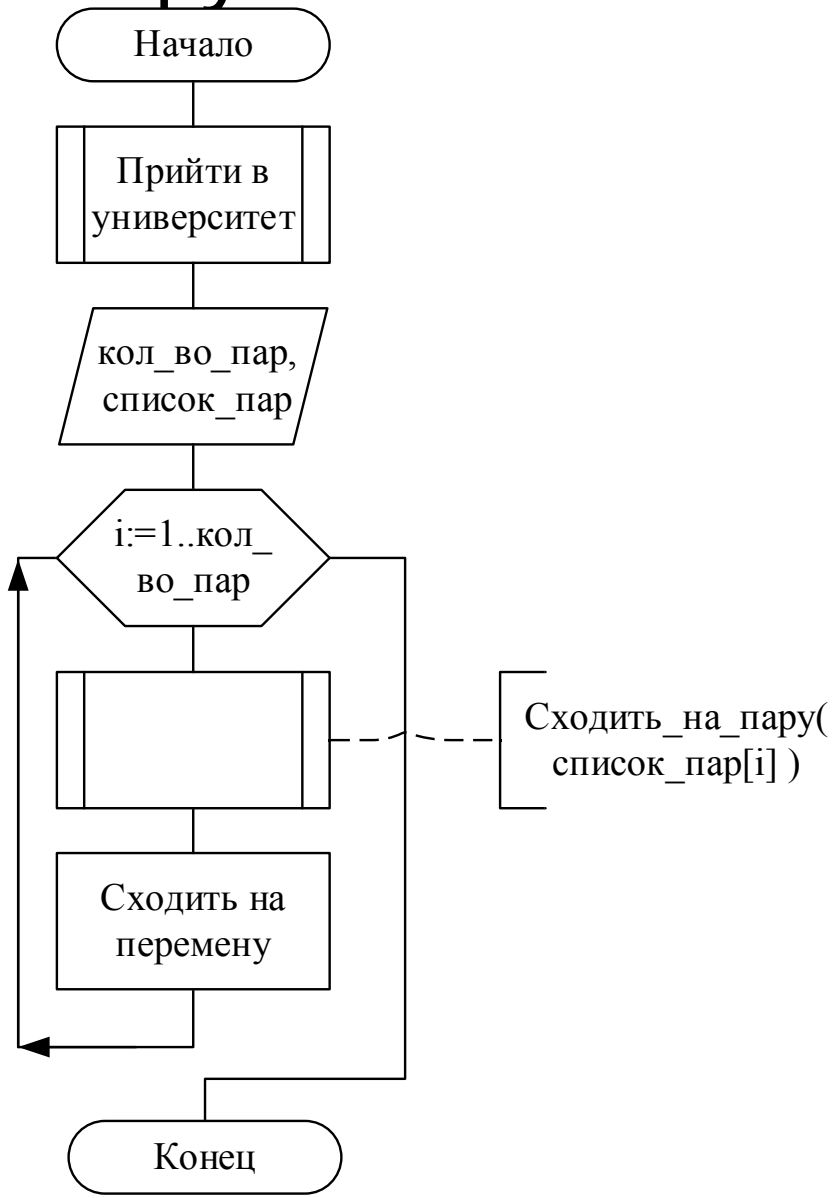
— a, b, c – целые числа;

Выходные данные алгоритма решения задачи:

— m – целое число.

Пример вызова функции "Сходить на пару"

```
int main(){
getToUniver();
unsigned classCount;
cin >> classCount; // например, 2
string *listOfLessons = new string[classCount];
for (unsigned i = 0; i < classCount; i++)
   cin >> listOfLessons[i];
//например, "Математический анализ",
// "Алгебра и теория чисел"
for (unsigned i = 0; i < classCount; i++){
      goToClass( listOfLessons[i] );
      goToRest();
      }
}
```



Формальные и фактические параметры

```
int myMax(int a, int b) {// a и b - формальные
return (a > b) ? a : b;
}

int myAbs(int z) { // z - формальный
return (z < 0) ? –z : z;
}

int main(){
int x = 3, y = -5;
int c = myMax(myAbs(y), x); // 1) фактический y для myAbs
// 2) фактические: результат вызова myAbs(y) и x
}
```

Еще одна функция с возвращаемым типом НЕ void

```
float sum(float s1, float s2){
float s = 0;
for (s1; s1 < s2 + 0.1; s1 += 0.1)
   s += s1;
return s; //обязательно присутствует return с операндом
}

int main(){
float a, b;
cin >> a >> b;
float c = sum(a, b);
cout << c;
//допустимо использовать результат работы функции в выражении
//cout << sum(a, b);
}
```

Несколько возвратов в теле функции с типом НЕ void

```
std::string roots(float a, float b, float c){
float d = b * b – 4 * a * c;
   if (d > 0)
      return "two roots";
   else
      if (d < 0)
         return "complex roots";
return "one multiplicity root";
}
```

Возвращаемый тип void

```
void say(std::string s, int val){  //функция ничего не
//val = 3
std::cout << s;                    //возвращает, просто
std::cout << ++val;                     // что-то делает
//val = 4
}

int main(){
int min = 3; //think
say("Хочу сдать экзамен на ", min);
//min = 3, значение локальной переменной не меняется
}
```

Применение оператора return в функции с типом void

```
void life(std::string s, int val){
val--;
std::cout << s << ' ';
if (val < 5 && val > 2){
  std::cout << val;
  return; //без операнда, досрочное завершение функции
  }
std::cout << "то, что заслуживаю";
}

int main(){
life("Хочу на экзамене получить ", 4);
}
```

Хороший стиль программирования

1. Функция main должна быть определена первой в файле с исходным кодом.
2. Все остальные функции определяются после main, но должны быть объявлены перед ней.

Порядок чтения определений функций в коде программы

```
int main(){
float a, b;
cin >> a >> b;
//float c = sum(a, b); ошибка – sum неизвестное имя
}

//определение функции
float sum(float s1, float s2){
float s = 0;
for (s1; s1 < s2 + 0.1; s1 += 0.1)
   s += s1;
return s;
}
```

Прототип функции (объявление функции)

```
//прототип функции
float sum(float x, float y); //равносильно: float sum(float,
  float);


int main(){
float a, b;
cin >> a >> b;
float c = sum(a, b); //ошибки нет
}


//определение функции
float sum(float s1, float s2){
float s = 0;
for (s1; s1 < s2 + 0.1; s1 += 0.1)
  s += s1;
return s;
}
```

Локальный прототип функции

```
int main(){
void f();
void g();
cout << "from main" << endl;
f();
g();
}

void f(){
cout << "f function" << endl;
//g(); ошибка, т.к. "void g();" – локальный прототип другой области
}

void g(){
f();//уже может быть вызвана, т.к. определена выше
cout << "g function" << endl;
}
```

Дано натуральное число, вывести его максимальную цифру

```
unsigned maxDigit(unsigned, unsigned);

int main(){
unsigned number;
cin >> number;
unsigned max = number % 10;
do{
   number /= 10;
   max = maxDigit(max, number % 10);
   }
while (number / 10 > 0);
cout << max;
}

unsigned maxDigit(unsigned fst, unsigned snd){
return (fst > snd) ? fst : snd;
}
```

Найти все простые числа, не превышающие n

```
bool isSimple(unsigned);

int main(){
unsigned n;
cin >> n;
for (unsigned i = 1; i <= n; i++)
   if(isSimple(i))
      cout << i << endl;
}

bool isSimple(unsigned num){
for (unsigned i = 2; i <= sqrt(num); i++)
   if(!(num % i))
      return false;
return true;
}
```

Найти n первых простых чисел

```
bool isSimple(unsigned);

int main(){
unsigned n, number = 1;
cin >> n;
while (n){
   if(isSimple(number)){
      cout << number << ", ";
      n--;
      }
   number++;
   }
}
```

Найти сумму делителей каждого из натуральных чисел, не превышающего n

```
unsigned dividersSum(unsigned);

int main(){
unsigned n;
cin >> n;
for (unsigned i = 1; i <= n; i++)
   cout << dividersSum(i) << endl;
}

unsigned dividersSum(unsigned num){
unsigned sum = 0;
for (unsigned i = 1; i <= num; i++)
   if(!(num % i))
      sum += i;
return sum;
}
```

Значения параметров функции по умолчанию

тип имяФункции(типП1 имяП1 = значениеТипаП1, типП2 имяП2 = значениеТипаП2);
//аналогично тип имяФункции(типП1 = значениеТипаП1, типП2 = значениеТипаП2);

Пример прототипа функции:

```
float f(float, float = 1.3, int = 5, bool = true);

int main(){
float k;
int i;
//возможные примеры вызовов функции f:
f(k);
f(3.5, k);
f(k, k + 0.1, 3);
f(k, 4.0, i, false);
}
float f(float a, float b, int c, bool d){/*тело функции*/}
```

Значением параметра по умолчанию может быть переменная или результат вызова функции

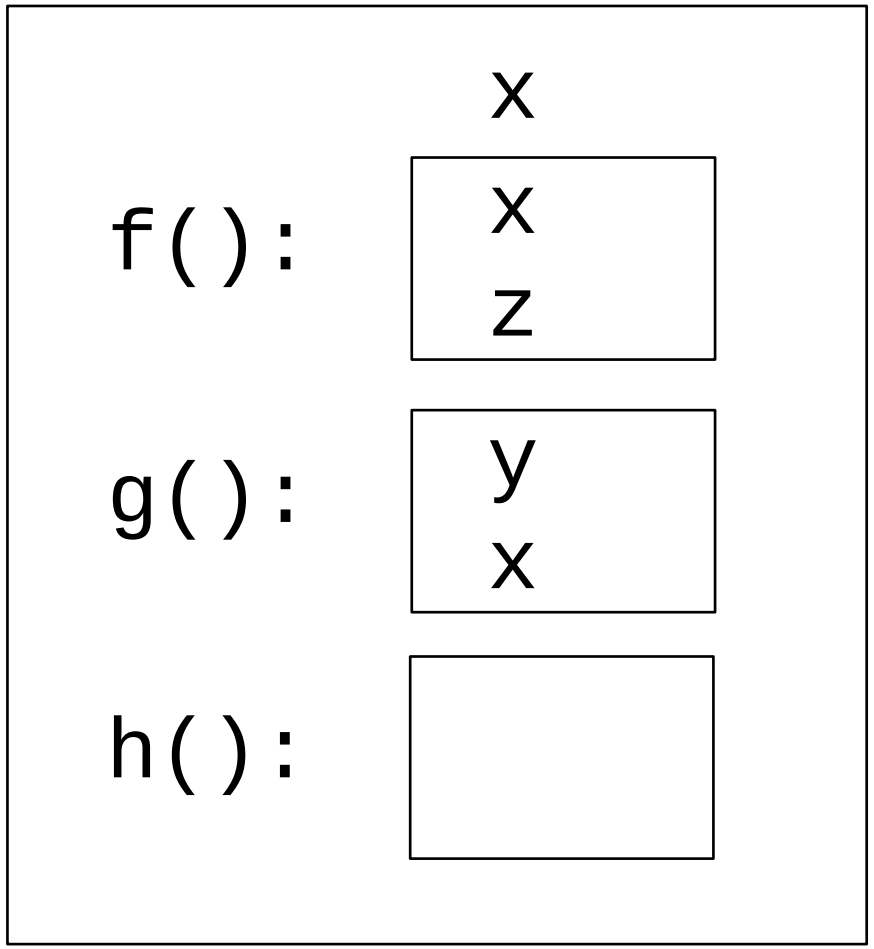
```
float n;
const int x = 10;
float r51(){return 5.1;}
…
//переменная
float fun1(int = x, float = n);
//результат вызова функции
float fun2(float = 0, float = r51());

//константы также можно инициализировать
//во время выполнения программы
const float y = r51();
```

Локальные и глобальные переменные

- локальные переменные существуют только в пределах блока, в котором они описаны,
- глобальные переменные объявляются вне какого-либо блока, существуют на протяжении работы программы, можно использовать в любых локальных блоках,
- область действия переменной – область кода программы, в которой можно обратиться к этой переменной каким-либо образом,
- область видимости переменной – область кода программы, в которой можно обратиться к переменной по имени (видимость может быть перекрыта одноименной локальной переменной),
- время жизни переменной – время существования переменной в памяти; для локальных – с момента начала выполнения инструкций блока, в котором она описана, до его завершения, для глобальных – от начала до конца работы программы.

Область видимости переменной

```
…
int x;
…
float f(){
char x;
int z;
}
…
int g(){
bool y;
double x;
}
…
void h(){
cout << x;
}
```



Разрешение (уточнение) видимости – оператор ::

```
//программа предназначена исключительно для демонстрации,
//использование в функции глобальной переменной, а также
//определение одноименной локальной переменной – это
//очень плохой стиль программирования
int x; //глобальная x типа int
…
float f(){
char x; //локальная x типа char
int z;
}
…
int g(){
bool y;
double x;//локальная x типа double
::x; //глобальная x типа int
}
```

Пример применения операции разрешения видимости (::)

```
int x = 1;
…
float f(){
int y;
char x = 2;
   {
   short x = 3;
   cout << x;   //3
   cout << ::x; //1
   }
cout << x;      //2
cout << ::x;    //1
}
```

Использование переменных глобальной области видимости

```
extern int x; // объявление переменной x
void fz();    // объявление функции fz
int z;        // определение переменной z, глобальные инициализируются 0

int main(){
cout << x << " " << z << endl; //1 0, доступна определенная ниже x = 1
fz();
cout << x << " " << z << endl; //1 2, z меняет значение после вызова fz
}

int x = 1;   // определение и инициализация x

/*
нельзя просто написать:
x = 1
в глобальной области,
т.к. здесь допустимы только объявления или определения
*/

void fz(){ z = 2; }
```