

Подпрограммы

Определение функции, фактические и формальные параметры функции


- **Функция** - это совокупность объявлений и операторов, предназначенная для решения определенной задачи. 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*. Объявление функции должно находиться в тексте раньше ее вызова для того, чтобы компилятор мог осуществить проверку правильности вызова.
- *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])[throw (исключения)]{ тело функции }

Определение функции, фактические и формальные параметры 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 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`.
- 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 (в объявлении имени можно опускать).

- Список формальных параметров - это последовательность объявлений формальных параметров, разделенная запятыми. Формальные параметры - это переменные, используемые внутри тела функции и получающие значение при вызове функции путем копирования в них значений соответствующих фактических параметров.
- Если функция не использует параметров, то наличие круглых скобок обязательно, а вместо списка параметров рекомендуется указать слово `void`.
- *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.
- Параметры функции передаются по значению и могут рассматриваться как локальные переменные, для которых выделяется память при вызове функции и производится инициализация значениями фактических параметров. При выходе из функции значения этих переменных теряются. Поскольку передача параметров происходит по значению, в теле функции нельзя изменить значения этих переменных в вызывающей функции, являющихся фактическими параметрами. Однако, если в качестве параметра передать указатель на некоторую переменную, то используя операцию разадресации можно изменить значение этой переменной.
- Тип возвращаемого значения и типы параметров совместно определяют *тип 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.

Найти сумму 5-ти факториалов

- `#include <iostream>`
- `using namespace std;`
- `int fact(int);` *//объявление ф-ции*
- `int main()`
- `{`
- `int s = 0, n, x;`
- `for (int i = 1; i <= 5; i++)`
- `{cout << "Input chislo: ";`
- `cin >> n;`
- `x = fact(n);`  *Обращение к функции*
- `cout << "factorial " << n << "=" << x << endl;`
- `s += x;`
- `}`

Найти число сочетаний из n по k

$$C_n^k = \frac{n!}{k!(n-k)!}$$

```
1.  #include <iostream>
2.  using namespace std;
3.  int fact (int);
4.  int main ()
5.  {
6.      int n,k;
7.      cout<<"vvod n i k ";
8.      cin>>n>>k;
9.      cout<<fact(n)/(fact(k)*fact(n-k))<<endl;
10.     system("pause");
11.     return 0;
```

Статические одномерные массивы

- **Массив** – это упорядоченный набор элементов одного типа.
- Для того чтобы объявить массив и проинициализировать его данными элементами, мы должны написать следующую инструкцию C++:

```
int fibon[9] = { 0, 1, 1, 2, 3, 5, 8, 13, 21 };
```

- Здесь **fibon** – это имя массива.
- Элементы массива имеют тип `int`, *размер* (длина) массива равна 9. Значение первого элемента – 0, последнего – 21.
- Для работы с массивом мы *индексируем* (нумеруем) его элементы, а доступ к ним осуществляется с помощью операции *взятия индекса*.

Статические одномерные массивы

- Для обращения к первому элементу массива естественно написать:
- `int first_elem = fibon[1];`
- Однако это не совсем правильно: в С/С++ индексация массивов начинается с 0, поэтому элемент с индексом 1 на самом деле является *вторым* элементом массива, а индекс первого равен 0.
- Чтобы обратиться к последнему элементу массива, мы должны вычесть единицу из размера массива:
- `fibon[0];` // первый элемент
- `fibon[1];` // второй элемент
- ...
- `fibon[8];` // последний элемент
- `fibon[9];` // ... ошибка
- Девять элементов массива **fibon** имеют индексы от 0 до 8.

Инициализируем массив из десяти элементов числами от 0 до 9 и затем напечатать их в обратном порядке:

1. `#include <iostream>`
2. `using namespace std;`
3. `void main()`
4. `{int ia[10];`
5. `int index;`
6. `cout << "isxodnyi massiv"<< endl;`
7. `for (index=0; index<10; ++index) // ia[0] = 0, ia[1] = 1 и т.д.`
8. `{ia[index] = index;`
9. `cout << ia[index] << " ";`

Генерация (псевдо)случайных чисел

- Функция стандартной библиотеки:
- **`int rand (void);`**
- Она генерирует псевдослучайное целое число на интервале значений от 0 до `RAND_MAX`. Последнее является константой, которая варьируется в зависимости от реализации языка, но в большинстве случаев составляет 32767.
- Генерация чисел от 0 до 9:
- **`rand() % 10`**
- Если нам нужны числа от 1 (а не от 0) до 9, то можно прибавить 1:
- **`rand() % 9 + 1`**
- Идея такая: генерируем случайное число от 0 до 8, и после прибавления 1 оно превращается в случайное число от 1 до 9.

- Функция `rand()` генерирует псевдослучайные числа, т.е. числа, которые кажутся случайными, но на самом деле являются последовательностью значений, вычисленных по хитрому алгоритму, в качестве параметра принимающему так называемое зерно (`seed`). Т.е. сгенерированные функцией `rand()` числа будут зависеть от значения, которое имеет зерно в момент ее вызова. А зерно всегда устанавливается компилятором в значение 1. Иными словами, последовательность чисел будет хоть и псевдослучайной, но всегда одинаковой. Исправить ситуацию помогает функция `srand()`.
- **`void srand (unsigned int seed);`**
- Она устанавливает зерно равным значению параметра, с которым была вызвана. И последовательность чисел тоже будет другая.

- Чтобы сделать зерно всегда разным, используем функцию `time()`.
- **`time_t time (time_t* timer);`**
- Она тоже досталась в наследство от языка Си и, будучи вызвана с нулевым указателем в качестве параметра, возвращает количество секунд, прошедших с 1 января 1970 года. Теперь значение этой функции мы можем передать в функцию `srand()` и будет случайное зерно. И числа будут неповторяющиеся.
- Для использования функций **`rand()`** и **`srand()`** нужно подключить заголовочный файл **`<cstdlib>`**, а для использования `time()` – файл **`<ctime>`**.

Найти минимальный элемент одномерного массива и его порядковый номер

1) `#include <iostream>`

2) `#include<cstdlib>`

3) `#include<ctime>`

4) `using namespace std;`

5) `int main()`

6) `{`

7) `const int n= 10;`

8) `int a[n];`

9) `int i, imin;`

10) `srand(time(NULL));`

11) `for (i = 0; i < n; i++)`

12) `{ a[i]=rand()% 10;`

Подсчитать сумму элементов массива

1. ...
2. `const int n = 10;`
3. `// размерность массива задана целой положительной`
`CONSTАНТОЙ`
4. `int marks[n] = {3, 4, 5, 4, 4};` `//массив инициализируется`
`при объявлении`
5. `int sum = 0;`
6. `for (int i = 0; i<n; i++)`
7. `sum += marks[i];`
8. `cout << "Сумма элементов: " << sum;`

Операция определения размера **sizeof**

- С помощью операции **sizeof** можно определить размер памяти (в байтах), которая необходима для хранения элемента с соответствующим идентификатором или определенного типа. Операция **sizeof** имеет следующий формат:

sizeof (выражение)

- В качестве выражения может быть использован любой идентификатор, либо имя типа, заключенное в скобки. Отметим, что не может быть использовано имя типа **void**, а идентификатор не может относиться к полю битов или быть именем функции.
- Если в качестве выражения указано имя массива, то результатом является размер всего массива (т.е. произведение числа элементов на длину типа), а не размер указателя, соответствующего идентификатору массива.

Одномерной массив сместить на k позиций вправо

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     int x[]={0,8,6,3,5,0,1,3,-7};
6.     int n=sizeof(x)/sizeof(x[0]); //определение размера массива
7.     int i, k, buf, r;
8.     cout<<"vvedite k=";
9.     cin>>k;
10.    cout << "isx"<<endl;
11.    for (i = 0; i < n; i++)
12.        cout << x[i] << ' ';
```


В одномерном массиве сдвинуть элементы на число k, вводимое с клавиатуры, влево

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     int x[]={1,2,3,5,4,0,1,3,-7};
6.     int n=sizeof(x)/sizeof(x[0]);
7.     int i, k, buf, r;
8.     cout<<"vvedite k=";
9.     cin>>k;
10.    cout << "isx"<<endl;
11.    for (i = 0; i < n; i++)
```

Многомерные массивы

- задаются указанием каждого измерения в квадратных скобках, например, оператор `int matr [6][8];`
- задает описание двумерного массива из 6 строк и 8 столбцов. В памяти такой массив располагается в последовательных ячейках *построчно*. Многомерные массивы размещаются так, что при переходе к следующему элементу быстрее всего изменяется последний индекс.
- Для *доступа* к элементу многомерного массива указываются все его индексы, например, `matr[i][j]`.
- *Инициализация* многомерного массива:
- `int mass2 [][]={ {1, 1}, {0, 2}, {1, 0} };`
- `int mass2 [3][2]={1, 1, 0, 2, 1, 0};`

Правила

- Размерность нединамического массива может быть только константой или константным выражением. Рекомендуется задавать размерность с помощью именованной константы.
- Элементы массивов нумеруются с нуля, поэтому максимальный номер элемента всегда на единицу меньше размерности.
- В двумерном массиве первый индекс всегда представляет собой номер строки, второй — номер столбца. Каждый индекс может изменяться от 0 до значения соответствующей размерности, уменьшенной на единицу.
- Массив хранится по строкам в непрерывной области памяти.
- Автоматический контроль выхода индекса за границы массива не производится, поэтому программист должен следить за этим самостоятельно.
- При описании массива можно в фигурных скобках задать начальные значения его элементов.

В двумерном массиве посчитать кол-во положительных элементов в строках

```
1. #include<iostream>
2. #include <iomanip> //определены некоторые манипуляторы потокового ввода/вывода
3. #include<cstdlib> // Для функций rand() и srand()
4. #include<ctime> //использование функции time() для усиления случайности "зерна"
5. using namespace std;
6. int main ()
7. {
8.     const int nstr = 4, nstb = 5;
9.     int b[nstr][nstb],a[nstr];
10.    int i, j,kol;
11.    srand(time(NULL));
12.    for (i = 0; i < nstr; i++)
13.        for (j = 0; j < nstb; j++)
14.        {
15.            b[i][j]=rand()%10-5; }
```

Сумма элементов главной и побочной диагонали a(4,4)

```
1. #include<iostream>
2. #include <iomanip>
3. #include<cstdlib>
4. #include<ctime>
5. using namespace std;
6. int main ()
7. {
8.     const int n= 4;
9.     int a[n][n];
10.    int i, j, s1,s2;
11.    s1=s2=0;
```