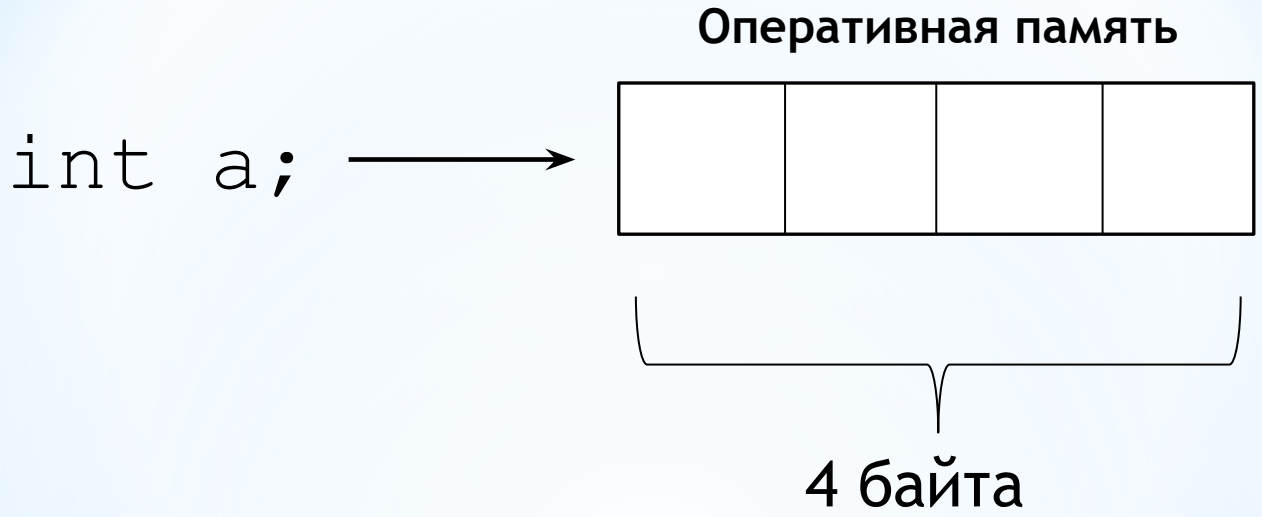
Кафедра бизнес-информатики ИЭМ ВлГУ

Программирование
Лекция № 4

*Массивы. Функции

Владимир, 2012

Целое число



Области памяти соответствует имя переменной (в данном случае a)

Как представить вектор $a = (5;3;4)$?

```
int a1 = 5, a2 = 3, a3 = 4;
```

Почему эта форма записи плоха?

* Вектор может иметь большую размерность
* Необходимо иметь механизм для выполнения однообразных действий с элементами вектора

Массив

* **Массив** – это конечная именованная последовательность однотипных величин.
* **Размерность массива** – количество его элементов.

Синтаксис:

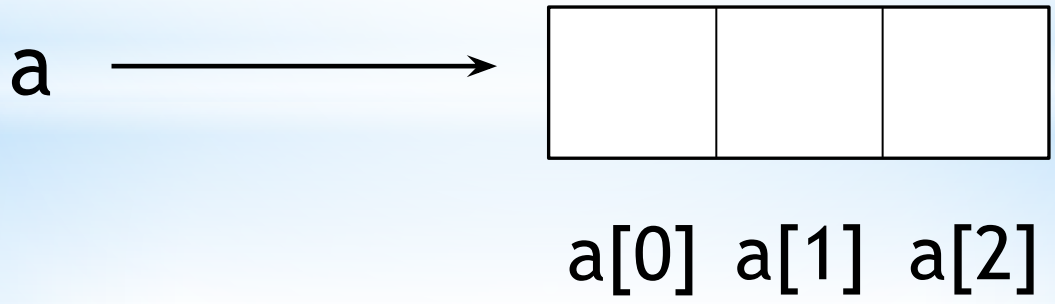
тип имя[размерность];

```
int a[3];
float b[5];
char c[15];
```

Индексация

* **Индекс** элемента массива – это его номер
* **Индексация** элементов в C++ идет **с нуля**

```
int a[3]; // массив целых
// элементы: a[0], a[1], a[2]
```



Инициализация

* Значения элементов массива пишутся в **фигурных скобках**

```
int a[3]={5,3,4};
// a[0]=5, a[1]=3, a[2]=4
```

* Попытка вывода неинициализированных элементов вызовет **ошибку времени исполнения**

```
int a[3];
cout << a[2]; // ошибка!
```

Инициализация

* В списке инициализации можно задать не все элементы, а несколько первых, в остальные будут записаны нули:

```
int b[5]={1,2,3};
// b[0]=1, b[1]=2, b[2]=3, b[3]=0, b[4]=0
```

* Заполнить массив нулями можно так:

```
int c[10]={};
```

Инициализация

* Допустимо не задавать размерность массива, в этом случае память будет выделена по числу элементов в списке инициализации:

```
double d[]={1.5,2.5,3.5};
// d[0]=1.5, d[1]=2.5, d[2]=3.5
```

* Выделение памяти происходит *на этапе компиляции*, поэтому размерность задается **константой** или **константным выражением**

```
const int M=5, N = 3;
int a[M*N]; // массив из 15 целых
```

Доступ к элементам массива

* Для доступа к элементам массива используются квадратные скобки:

```
// d[0]=1.5, d[1]=2.5, d[2]=0
double d[3]={1.5,2.5};
d[2]=3.5; // d[2]=3.5
double sum = 0; // переменная для хранения суммы
for (int i = 0; i < 3; i++) sum += d[i];
cout << "Сумма: « << sum << endl;
// Сумма: 7,5
```

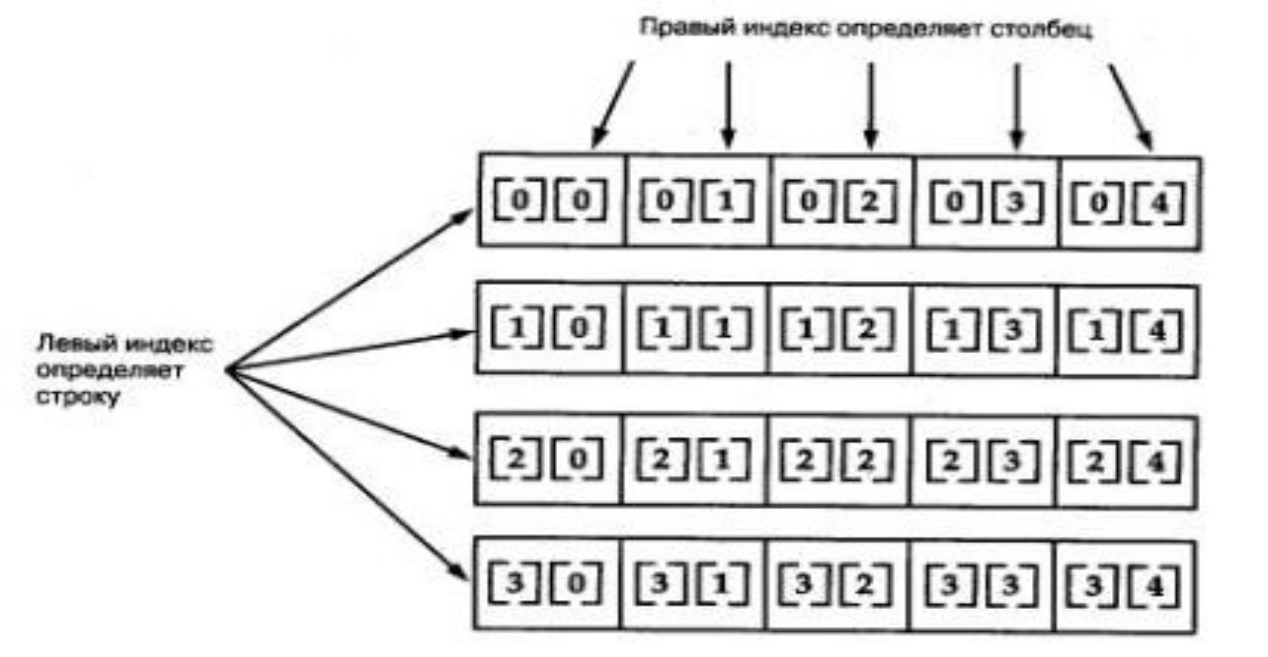
Многомерные массивы

Как представить матрицу? В виде массива массивов

Синтаксис объявления **N-мерного** массива:

тип имя[размерность1][размерность2]…[размерностьN];

```
int a[4][5]; // массив из 4 элементов, каждый
// из кот. является массивом из 5 эл-в
```



Двумерный массив: инициализация

1 способ: Каждая строка заключается в свои фигурные скобки (можно не указывать число строк)

```
int mass2 [][2] = { {1, 1}, {0, 2}, {1, 0} };
```

2 способ: Все элементы указываются в одних фигурных скобках, массив инициализируется построчно:

```
int mass2 [3][2] = {1, 1, 0, 2, 1, 0};
// mass2[0][0]=1, mass2[0][1]=1
// mass2[1][0]=0, mass2[1][1]=2
// mass2[2][0]=1, mass2[2][1]=2
```

Пример работы с двумерным массивом

Программа ищет номер строки двумерного массива, в которой больше всего нулей:

```
const int M = 4, N = 5; // размерности массива
int b[M][N]; // описание массива
int i, j; // счетчики циклов
for (i = 0; i<M; i++) // ввод массива
   for (j = 0; j<N; j++) scanf("%d",&b[i][j]);
 // номер искомой строки,
// максимальное количество нулей
int istr = -1, max_count = 0;
```

Пример работы с двумерным массивом

```
for (i = 0; i<M; i++)
{ // просмотр массива по строкам
   int count = 0;
   for (j = 0; j<N; j++)
      if ( b[i][j] == 0) count++;
   if ( count > max_count)
   {
      istr = i ; max_count =  count;
   }// if
}// for
```

Пример работы с двумерным массивом

```
printf("Исходный массив:\n");
for (i = 0; i<M; i++)
{
   for (j = 0; j<N; j++) printf("%d ", b[i][j]);
   printf ("\n" ) ;
}
if (istr == -1)
   printf("Нулевых элементов нет");
else printf("Номер строки: %d", istr + 1 );
```

Вложенные циклы

Пример (подсчет суммы элементов массива):

```
int a[][N]={1,2,3,4};
int sum[N]={};
for (int i = 0; i < N; i++)
{
   for (int j = 0; j < N; j++)
sum[i]+=a[i][j]; // for2
   cout << "Сумма элементов " << i+1 << "-й
строки равна " << sum[i] << endl;
} // for1
```

Оператор break

Оператор **break** прекращает выполнение ближайшего внешнего для него оператора цикла.

```
const int M=3, N=2;
int a[M][N] = {{0,2}, {3,0}, {5,6}};
int sum = 0;
for (int i = 0; i < M; i++)
{
   for (int j = 0; j < N; j++)
   {
      if (a[i][j]==0) break;
      sum += a[i][j];
   }
}
cout << sum << endl;
```

Результат: 14

Оператор continue

Оператор **continue** передает управление на ближайшую внешнюю проверку условия продолжения цикла.

```
const int M=3, N=2;
int a[M][N] = {{0,2}, {3,0}, {5,6}};
int sum = 0;
for (int i = 0; i < M; i++)
{
   for (int j = 0; j < N; j++)
   {
      if (a[i][j]==0) continue;
      sum += a[i][j];
   }
}
cout << sum << endl;
```

Результат: 16

Пример

$$Sum = \sum_{i=1}^{M} \prod_{j=1}^{N} a_{ij}$$

```
const int M=3, N=2;
int a[M][N] = {{1,2}, {3,4}, {5,6}};
int sum = 0;
for (int i = 0; i < M; i++)
{
   int p = 1, j;
   for (j = 0; j < N; j++)
   {
      if (a[i][j]==0) break;
      p *= a[i][j];
   }
   if (j < N-1) continue;
   sum += p;
}
cout << sum << endl;
```

Результат: 44

*Функции

Функции используются
для наведения порядка в хаосе алгоритмов.
Бьорн Страуструп

Функция

Функция — это именованная последовательность описаний и операторов, выполняющая какое-либо законченное действие.

Синоним: **метод**

*f(x, y)= x * y – математическая функция*

Принимает значения параметров (x и y)

Возвращает результат (произведение)

Функция

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

Синонимы: **прототип, заголовок, сигнатура**

Задается имя функции, тип возвращаемого значения, типы параметров (можно без имен):

тип имя ([тип_пар1, тип_пар2,…, тип_парN]);

```
int f1 (int, int);
void f2 (double);
void f3();
```

Определение функции

Задается имя функции, тип возвращаемого значения, типы параметров (с именами), тело функции – последовательность операторов и описаний в фигурных скобках:

тип имя ([тип_пар1, тип_пар2,…, тип_парN])
{ тело функции; }

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

Тип возвращаемого значения

Может быть любым, кроме массива и функции

Если функция не возвращает значения, указывается тип **void**

В теле функции возвращаем значение с помощью оператора **return** (если значение не возвращается, return писать не надо)

Список параметров

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 имена можно опускать).

Вызов функции

Для вызова функции в простейшем случае нужно указать ее имя, за которым в круглых скобках через запятую перечисляются имена передаваемых аргументов:

```
sum(3,5);
```

Если функция возвращает значение, она может входить в состав выражений:

```
int s = sum(3,5);
```

Параметры функции

Параметры, перечисленные в заголовке, называются **формальными параметрами:**

```
int sum (int a, int b);
// a и b – формальные параметры
```

Параметры, передаваемые при вызове функции, называются **фактическими параметрами:**

```
int s = sum(3,5);
// 3 и 5 – фактические параметры
```

Пример (вычисление суммы)

```
int sum(int a, int b); // объявление функции

int _tmain(int argc, _TCHAR* argv[])
{
   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);
}
```

Преимущества использования функций

Возможность избежать дублирования кода

Упрощение структуры программ

Повышение степени абстракции программ

Возможность создавать библиотеки часто используемых функций

Упрощение отладки и сопровождения

Источники

1. Керниган Б., Ритчи Д. Язык программирования Си. – СПб.: «Невский диалект», 2001. – 352 с.: ил.
2. Павловская Т. А. C/C++. Программирование на языке высокого уровня. – СПб.: Питер, 2003. – 461 с.: ил.
3. Подбельский В. В., Фомин С. С. Программирование на языке Си: Учеб. пособие. – 2-е доп. изд. – М.: Финансы и статистика, 2004. – 600 с.