Массивы, строки и указатели.

В рамках модуля «Язык программирования C/C++»

Мухаметов Данил Илгизович

seemsclever@mail.ru

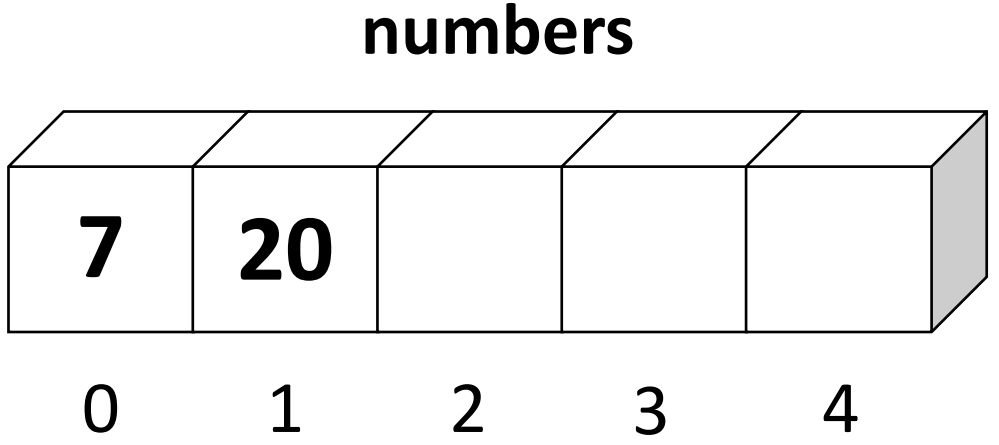
2025

Что такое массивы?

Массив – это набор элементов одного типа, размещённых в непрерывной области памяти. Все элементы массива имеют общее имя и нумеруются индексами, начиная с нуля.

Массивы используются, когда нужно хранить и обрабатывать несколько однотипных значений – например, результаты измерений

```
int numbers[5];   // массив из 5 целых чисел
numbers[0] = 7;   // первый элемент
numbers[1] = 20;  // второй элемент
```



Объявление и инициализация массива

Для создания массива необходимо указать тип элементов, имя массива и количество элементов в квадратных скобках. Размер массива должен быть известен на этапе компиляции.

Примеры объявления:

```
int numbers[5];    // массив из 5 целых чисел
double grades[10];    // массив из 10 вещественных чисел
char letters[3];     // массив из 3 символов
```

Примеры инициализации:

```
int a[5] = {1, 2, 3, 4, 5};      // полная инициализация
int b[5] = {1, 2};              // частичная (остальные элементы = 0)
int c[] = {10, 20, 30};         // размер определяется автоматически
```

Доступ к элементам в массиве

Каждый элемент массива имеет свой индекс – номер, который указывает его положение.

Доступ к элементу осуществляется по индексу в квадратных скобках.

Нумерация элементов начинается с нуля.

```
int numbers[5] = {10, 20, 30, 40, 50};


cout << numbers[0];   // 10 — первый элемент
cout << numbers[4];   // 50 — пятый элемент


numbers[4] = 100;     // изменение третьего элемента


cout << numbers[4];   // 100
```

Обработка массива

Часто требуется выполнить одно и то же действие для всех элементов массива: вывести их на экран, посчитать сумму, найти минимальное или максимальное значение. Для этого удобно использовать цикл for.

```
int numbers[5] = {10, 20, 30, 40, 50};


for (int i = 0; i < 5; i++) {     // i от 0 до 4
   cout << numbers[i] << " ";
}


// Результат работы программы: 10 20 30 40 50
```

Диапазонный цикл for

Диапазонный цикл for – это удобный способ пройти по всем элементам массива или строки.

Цикл автоматически проходит по всем элементам массива или строки – без индексов и счётчиков.

```
int numbers[5] = {10, 20, 30, 40, 50};
for (int x : numbers) {
    cout << x << " ";
} // будет выведено 10 20 30 40 50
```

```
for (int& x : numbers) {
    x += 1;
} // массив изменится на {11, 21, 31, 41, 51}
```

```
string text = "C++";
for (char c : text) {
    cout << c << " ";
} // будет выведено C + +
```

Пример – подсчёт суммы элементов массива

```
int numbers[5] = {4, 7, 2, 9, 3};
int sum = 0;

for (int i = 0; i < 5; i++) {
    sum += numbers[i];
}
```

```
cout << "Сумма элементов: " << sum;

// Результат работы программы:
// Сумма элементов: 25
```

Пример – подсчет средней температуры

```
int temps[7] = {15, 17, 14, 18, 20,
16, 19};

int sum = 0;

for (int i = 0; i < 7; i++) {
    sum += temps[i];
}

double avg = sum / 7.0;

cout << "Средняя температура: " <<
avg;

// Результат работы программы:
// Средняя температура: 17
```

Выход за границы массива

При обращении к элементу массива по индексу важно, чтобы индекс находился в допустимом диапазоне: от 0 до размер - 1.

```
int numbers[5] = {1, 2, 3, 4, 5};

cout << numbers[5];   // ошибка: элемента с индексом 5 нет

numbers[-1] = 10;     // ошибка: отрицательный индекс
```

Как посчитать размер массива?

В языке C++ обычный массив не имеет встроенного метода для определения своего размера. Для того, чтобы определить его размер, поделим размер всего массива на размер одного элемента.

```
int array[5] = {1, 2, 3, 4, 5};

int size = sizeof(array) / sizeof(array[0]);

cout << "Количество элементов: " << size;
```

Двумерные массивы

Двумерный массив – это структура, в которой данные расположены в виде таблицы, состоящей из строк и столбцов.

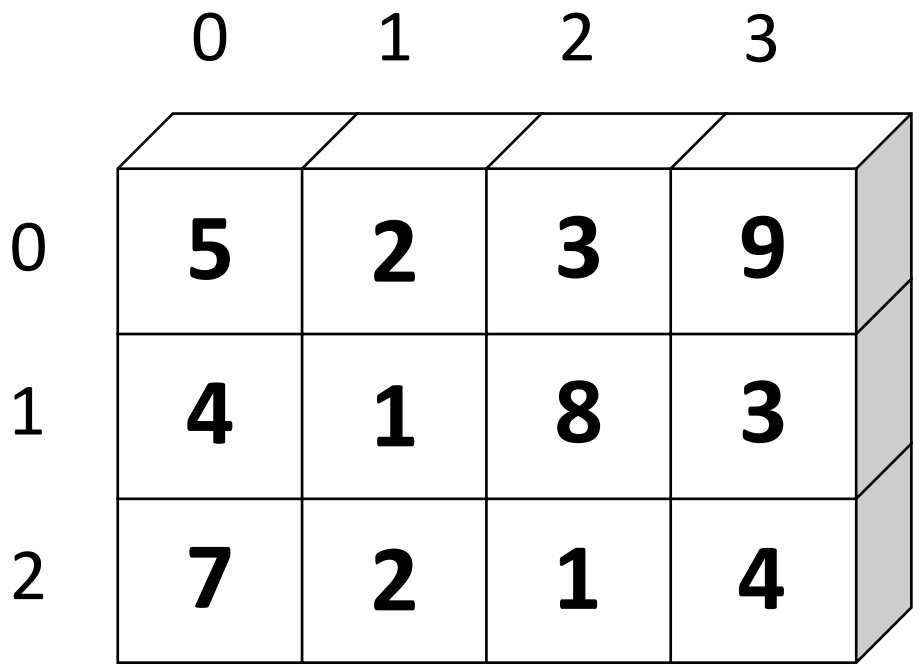
Каждый элемент такого массива имеет два индекса:

первый – номер строки,

второй – номер столбца.

```
int matrix[3][4] = {
    {5, 2, 3, 9},
    {4, 1, 8, 3},
    {7, 2, 1, 4},
};
```

matrix



Доступ к элементам в двумерном массиве

```
int matrix[3][4] = {
    {5, 2, 3, 9},
    {4, 1, 8, 3},
    {7, 2, 1, 4}
};
```

```
cout << matrix[0][2];   // 3-й элемент 1-й строки (значение 3)
```

```
cout << matrix[2][1];   // 2-й элемент 3-й строки (значение 2)
```

```
matrix[1][3] = 10;      // изменение элемента (2-я строка, 4-й столбец)
```

Обработка двумерного массива

Чтобы обработать все элементы двумерного массива, используются два вложенных цикла:

внешний проходит по строкам,

а внутренний — по столбцам.

```
int matrix[3][4] = {
    {5, 2, 3, 9},
    {4, 1, 8, 3},
    {7, 2, 1, 4},
};
```

```
for (int i = 0; i < 3; i++) {        // строки
    for (int j = 0; j < 4; j++) {    // столбцы
        cout << matrix[i][j] << " ";
    }
    cout << "\n";
}
//  Результат работы программы:
//      5 2 3 9
//      4 1 8 3
//      7 2 1 4
```

Многомерные массивы

Помимо двумерных, в C++ можно создавать массивы с большим числом измерений.

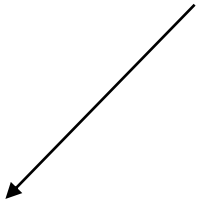
Каждое новое измерение добавляет уровень вложенности.

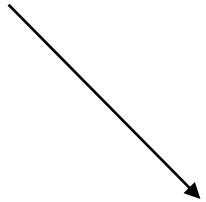
На практике чаще всего встречаются трёхмерные массивы.

```
int cube[2][3][4] = {
   {
      {1, 2, 3, 4},
      {5, 6, 7, 8},
      {9, 10, 11, 12}
   },
   {
      {13, 14, 15, 16},
      {17, 18, 19, 20},
      {21, 22, 23, 24}
   }
};
cout << cube[1][2][3];   // 24
```

Что такое строки?

Строка – это последовательность символов, образующих текст. В C++ существует два основных способа работы со строками.





Строки в стиле C

представляют собой массив символов типа char

```
char word[] = "Hello";
```

Тип данных std::string

современный способ работы со строками

```
string word = "Hello";
```

Особенности строк в стиле С

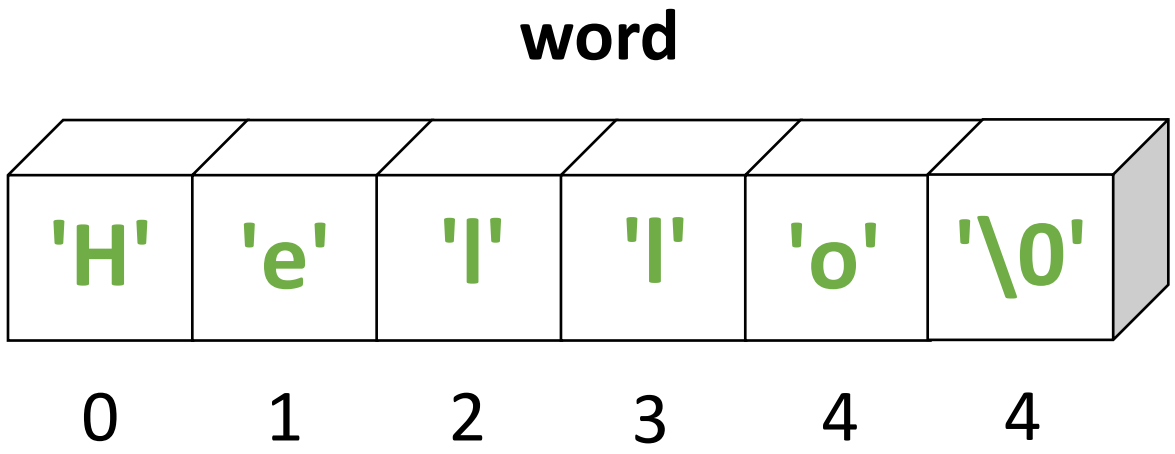
Строка в стиле С – это массив символов типа char, в конце которого обязательно стоит символ конца строки '\0' (нуль-символ).

Этот символ сообщает программе, где строка заканчивается.

Размер массива должен быть на один символ больше, чем длина строки.

```
char word[6] = "Hello";


cout << word;     // Hello
cout << word[1];  // e
```



Особенности типа string

Он входит в стандартную библиотеку и предоставляет удобные средства для работы с текстом.

Тип string автоматически управляет памятью и длиной строки.

Основные методы:

length() – длина строки;

substr(pos, len) – получение подстроки;

find(str) – поиск подстроки;

append(str) – добавление текста;

clear() – очистка строки.

```
#include <iostream>
#include <string>
int main() {
    string text = "Hello, ";
    text = text + "world!";
    cout << text << "\n";
    cout << text.length();   // длина строки
}
//    Результат работы программы:
//    Hello, world!
//    13
```

Пример – подсчет букв в строке

```
string text = "These trees are green.";
int count = 0;
for (int i = 0; i < text.length(); i++) {
    if (text[i] == 'e' || text[i] == 'E') {
        count++;
    }
}

cout << "Количество букв e и E: " << count;
// Вывод: Количество букв e и E: 9
```

Массивы строк

Массив строк используется, когда нужно хранить несколько текстовых значений одновременно – например, список имён или слов.

Каждый элемент массива является отдельной строкой.

```
#include <iostream>
#include <string>
int main() {
    string cities[3] = {"Москва", "Казань", "Самара"};
    for (int i = 0; i < 3; i++) {
        cout << cities[i] << "\n";
    }
}
//    Результат работы программы:
//    Москва
//    Казань
//    Самара
```

Что такое указатель?

Указатель – это переменная, которая хранит адрес другой переменной.

С помощью указателей можно обращаться к данным в памяти косвенно – по их адресу.

```
int x = 10;
int* p = &x;      // p хранит адрес переменной x

cout << x << "\n";      // 10
cout << p << "\n";      // например 0x61fe14
cout << *p << "\n";   // 10 — значение по адресу
```

Операции с указателями

С указателями можно выполнять операции получения адреса и разыменовывания, а также изменять значение, находящееся по адресу.

Основные операции:

& – получить адрес переменной;

* – разыменовать указатель (обратиться к значению по адресу);

присваивание – указателю можно присвоить адрес другой переменной того же типа.

```
int a = 5, b = 10;
int* p = &a;

cout << *p << "\n";   // 5

p = &b;          // теперь p указывает на b
cout << *p << "\n";   // 10

*p = 20;          // изменение значения b через указатель
cout << b << "\n";    // 20
```

Динамическое выделение памяти

Динамическая память используется, когда размер данных известен только во время выполнения программы.

В C++ для этого применяются операторы new и delete.

```
int* p = new int;      // выделение памяти под одно число
*p = 10;
cout << *p;            // 10
delete p;              // освобождение памяти
```

```
cin >> n;
int* arr = new int[n];  // динамический массив
arr[0] = 7;
delete[] arr;          // освобождение массива
```

Спасибо за внимание!