

Лекция 7. Массивы и строки

Одномерные и многомерные массивы, работа со строками C-style и класс `std::string`

1. Введение

Массивы и строки — фундаментальные структуры данных в C++.

Они позволяют хранить и обрабатывать **наборы однотипных элементов**, таких как числа, символы или объекты.

Понимание массивов и строк необходимо для эффективной работы с памятью и данными, а также для построения более сложных структур — векторов, матриц, списков и т.д.

2. Одномерные массивы

Массив — это непрерывная область памяти, где хранятся элементы одного типа.

Каждый элемент имеет **индекс**, по которому к нему можно обратиться.

Синтаксис объявления:

тип имя[размер];

Пример:

```
int numbers[5];      // массив из 5 элементов
numbers[0] = 10;
numbers[1] = 20;
numbers[2] = 30;
numbers[3] = 40;
numbers[4] = 50;
```

Можно инициализировать массив при объявлении:

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

Размер можно не указывать, если он выводится из количества элементов.

3. Доступ к элементам массива

Доступ осуществляется по индексу:

```
cout << nums[2]; // 3
```

Индексы начинаются с нуля.

Ошибкой является обращение к элементу за пределами массива:

```
nums[10] = 100; // Ошибка! Выход за границы массива
```

4. Перебор массива

Самый распространённый способ — с помощью цикла for:

```
for (int i = 0; i < 5; i++) {  
    cout << nums[i] << " ";  
}
```

В C++11 и новее можно использовать **диапазонный цикл**:

```
for (int n : nums) {  
    cout << n << " ";  
}
```

5. Многомерные массивы

Многомерные массивы позволяют хранить таблицы или матрицы данных.

Пример двумерного массива (матрицы):

```
int matrix[3][3] = {  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9}  
};
```

Доступ к элементам:

```
cout << matrix[1][2]; // 6
```

Перебор всех элементов:

```
for (int i = 0; i < 3; i++) {  
    for (int j = 0; j < 3; j++) {  
        cout << matrix[i][j] << " ";  
    }  
    cout << endl;  
}
```

6. Динамические массивы

Иногда размер массива заранее неизвестен.

В этом случае можно использовать **динамическое выделение памяти**:

```
int n;  
cin >> n;  
int* arr = new int[n];
```

Не забудьте освободить память:

```
delete[] arr;
```

Однако, в современном C++ предпочтительно использовать **вектор** (**std::vector**), который автоматически управляет памятью.

7. Строки в стиле C (C-style strings)

Строки в стиле C — это массив символов, **завершающийся нулевым символом '\0'**.

Пример:

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

Особенности:

- Строка word содержит 6 элементов: {'H', 'e', 'l', 'l', 'o', '\0'}.
- Функции для работы со строками находятся в <cstring>:
- #include <cstring>
- strlen(word); // длина строки
- strcpy(dest, src); // копирование
- strcat(a, b); // конкатенация
- strcmp(a, b); // сравнение

Пример:

```
char name[20];  
strcpy(name, "Alice");  
cout << strlen(name); // 5
```

8. Недостатки C-style строк

- Возможность переполнения буфера (если длина строки превышает размер массива).
- Неудобство при конкатенации и копировании.
- Неуправляемое использование памяти.

Для решения этих проблем был создан класс **std::string**, который значительно упрощает работу со строками.

9. Класс std::string

Библиотечный класс `std::string` (заголовок `<string>`) представляет строки как объекты с богатым набором функций.

Пример:

```
#include <string>
#include <iostream>
using namespace std;

int main() {
    string name = "Alice";
    cout << name << endl;
}
```

Основные операции:

```
string a = "Hello";
string b = "World";
string c = a + " " + b;    // Конкатенация
cout << c;                 // "Hello World"

cout << c.size();          // длина строки
cout << c[0];              // доступ по индексу
c[0] = 'h';                // изменение символа
```

10. Сравнение и поиск

Класс `std::string` поддерживает сравнение строк:

```
if (a == b) cout << "Равны";
```

Также можно искать подстроку:

```
string text = "Data mining is interesting";
size_t pos = text.find("mining");
if (pos != string::npos)
    cout << "Найдено на позиции " << pos;
```

11. Преобразование между string и C-style

Иногда требуется передать строку в функции, использующие C-style. Для этого используется метод `.c_str()`:

```
string str = "Пример";
printf("%s", str.c_str());
```

А для преобразования C-style строки в `std::string`:

```
char arr[] = "Hello";
string s(arr);
```

12. Примеры практического использования

Пример 1: поиск максимального элемента в массиве

```
int arr[] = {2, 8, 1, 5, 9};
int maxVal = arr[0];
for (int i = 1; i < 5; i++)
    if (arr[i] > maxVal) maxVal = arr[i];
cout << "Максимум: " << maxVal;
```

Пример 2: обработка строки

```
string s = "Программирование на C++";
for (char ch : s) {
    cout << ch << ' ';
}
```

13. Заключение

Массивы и строки — это основа работы с данными в C++.

Они позволяют хранить большие объёмы информации и эффективно управлять памятью.

Современный подход — использовать **векторы** вместо "сырых" массивов и **std::string** вместо строк C-стиля, так как они безопаснее и удобнее.

14. Вопросы для самопроверки

1. Что такое массив и как осуществляется доступ к его элементам?
2. Чем отличаются одномерные и двумерные массивы?
3. В чём разница между строками C-style и `std::string`?
4. Почему важно использовать нулевой символ `'\0'` в C-строках?
5. Какие преимущества предоставляет класс `std::string`?

15. Рекомендуемая литература

1. Стивен Прата — *Язык программирования C++. Лекции и упражнения.*
2. Бьерн Страуструп — *Язык программирования C++ (4-е издание).*
3. Nicolai M. Josuttis — *The C++ Standard Library: A Tutorial and Reference.*
4. Херб Саттер — *Exceptional C++.*
5. ISO/IEC 14882:2020 — *The C++20 Standard.*