

Лекция 8. Указатели и ссылки

Адресация, разыменование, передача по ссылке, арифметика указателей, динамическая память (new, delete)

1. Введение

Понимание **указателей** и **ссылок** — ключевой аспект владения языком C++. Они обеспечивают прямой доступ к памяти, позволяют работать с динамическими структурами данных, эффективно передавать объекты в функции и управлять ресурсами.

C++ предоставляет программисту мощные средства управления памятью, но требует осторожности, поскольку ошибки с указателями могут привести к сбоям и утечкам.

2. Адресация и указатели

Указатель — это переменная, хранящая **адрес** другой переменной.

Объявление указателя:

```
int a = 10;
int* p;    // объявление указателя на int
p = &a;    // оператор & возвращает адрес переменной
```

Теперь p хранит адрес a.

Например, если a находится по адресу 0x7ffeef, то p = 0x7ffeef.

Вывод адреса:

```
cout << &a << endl; // адрес переменной
cout << p << endl;  // тот же адрес
```

3. Разыменование указателя

Чтобы получить значение, на которое указывает указатель, используется оператор * (разыменование):

```
cout << *p << endl; // 10
```

Также через *p можно изменять значение переменной:

```
*p = 20;  
cout << a; // теперь a = 20
```

4. Типы указателей

Тип указателя должен соответствовать типу переменной, на которую он указывает:

```
double d = 3.14;  
double* pd = &d;    // правильно  
int* pi = &d;        // ошибка: разные типы
```

Можно объявить указатель на `void`, который может хранить адрес любого типа:

```
void* ptr;  
int x = 10;  
ptr = &x;
```

Но разыменовывать `void*` напрямую нельзя — сначала нужно привести к конкретному типу.

5. Указатели и массивы

Имя массива — это указатель на его первый элемент.

```
int arr[] = {10, 20, 30};  
int* p = arr;
```

Доступ к элементам:

```
cout << *p;    // 10  
cout << *(p + 1); // 20
```

Выражение `p + 1` сдвигает указатель на один элемент вперёд (в зависимости от размера типа).

Эквивалентность:

```
arr[i] == *(arr + i);
```

6. Арифметика указателей

C++ позволяет выполнять арифметические операции над указателями:

Операция	Описание
<code>r++</code>	переход к следующему элементу
<code>r--</code>	переход к предыдущему элементу
<code>r + n</code>	переход на n элементов вперёд
<code>r - n</code>	переход на n элементов назад

Пример:

```
int arr[] = {1, 2, 3, 4, 5};
int* p = arr;

for (int i = 0; i < 5; i++) {
    cout << *(p + i) << " ";
}
```

7. Передача параметров по указателю и по ссылке

7.1. Передача по указателю:

```
void change(int* ptr) {
    *ptr = 50;
}

int main() {
    int x = 10;
    change(&x);
    cout << x; // 50
}
```

7.2. Передача по ссылке:

```
void change(int& ref) {
    ref = 100;
}

int main() {
    int y = 10;
    change(y);
    cout << y; // 100
}
```

Отличие: ссылки безопаснее и удобнее, так как не требуют операторов & и * при использовании.

8. Ссылки (References)

Ссылка — это **альтернативное имя** (псевдоним) для уже существующей переменной.

После инициализации ссылка не может быть изменена, чтобы ссылаться на другую переменную.

Пример:

```
int a = 5;
int& ref = a;

ref = 10;
cout << a; // 10
```

Ссылки часто используются:

- для передачи аргументов в функции без копирования,
 - для возврата значений из функций,
 - в конструкторах копирования и операторах присваивания.
-

9. Динамическая память

Обычно память выделяется автоматически (на стеке), но иногда требуется выделить её **вручную** (в куче).

Для этого используются операторы new и delete.

9.1. Выделение памяти под одну переменную:

```
int* p = new int;
*p = 42;
cout << *p;
delete p; // освобождение памяти
```

9.2. Выделение памяти под массив:

```
int* arr = new int[5];
for (int i = 0; i < 5; i++) arr[i] = i;
delete[] arr; // обязательно с []
```

Если не освободить память — произойдёт **утечка памяти (memory leak)**.

10. Умные указатели (Smart Pointers, C++11 и новее)

В современном C++ рекомендуется использовать **умные указатели**, которые автоматически управляют памятью:

- `std::unique_ptr` — владеет объектом эксклюзивно;
- `std::shared_ptr` — совместное владение объектом;
- `std::weak_ptr` — слабая ссылка, не влияет на время жизни объекта.

Пример:

```
#include <memory>
using namespace std;

int main() {
    unique_ptr<int> ptr = make_unique<int>(10);
    cout << *ptr;
}
```

11. Указатели на функции

Можно хранить в указателе адрес функции и вызывать её через этот указатель.

Пример:

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

int main() {
    int (*funcPtr)(int, int) = add;
    cout << funcPtr(3, 4); // 7
}
```

12. Частые ошибки при работе с указателями

- **Разыменование нулевого указателя:**
 - `int* p = nullptr;`
 - `cout << *p;` // Ошибка!
- **Утечки памяти** — неосвобождённая память после `new`.
- **Dangling pointer** — указатель на уже удалённый объект.

- **Неверная арифметика указателей** — выход за границы массива.
-

13. Заключение

Указатели и ссылки — это основа управления памятью в C++.

Понимание этих механизмов необходимо для создания эффективных и безопасных программ.

Однако современные стандарты рекомендуют **использовать умные указатели** и **RAII-подход** (Resource Acquisition Is Initialization) для автоматического освобождения ресурсов.

14. Вопросы для самопроверки

1. Что такое указатель и как его объявить?
 2. В чём отличие между указателем и ссылкой?
 3. Что делает оператор * и оператор &?
 4. Как осуществляется динамическое выделение памяти в C++?
 5. Что такое умные указатели и зачем они нужны?
-

15. Рекомендуемая литература

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