



УКАЗАТЕЛИ В C++



ПЛАН

1. Понятие указателя, виды указателей в C++
2. Способы инициализации указателей
3. Операции с указателями в C++
4. Динамические массивы

Литература



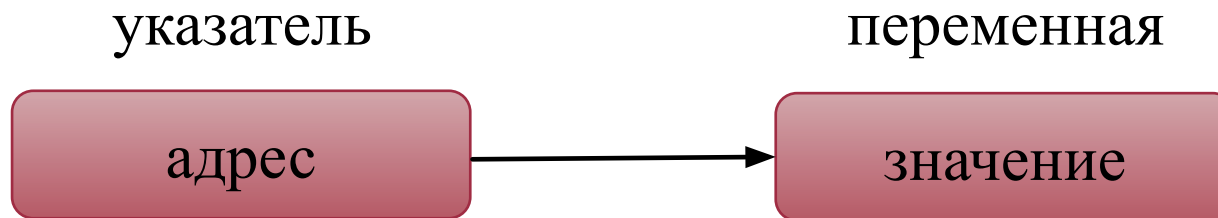
УКАЗАТЕЛЬ

- Когда компилятор обрабатывает оператор определения переменной (например, `int i=10;`), он выделяет память в соответствии с типом (`int`) и инициализирует ее указанным значением (`10`).
- Все обращения в программе к переменной по ее имени (`i`) заменяются компилятором на адрес области памяти, в которой хранится значение переменной.
- Программист может определить собственные переменные для хранения адресов областей памяти. Такие переменные называются указателями.



УКАЗАТЕЛЬ

Указатели предназначены для хранения адресов областей памяти.



«КУЧА»

- Указатели чаще всего используют при работе с динамической памятью, называемой «кучей» (англ. heap).
- Это свободная память, в которой можно во время выполнения программы выделять место в соответствии с потребностями.
- Доступ к выделенным участкам динамической памяти (*динамическим переменным*) производится только через указатели.
- Время жизни динамических переменных — от точки создания до конца программы или до явного освобождения памяти.



ВИДЫ УКАЗАТЕЛЕЙ В C++

Виды указателей

Указатель на объект

Указатель на функцию

Указатель на void



УКАЗАТЕЛЬ НА ФУНКЦИЮ

- Содержит адрес в сегменте кода, по которому располагается исполняемый код функции, то есть адрес, по которому передается управление при вызове функции.
- Указатели на функции используются для косвенного вызова функции (не через ее имя, а через обращение к переменной, хранящей ее адрес), а также для передачи имени функции в другую функцию в качестве параметра.



УКАЗАТЕЛЬ НА ФУНКЦИЮ

- *Указатель на функцию* имеет тип «указатель на функцию, возвращающую значение заданного типа и имеющую аргументы заданного типа»:

тип (*имя) (список_типов_аргументов);

- Например, объявление:

`int (*fun) (double, double);`

задает указатель с именем `fun` на функцию, возвращающую значение типа `int` и имеющую два аргумента типа `double`.



УКАЗАТЕЛЬ НА ОБЪЕКТ

- Содержит адрес области памяти, в которой хранятся данные определенного типа (основного или составного).
- Простейшее объявление указателя на объект (в дальнейшем «указателя») имеет вид:

ТИП *ИМЯ;

где тип может быть любым, кроме ссылки и битового поля.

- Можно определить указатель на указатель и др.



УКАЗАТЕЛЬ НА ОБЪЕКТ

- Звездочка относится непосредственно к имени, поэтому для того, чтобы объявить несколько указателей, требуется ставить * перед именем каждого из них.
- Например, в операторе: `int *a, b, *c;`
описываются два указателя на целое с именами `a` и `c`, а также целая переменная `b`.
- Размер указателя зависит от модели памяти.



УКАЗАТЕЛЬ НА VOID

- Применяется в тех случаях, когда конкретный тип объекта, адрес которого требуется хранить, не определен (например, если в одной и той же переменной в разные моменты времени требуется хранить адреса объектов различных типов).
- Указателю на `void` можно присвоить значение указателя любого типа, а также сравнивать его с любыми указателями, но перед выполнением каких-либо действий с областью памяти, на которую он ссылается, требуется преобразовать его к конкретному типу явным образом.



УКАЗАТЕЛЬ

- Указатель может быть константой или переменной, а также указывать на константу или переменную.
- Рассмотрим примеры:

`int i; //целая переменная`

`const int ci = 1; //целая константа`

`int *pi; //указатель на целую переменную`

`const int *pci; //указатель на целую константу`

`int *const cp = &i; //указатель-константа на целую
//переменную`

`const int *const cps = &ci; //указатель-константа на //целую
константу`



УКАЗАТЕЛЬ

- Модификатор `const`, находящийся между звездочкой и именем указателя, относится к самому указателю и запрещает его изменение, а `const` слева от звездочки задает постоянство значения, на которое он указывает.
- Для инициализации указателей использована операция получения адреса `&`.



ПРИСВАИВАНИЕ УКАЗАТЕЛЮ АДРЕСА СУЩЕСТВУЮЩЕГО ОБЪЕКТА

- с помощью операции получения адреса:

```
int a = 5; //целая переменная
```

```
int *p = &a; //в указатель записывается адрес a
```

```
int *p (&a); //то же самое другим способом
```

- с помощью значения другого инициализированного указателя:

```
int *r = p;
```

- с помощью имени массива или функции, которые трактуются как адрес:

```
int b[10]; //массив, имя массива хранит адрес первого элемента
```

```
int *t = b; //присваивание адреса начала массива
```

```
void f(int a ){ /* ... * / } // определение функции
```

```
void (*pf)(int); // указатель на функцию
```

```
pf = f; // присваивание адреса функции
```



ВЫДЕЛЕНИЕ УЧАСТКА ДИНАМИЧЕСКОЙ ПАМЯТИ И ПРИСВАИВАНИЕ ЕЕ АДРЕСА УКАЗАТЕЛЮ

тип_данных *имя_указателя = new тип_данных;

int *n = new int; //выделение памяти под величину типа
int

int *m = new int (10); //выделение памяти под величину
типа int, //инициализация выделенной динамической
памяти значением 10

int *q = new int [10]; //выделение памяти под массив
//из 10 целых чисел



ОСВОБОЖДЕНИЕ ПАМЯТИ

- Освобождение памяти, выделенной с помощью операции `new`, должно выполняться с помощью `delete`.
- При этом переменная-указатель сохраняется и может инициализироваться повторно. Приведенные выше динамические переменные уничтожаются следующим образом:

```
delete n; delete m; delete [ ] q;
```



ОПЕРАЦИИ С УКАЗАТЕЛЯМИ В C++

Операции с указателями

Разадресация

Присваивание

Сравнение

Приведение типов

Арифметические операции



РАЗАДРЕСАЦИЯ

- Операция разадресации, или разыменования, предназначена для доступа к величине, адрес которой хранится в указателе.
- Эту операцию можно использовать как для получения, так и для изменения значения величины (если она не объявлена как константа).

```
char a; //переменная типа char
```

```
char *p = new char; /* выделение памяти под указатель и под  
динамическую переменную типа char */
```

```
*p = 'Ю'; a = *p; // присваивание значения обоим переменным
```



ПРИСВАИВАНИЕ

- Указателю можно присвоить либо адрес объекта того же типа, либо значение другого указателя.
- Для получения адреса объекта используется операция **&**.
- Например:
`int a = 10;`
`int *pa = &a; // указатель pa хранит адрес переменной a`
- При этом указатель и переменная должны иметь один и тот же тип, в данном случае это тип `int`.



ПРИСВАИВАНИЕ

- Когда указателю присваивается другой указатель, то первый указатель начинает указывать на тот же адрес, на который указывает второй.

- Например:

```
int a = 10; int b = 2;
```

```
int *pa = &a; int *pb = &b;
```

```
pa = pb;
```

На какой адрес указывает pa?



НУЛЕВОЙ УКАЗАТЕЛЬ

- Нулевой указатель (*англ. null pointer*) – это указатель, который не указывает ни на какой объект.
- Для создания нулевого указателя можно применять следующие способы:

```
int *p2 = NULL;
```

```
int *p3 = 0;
```



СРАВНЕНИЕ

- К указателям могут применяться операции сравнения $>$, $>=$, $<$, $<=$, $==$, $!=$.
- Операции сравнения применяются только к указателям одного типа и к значениям `NULL`.
- Для сравнения используются адреса, на которые ссылаются указатели.



ПРИВЕДЕНИЕ ТИПОВ

- Иногда требуется присвоить указателю одного типа значение указателя другого типа.
- В этом случае следует выполнить операцию приведения типов с помощью операции

(тип_указателя *)

В скобках указывается тип, к которому следует выполнить преобразование.

- Например:

```
char c = 'N';
```

```
char *pc = &c;
```

```
int *pd = (int *)pc; //преобразование к int
```

```
void *pv = (void*)pc; //преобразование к void
```



АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

- Указатели могут участвовать в арифметических операциях (инкремент, декремент, сложение, вычитание).
- **Операция инкремента ++** увеличивает значение на единицу. В случае с указателем увеличение на единицу будет означать увеличение адреса, который хранится в указателе, на размер типа указателя.
- **Операция декремента --** уменьшает значение на единицу. В случае с указателем уменьшение на единицу будет означать уменьшение адреса, который хранится в указателе, на размер типа указателя.



АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

- Аналогично указатель будет изменяться при прибавлении/вычитании не единицы, а какого-то другого числа.
- Например, добавление к указателю типа `double` числа 2 будет означать, увеличение адреса, который хранится в указателе, два раза на размер типа указателя.



Напишите программу для вычисления значения выражения, используя указатели.

ПРИМЕР I

...

```
int main()
```

```
{ double *x, *y; //указатели на величины типа double
```

```
x = new double(10);
```

```
y = new double;
```

```
*y = (pow(*x, 2) - 7*(*x) + 10) /
```

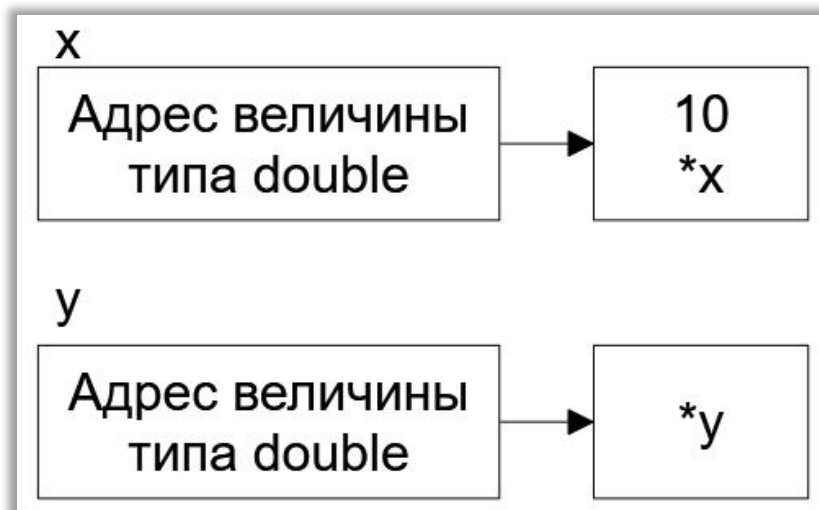
```
(pow(*x, 2) - 8*(*x) + 12);
```

```
cout << *y << endl;
```

```
delete x; delete y;
```

```
return 0; }
```

$$\frac{x^2 - 7x + 10}{x^2 - 8x + 12};$$



ПРИМЕР 2

Напишите программу для решения задачи, используя указатель на массив: если в массиве $A(10)$ есть элемент, равный кубу последнего элемента, то все элементы, следующие за ним, возвести в куб, иначе вывести массив без изменений.

```
#include "stdafx.h"
#include <iostream>
#include <cmath>
using namespace std;

void vmas(int *a, int k)
{ for (int i = 0; i < k; i++) cout << a[i] << " ";
  cout << endl; }
```



ПРИМЕР 2

```
int main()
{int n = 10;
int *mas = new int[n] {2,5,6,7,8,3,2,4,9,2}; //указатель на массив
vmas(mas, n); //ВЫВОД массива
int nmas = -1; //нет элементов равных кубу последнего элемента
for (int i = 0; i < n-1; i++)
if (mas[i] == pow(mas[n-1], 3)) {nmas = i; break;}
if (nmas != -1)
for (int i = nmas+1; i < n; i++) mas[i] = pow(mas[i], 3);
vmas(mas, n); //ВЫВОД массива
return 0;}
```



ДИНАМИЧЕСКИЕ МАССИВЫ

- При объявлении статического массива, его размером является числовая константа, например:

```
int n = 10;
```

```
int arr[n];
```

- Но в некоторых случаях изначально неизвестно сколько элементов окажется в массиве, например, их количество задаст пользователь.
- В этом случае используются *динамические массивы*.



СОЗДАНИЕ ОДНОМЕРНОГО ДИНАМИЧЕСКОГО МАССИВА

Динамические массивы
нельзя инициализировать
при создании.

```
int main()
{   int num; //размер массива
    cout << "Enter integer value: ";
    cin >> num; //получение от пользователя размера массива
    int *p = new int[num]; //выделение памяти для массива
    for (int i = 0; i < num; i++) //заполнение и вывод массива
    { p[i] = i;
      cout << "Value of " << i << " element is " << p[i] << endl; }
    delete [] p; //очистка памяти
    return 0; }
```



ДОСТУП К ЭЛЕМЕНТАМ ДИНАМИЧЕСКОГО МАССИВА

- Доступ к элементам динамического массива осуществляется так же, как к статическим.
- Например, к элементу номер 5 приведенного ранее массива можно обратиться как `p[5]` или `*(p+5)`.



СОЗДАНИЕ ДВУМЕРНОГО ДИНАМИЧЕСКОГО МАССИВА

...

```
int nstr, nstb;
```

```
cout « "Введите количество строк и столбцов :";
```

```
cin » nstr » nstb;
```

```
int **a = new int *[nstr]; // 1
```

```
for (int i = 0; i < nstr; i++) // 2
```

```
a[i] = new int [nstb]; // 3
```

...



СОЗДАНИЕ ДВУМЕРНОГО ДИНАМИЧЕСКОГО МАССИВА

- В операторе 1 объявляется переменная типа «указатель на указатель на int» и выделяется память под массив указателей на строки массива (количество строк — `nstr`).
- В операторе 2 организуется цикл для выделения памяти под каждую строку массива.
- В операторе 3 каждому элементу массива указателей на строки присваивается адрес начала участка памяти, выделенного под строку двумерного массива.
- Каждая строка состоит из `nstb` элементов типа `int`.



СОЗДАНИЕ ДВУМЕРНОГО ДИНАМИЧЕСКОГО МАССИВА

Павловская Т.А. - С и С++, Программирование на языке высокого уровня.pdf - Adobe Acrobat Reader DC

Файл Редактирование Просмотр Окно Справка

Главная Инструменты Павловская Т.А. - ... x

63 / 461 182%

Глава 1. Базовые средства языка C++ 63

ваается адрес начала участка памяти, выделенного под строку двумерного массива. Каждая строка состоит из $nstb$ элементов типа `int` (рис. 1.10).

Рис. 1.10. Выделение памяти под двумерный массив

Добавить комментарий

Заполнить и подписать

Храните файлы и обменивайтесь ими в Document Cloud

Подробнее

22:18 27.03.2016

