

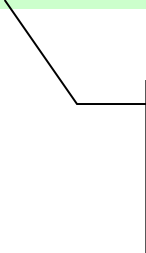
УКАЗАТЕЛИ И ССЫЛКИ

Применение указателей и ССЫЛОК

- ◆ Доступ к элементам массива
- ◆ Передача аргументов в функцию, которая должна изменить эти аргументы
- ◆ Возвращение значений из функции
- ◆ Передача в функции массивов и строковых переменных
- ◆ Выделение памяти
- ◆ Создание сложных структур (связный список, дерево и т.п.)

Указатели

Указатели – переменные, предназначенные для хранения адресов областей памяти

- 
- 1) на объект;
 - 2) на **void**;
 - 3) на функцию.

Указатель не является самостоятельным типом, он всегда связан с к.л. другим конкретным типом

Указатель на объект содержит адрес области памяти, в которой хранятся данные определенного типа (основного или составного)

```
тип *ИМЯ;  
  
int *a, b, *c;
```

Указатель на void применяется в тех случаях, когда конкретный тип объекта, адрес которого требуется хранить, не определен

Указателю на void можно присвоить значение указателя любого типа, а также сравнивать его с любыми указателями, но перед выполнением к.л. действий с областью памяти, на которую он ссылается, требуется преобразовать его к конкретному типу явным образом:

```
void *q;  
float *q = (float *) malloc(100 * sizeof(float));
```

Указатель функции имеет тип «указатель функции, возвращающей значение заданного типа и имеющей аргументы заданного типа»:

косвенный вызов функций

Указатель на функцию содержит адрес в сегменте кода, по которому располагается исполняемый код функции, т.е. адрес, по которому передается управление при вызове функции.

```
тип (*имя) (список_типов_аргументов);  
int (*fun) (double, double);
```

Указатель может быть константой или переменной, а также указывать на константу или переменную

```
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;   //константный указатель на целую
                      //переменную
const int * const cps = &ci; //константный указатель на целую
                      //константу
```

Модификатор **const**, находящийся между именем указателя и символом «*», относится к самому указателю и запрещает его изменение.

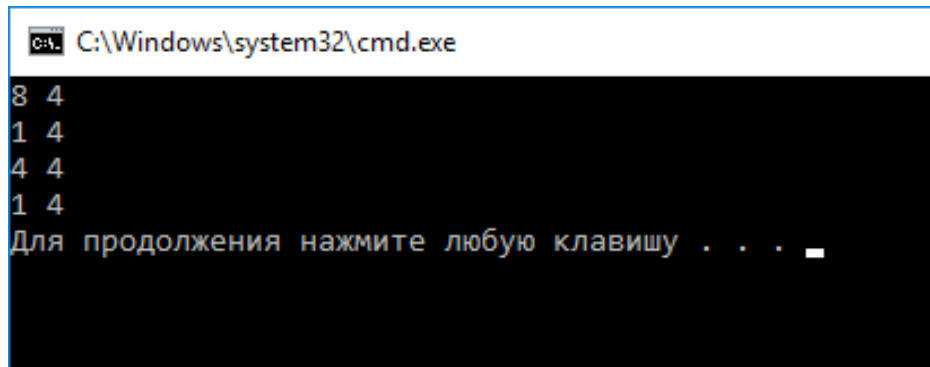
Модификатор **const** слева от символа «*» задает постоянство значения, на которое он указывает.

Для инициализации указателей используется операция получения адреса «&».

А сколько занимает места в памяти сам указатель?

```
cout<<sizeof(double)<<" "<<sizeof(double *)<<endl;  
cout<<sizeof(char)<<" "<< sizeof(char *)<<endl;  
cout<<sizeof(int)<<" "<<sizeof(int *)<<endl;  
cout<<sizeof(bool)<<" "<<sizeof(bool *)<<endl;
```

Проверьте



A screenshot of a Windows command prompt window. The title bar shows the path C:\Windows\system32\cmd.exe. The command prompt has a black background with white text. It displays the output of the C++ program: four lines of memory sizes, each consisting of two numbers separated by a space, and a final prompt line in Russian.

```
C:\Windows\system32\cmd.exe  
8 4  
1 4  
4 4  
1 4  
Для продолжения нажмите любую клавишу . . .
```

зависит от разрядности аппаратной платформы

Инициализация указателей

Указатели чаще всего используются при работе с динамической памятью, называемой «**куча**» (*heap* (англ.) – куча)

Куча (динамическая память) – свободная память .в которой можно во время выполнения программы выделять место в соответствии с потребностями

Доступ к выделенным участкам памяти (**динамическим переменным**) производится только через указатели

Время жизни динамических переменных – от точки создания до конца программы или до момента явного освобождения памяти.

Способы работы с динамической памятью:

- 1) при помощи семейства функций `malloc` (стандарт для C);
- 2) при помощи операций **new** и **delete** (C++).

Способы инициализации указателя:

1. Присвоение указателю адреса существующего объекта:

с помощью операции получения адреса:

```
int a = 5;           //целая переменная
int *p = &a;         //запись адреса a в указатель
int *p (&a);          //то же
cout<<*p; /*разыменовывание указателя (получение значения по
адресу, который он хранит*/
```

с помощью значения другого инициализированного указателя:

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

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

```
int b[10];                                //массив
int *t = b;                               //присвоение адреса начала
                                           //массива
.....
void f(int a) { /*.....*/};              //определение функции
void (*pf) (int);                         //указатель на функцию
pf = f;                                   //присвоение адреса функции
```

2. Присвоение указателю адреса области памяти в явном виде:

```
char *vp = (char *)0xB8000000;
```


3. Присвоение пустого значения:

```
int *mistake = NULL;    //проблемы с приведением типов  
int *correct = 0;       //рекомендуется к использованию
```

4. Выделение участка динамической памяти и присвоение ее адреса указателю:

с помощью операции new:

```
int *n = new int;                //1  
int *m = new int(10);            //2  
int *q = new int[10];            //3
```

с помощью функции malloc:

```
#include <malloc.h>  
int *u = (int *)malloc(sizeof(int)); //4
```

1. Операция **new** выполняет выделение достаточного для размещения величины типа **int** участка динамической памяти и записывает адрес начала этого участка в переменную *n*. Память под саму переменную *n* (размера, достаточного для размещения указателя) выделяется на этапе компиляции.
2. В дополнение к действиям, описанным в п. 1 производится инициализация выделенной динамической памяти значением 10.
3. Операция **new** выполняет выделение памяти под 10 величин типа **int** (массив из 10 элементов) и записывает адрес начала этого участка в переменную *q*, которая может трактоваться как имя массива. Через имя можно обратиться к любому элементу массива.
4. Аналогично п. 4, но с помощью функции выделения памяти **malloc**. Параметр функции – количество выделяемой памяти в байтах. Конструкция (**int***) используется для явного приведения типа указателя, возвращаемого функцией.

Освобождение памяти, выделенной с помощью операции **new** должно выполняться с помощью **delete**, а памяти, выделенной функцией **malloc()** – посредством функции **free()**

После освобождения памяти переменная-указатель сохраняется и может инициализироваться повторно.

```
delete n;  
  
delete m  
  
delete []q;  
  
free(u);
```

Ссылки

Ссылка представляет собой *синоним имени*, указанного при инициализации ссылки

Ссылку можно трактовать как указатель, который всегда разыменовывается

```
тип & имя;  
  
int kol;  
  
int & pal = kol;           //ссылка pal - альтернативное имя для kol  
  
const char & CR = '\n';   //ссылка на константу
```

Правила

- Переменная-ссылка должна явно инициализироваться при ее описании, кроме случаев, когда она является параметром функции, описана как `extern` или ссылается на поле данных класса
- После инициализации ссылке не может быть присвоена другая переменная
- Тип ссылки должен совпадать с типом величины, на которую она ссылается
- Не разрешается определять указатели на ссылки, создавать массивы ссылок и ссылки на ссылки
- Ссылка не занимает дополнительного пространства в памяти и является альтернативным именем величины.
- Операции над ссылкой приводят к изменению величины, на которую она ссылается.
- Ссылки применяются чаще всего в качестве параметров функций и типов возвращаемых функциями значений.