УКАЗАТЕЛИ И ДИНАМИЧЕСКАЯ ПАМЯТЬ

Динамическая память – неименованная область памяти
Указатель — хранит адрес другого объекта (составной тип данных С++)

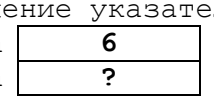
Объявление указателей

тип_указателя * имя_указателя;

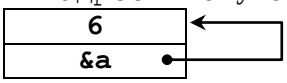
```
int   *ptr;  // в стиле C
int*   ptr;  // в стиле C++
int*ptr;
```
НО!
```
int* p1, p2; // объявление одного указателя на int
             // и одной переменной типа int
```

Инициализация указателей (1 способ)

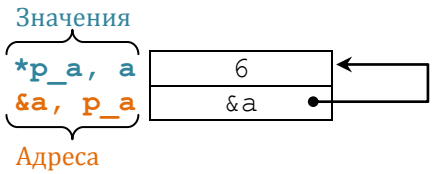
```
int a =6;   // объявление переменной
int * p_a;  // объявление указателя на int
```




```
p_a = &a;   // присвоить адрес int указателю
```




```
cout << a << *p_a;   // 6  6              - Выразили значения двумя способами
cout << &a << p_a;   //0x23ef20  0x23ef20  - Выразили адреса двумя способами
```



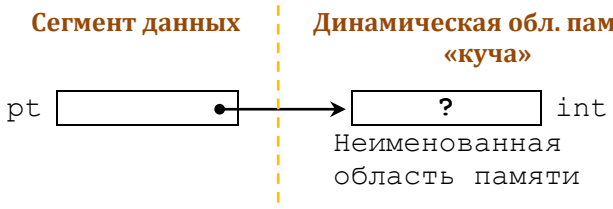

```
*p_a = *p_a + 1; // Изменить значение через указатель
cout << a;       // 7
```

ОШИБКА!
```
char * p_ch;
*p_ch = 'A';  // не определено место сохранения значения
```

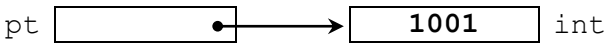
Инициализация указателей (2 способ)

имяТипа * имя_указателя = new имяТипа;

```
int * pt = new int;          // выделение пространства для int
                             // в "куче" или динамической области памяти
```




```
*pt = 1001;                  // сохранение значения
cout << sizeof(pt) << sizeof(*pt);  // 8  4
```



null-указатель (нулевой указатель) – ***nullptr***.

```
pt = nullptr;
```
pt **nullptr**

Освобождение памяти с помощью операции delete

delete имя_указателя;

```
int * ps = new int; // выделить память с помощью операции new
```
ps → **?** int

```
...                 // использовать память
delete ps; // по завершении освободить память
           // с помощью операции delete
```
ps **?** (зачеркнуто)

```
ps = nullptr;
```
ps **nullptr**

Особенности delete:

```
int * ps = new int; // нормально
delete ps;          // нормально
delete ps;          // теперь - не нормально!
                    // Вызовет ошибку времени выполнения программы

int jugs = 5;       // нормально
int * pi = &jugs;   // нормально
delete pi;          // не допускается, память не была выделена new
                    // Вызовет ошибку времени выполнения программы

int * ps = new int; // выделение памяти
int * pq = ps;      // установка второго указателя на тот же блок
delete pq;          // вызов delete для второго указателя
```

Создание/удаление динамических массивов

*имяТипа * имя_указателя =* ***new имяТипа [количество_элементов];***

delete [] *имя_указателя;* ***// освобождение динамического массива***

Особенности:

```
int * pt = new int;
short * ps = new short [500];
delete [] pt; // эффект не определен, не делайте так
delete ps;    // эффект не определен, не делайте так
```

спользование динамического массива

```
double * pm = new double [3]; // пространство для 3 значений double
pm[0] = 0.2;                  // трактовать p3 как имя массива
pm[1] = 0.5;
pm[2] = 0.8;

cout << pm[1]; // вывод p3[1]
   //   0.5
pm = pm + 1;   // увеличение указателя
cout << pm[0] << pm[1];
   //   0.5      0.8
pm = pm - 1;   // возврат указателя в начало
delete [] pm;  // освобождение памяти
```

АРИФМЕТИКА УКАЗАТЕЛЕЙ

Операторы для работы с указателями

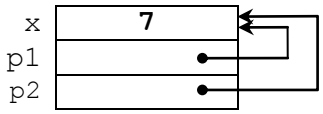
– *получения адреса* **&**;
– *разыменования указателя* *****.

Выражения с указателями

①Присваивание указателей

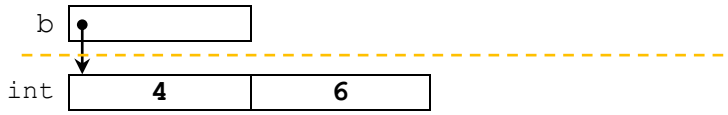
Указатель можно присваивать другому указателю.

```
int x = 7;
int *p1, *p2;
p1 = &x;
p2 = p1; // Теперь на переменную x ссылаются оба указателя p1 и p2.
```

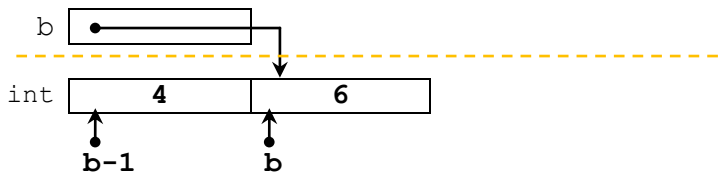


②Сложение и вычитание указателей

```
int * b = new int [2];  // динамический массив
b[0]=4; b[1]=6;
```



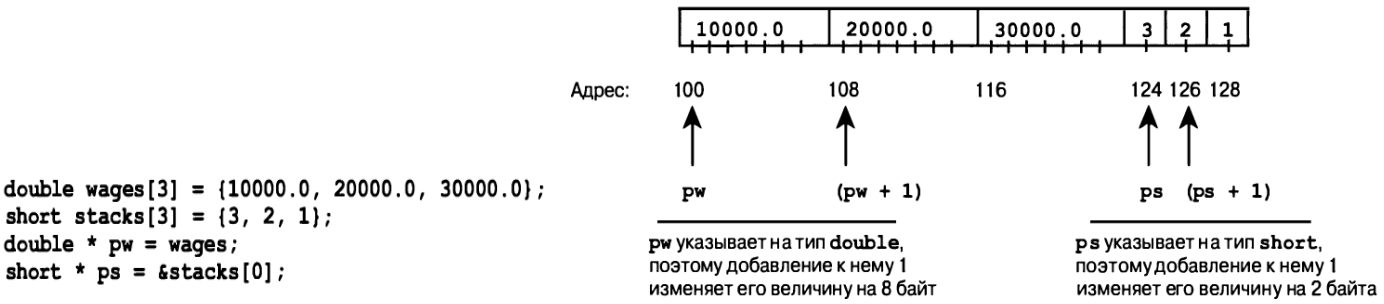

```
cout << b;
b = b+1;  // изменение *b – адреса
cout << b[0] << b[-1];
     // 6       4
```




```
b--;          // вернуть b исходное значение
delete [] b;  // для верного освобождения памяти

int a[2]={3,5};  // статический массив
```

```
// поскольку a – указатель, можно попробовать его увеличить на 1
// по аналогии с арифметикой указателей
a = a+1;
// error: incompatible types in assignment of 'int*' to 'int [2]'
cout << *a << *(a+1); // 3 5 – имя массива является указателем
// НО!!!
cout << *a+1; // 4 – т.к. приоритет * выше, чем у +
```



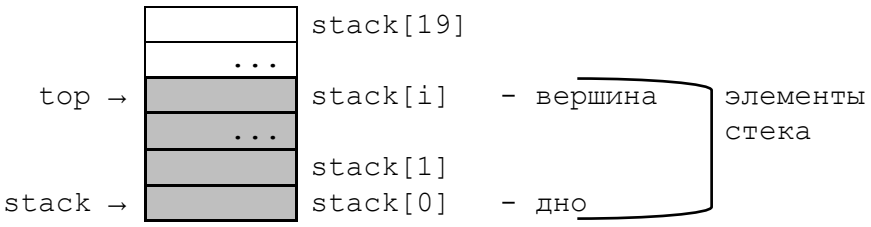
③Сравнение указателей

```
if (p<q)
   cout << "Указатель p содержит меньший адрес, чем указатель q\n";

double *pd = new double;
char * pc  = new char;
cout << ( pd < (double*)pc ) << ( (char*) pd < pc);
```

Пример 1. Стек – базовая структура данных

```
const int size=20;
int stack[size];
int * top;
```



Пример 2. Очередь – базовая структура данных

См. лекцию и видео-материал.

Исключение из правила:
Имя массива – адрес первого элемента массива.

Исключение 1. Адрес массива.
Операции & возвращает адрес целого массива:

```
int a[2];
cout << a ;  // отображение &a[0] – адреса первого элемента массива
cout << &a;  // отображение адреса целого массива
//  0x23fdf0  0x23fdf0

cout << a+1 << " " << &a+1;
//  0x23fdf4  0x23fdf8
```

Продолжение ***примера*** в лекции от 27.03.

Исключение 2. Размер массива. Операция sizeof.
- к имени статического массива возвращает размер массива в байтах,
- к указателю возвращает размер указателя, даже если он указывает на массив.

Пример в лекции от 27.03.

УКАЗАТЕЛИ И СТРОКИ В СТИЛЕ С

Специальные отношения между массивами и указателями расширяют строки в стиле С.

Пояснение и ***примеры*** в лекции от 27.03.

!!! С объектом cout имя массива из элементов `char`, указатель на `char` и строковая константа в кавычках — все интерпретируются как адрес первого символа строки.

!!! При вводе строки внутри программы всегда необходимо использовать адрес ранее распределённой памяти. Этот адрес может иметь форму имени массива либо указателя, инициализированного с помощью операции `new`.

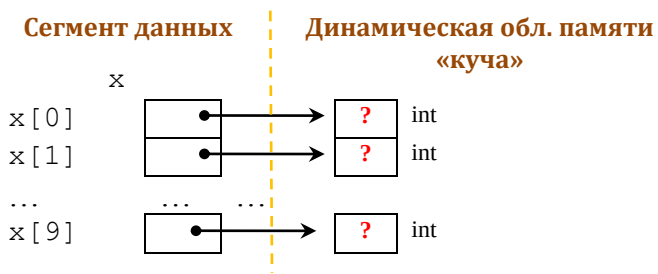
!!! Для копирования строки в массив применяйте `strcpy()` или `strncpy()`, а не операцию присваивания.

Лабораторная работа №4 «Динамические переменные»!!!

СЛОЖНЫЕ СТРУКТУРЫ ДАННЫХ НА ОСНОВЕ УКАЗАТЕЛЕЙ

I. *Массивы указателей (каталоги) на динамические элементы простых типов*

```
int *x[10] {};  // массив из 10 элементов,
                // каждый из к-ых – указатель на int
                // значения == ?
```




```
cout << * x[2]; // вывод значения 3го элемента
```

Пример 1.

```
const int size = 10;
void FillArray(int *p[]);// передача в функцию – по имени

int main()
{
   int *x[size]{};    // объявление и инициализация x
   FillArray(x);      // создание элементов и их инициализация
   for (int i=0; i<size; i++)
      cout << *x[i] << " ";
   for (int i=0; i<size; i++)
      delete x[i];
   return 0;
}
void FillArray(int *p[]){
   for (int i=0; i<size; i++) {
      p[i] = new int;  // создание элементов
      *p[i] = i;       // и их инициализация
   }
}
```