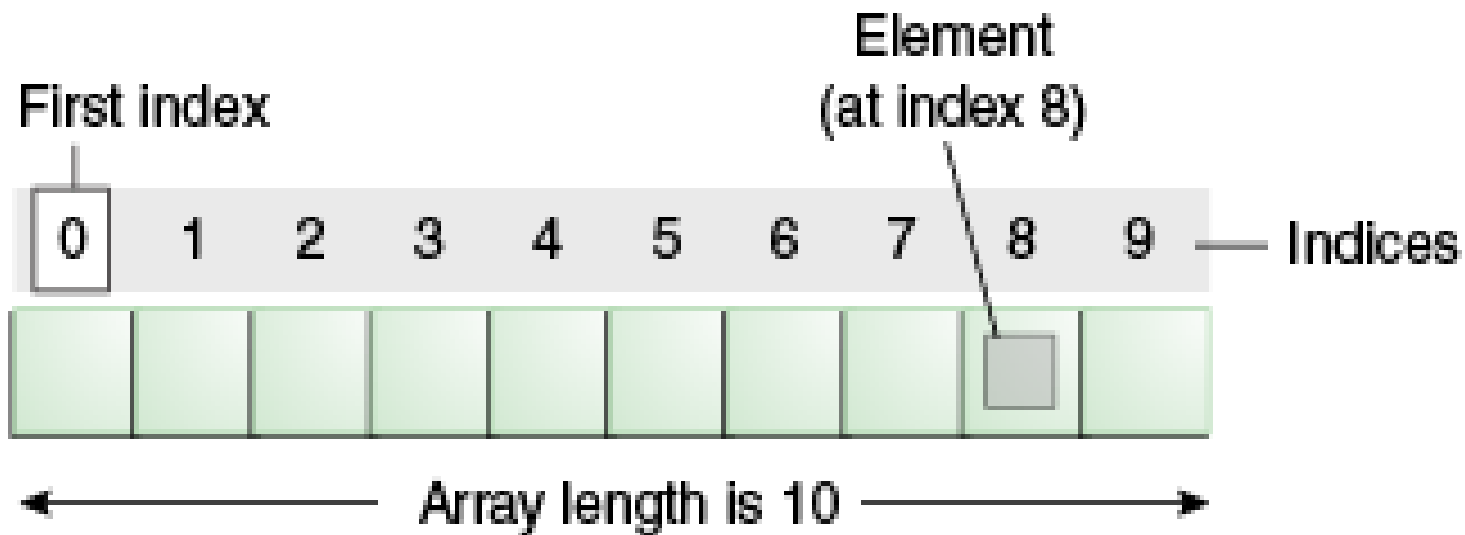


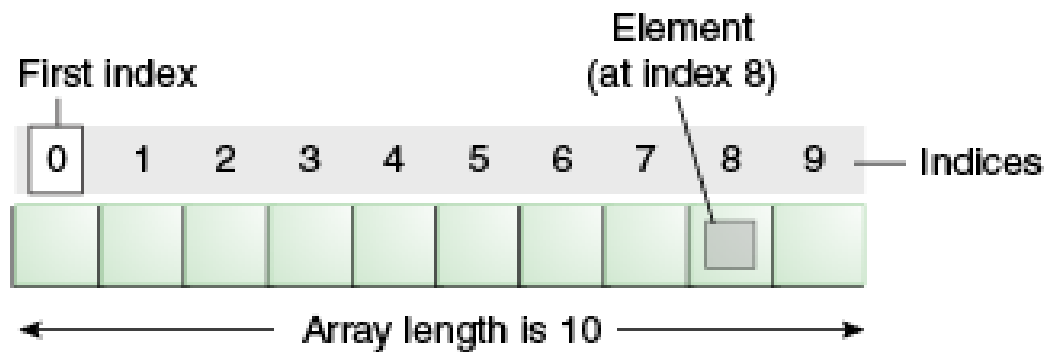
Типы данных

Простые (встроенные)	Структурированные (пользовательские, создаваемые программистом)
int (signed, unsigned, short, long) double, float char bool	Массивы Перечисления Структуры Классы

Массивы

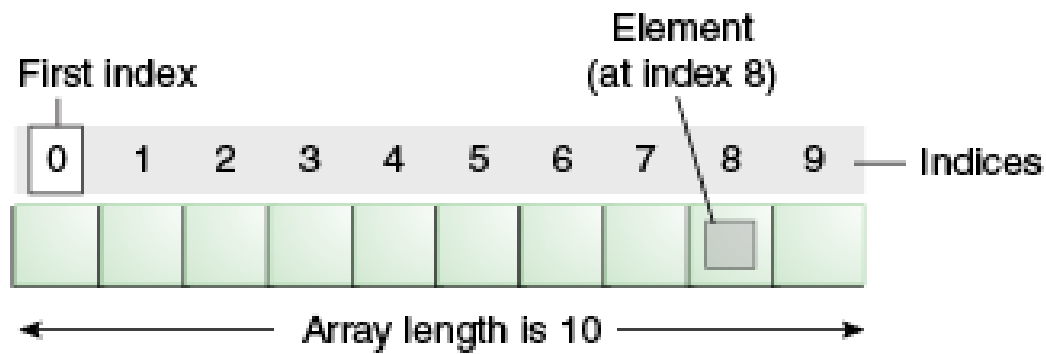
Массив – набор **однотипных элементов**, который расположен **в памяти единым блоком**, его **элементы следуют непосредственно друг за другом** и доступ к элементам - по **индексу** или **набору индексов**





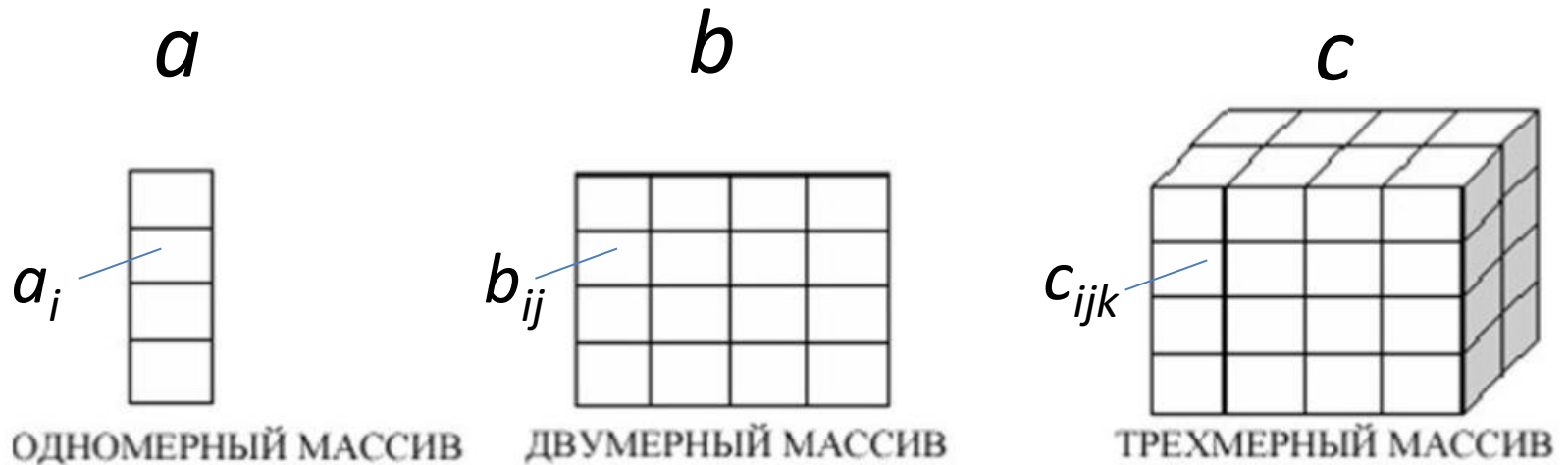
Свойства

- единое целое, имеет одно имя;
размещается в памяти одним блоком,
элементы следуют друг за другом
- все элементы одного типа
- все элементы пронумерованы (индексация в С-языках начинается с 0)
- **доступ к элементу по имени массива и индексу элемента**
 - чтение элемента по индексу, за $O(1)$
 - запись значения в элемент по индексу, за $O(1)$



Размер массива – количество элементов в нем

Размерность массива – количество индексов (номеров), используемых для доступа к одному элементу



Массивы:

- одномерные
- многомерные

Массивы:

- автоматические (встроенные, обычные, статические)
- динамические

Встроенные одномерные массивы

Объявление и инициализация

```
// объявлены, не инициализированы
```

```
int a[5];
```

```
double b[10];
```

```
char symbols[100];
```

```
int c[5], f, g = 12, d[20];
```

```
// инициализированы
```

```
int x[5] {7, -4, 12, 8, 1};
```

```
int y[5] {7, -4}; // {7, -4, 0, 0, 0}
```

```
double yy[5] {0};
```

```
int z[] {7, -4, 12, 8, 1};
```

```
// int w[2] {7, -4, 12, 8, 1}; // ошибка
```

```
int r[] {0};
```

Объявление и инициализация с именованными константами или constexpr

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    const int n = 15;
```

```
    constexpr int m = 105 * n;
```

```
    const size_t size = 1000;
```

```
    constexpr size_t len = 1000 + 12 * n;
```

```
    int aa[n];
```

```
    bool bb[m];
```

```
    char cc[size];
```

```
    double dd[len];
```

```
    ...
```

```
    int k;
```

```
    cin >> k;
```

```
    double t[k]; // ошибка
```


Длина простого массива = количество элементов в нем

```
int numbers[]{11, 12, 13, 14};
```

```
// ....
```

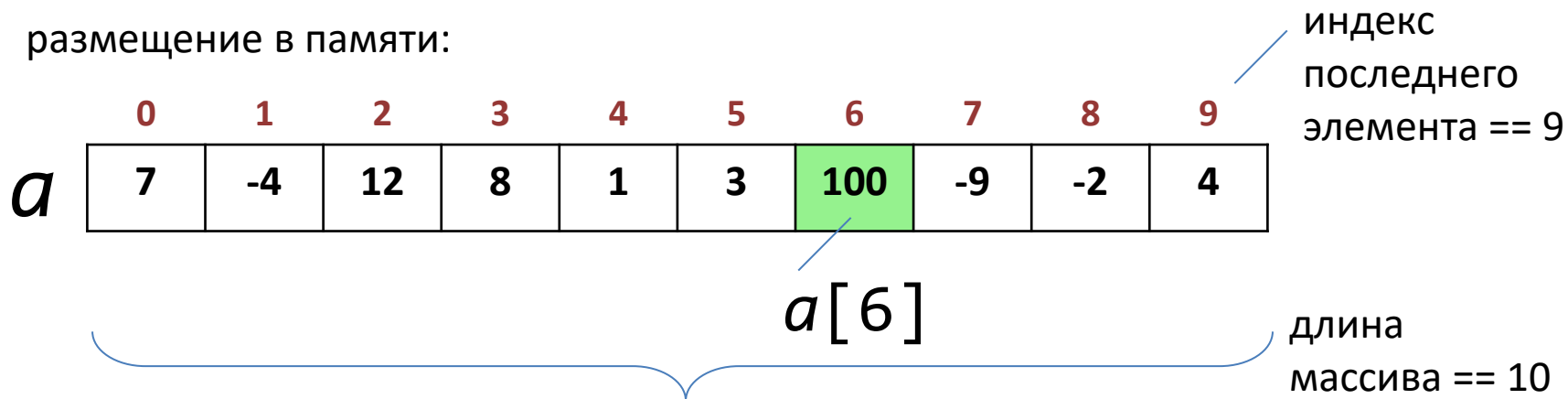
```
int length = sizeof(numbers) / sizeof(numbers[0]); //4
```

```
// или
```

```
size_t length = std::size(numbers);
```

Одномерные массивы – доступ к отдельному элементу

размещение в памяти:



```
int a[10] {7, -4, 12, 8, 1, 3, 100, -9, -2, 4};
```

a[0] – первый элемент массива

a[3] – четвертый по порядку элемент

```
int k = std::size(a) - 2; // выражение для расчета индекса элемента
```

a[*k*] – элемент с индексом *k*, находится в массиве под номером *k*+1

a[10] – ошибка, такого элемента нет!

Но при компиляции это не отслеживается,
программа будет работать, обращаясь к этой ячейке памяти!

Одномерные массивы – проход по всему массиву

0	1	2	3	4	5	6	7	8	9
7	-4	12	8	1	3	100	-9	-2	4

```
int a[10] {7, -4, 12, 8, 1, 3, 100, -9, -2, 4};

for (int i = 0; i < 10; i++)
{
    ....a[i].... // ... операции с элементом a[i]
                  // как с переменной своего типа
}
```

```
for (int i = 0; i < 10; i++)
{
    cout << a[i] << " ";
}
```

желательно использовать не литеральные константы, а типизированные или constexpr

```
const int count = 10;
int a[count] {7, -4, 12, 8, 1, 3, 100, -9, -2, 4};

for (int i = 0; i < count; i++)
{
    //a[i]  // ... операции с элементом a[i]
}
```

```
int a[] {7, -4, 12, 8, 1, 3, 100, -9, -2, 4};

size_t count = std::size(a);

for (size_t i = 0; i < count; i++)
{
    //a[i]  // ... операции с элементом a[i]
}
```

Перебор части элементов, если индексы удовлетворяют какой-либо закономерности

```
int a[] {7, -4, 12, 8, 1, 3, 100, -9, -2, 4};

size_t count = std::size(a);

for (size_t i = count / 2; i < count; i++)
{
    cout << a[i] << " ";
}
```

```
int a[] {7, -4, 12, 8, 1, 3, 100, -9, -2, 4};

size_t count = std::size(a);

for (size_t i = 1; i < count; i += 2)
{
    cout << a[i] << " ";
}
```

Другой порядок перебора:

```
int a[] {7, -4, 12, 8, 1, 3, 100, -9, -2, 4};

size_t count = std::size(a);

for (size_t i = count - 1; i >= 0; i--)
{
    cout << a[i] << " ";
}
```

Перебор всех элементов циклом for-each

```
double a[] { 7.0, -4.4, 12.1, 8., .1, 3.2, 10.0, -9., -2.4, 4.};
```

```
for (double a_elem : a)
```

```
{
```

```
    // будут пройдены все элементы массива
```

```
    // изменить порядок прохода нельзя
```

```
    // a_elem – обращение к отдельному элементу в массиве
```

```
    // доступ только на чтение, изменить элемент нельзя
```

```
    .... a_elem .....
```

```
}
```

Перебор всех элементов циклом for-each

```
int a[]{ 7, -4, 12, 8};
```

```
size_t count = std::size(a);
```

```
for (int a_elem : a)
{
    a_elem = 5;
    cout << a_elem;
    // a_elem – обращение к копии элемента массива
    // доступ только на чтение,
    // можно изменить локальную переменную a_elem
    // нельзя изменить значение в массиве
}
```

```
for (size_t i = 0; i < count; i++)
{
    cout << a[i];
}
```


Заполнение массива случайными числами

```
srand(time(NULL)); //инициализирует генератор случ.чисел текущим системным временем
```

```
cout << "RAND_MAX = " << RAND_MAX << endl;
```

```
cout << "Целочисленный массив случайных чисел из диапазона [0, RAND_MAX]" << endl;
```

```
int a[10];
```

```
for(int i = 0; i < 10; i++) {  
    a[i] = rand();    //числа в диапазоне [0, RAND_MAX]  
    cout << a[i]<<endl;
```

```
}
```

```
int a = 7, b = 15; // границы диапазона, из которого будут случайные числа
```

```
cout << endl << "a = " << a << " b = " << b << endl ;
```

```
cout << "Целочисленный массив случайных чисел из диапазона [a, b]" << endl;
```

```
int b[10];
```

```
for(int i = 0; i < 10; i++) {  
    b[i] = rand()%(b - a + 1) + a;    //целые числа в диапазоне [a, b]  
    cout << b[i]<<endl;
```

```
}
```

```
cout << "Вещественный массив случайных чисел из диапазона [a, b]" << endl;
```

```
double aa = 7, bb = 15; // границы диапазона, из которого будут случайные числа
```

```
double c[10];
```

```
for(int i = 0; i < 10; i++) {  
    //вещественные числа в диапазоне [aa, bb]  
    c[i] = rand()*(bb - aa)/(1 + RAND_MAX) + aa;  
    cout << c[i]<<endl;
```

```
}
```

Пример:

В течение суток через каждый час измеряли температуру воздуха. Ввести эти данные и разместить их в одномерном массиве

Вывести

- а) температуру в девять утра и девять вечера
- б) среднесуточную температуру
- в) количество отрицательных
- г) максимальную температуру
- д) минимальная температуру и час, когда она была
- е) часы, когда была нулевая температура
- ж) новый массив, содержащий те же температуры, но в градусах Фаренгейта

Многомерные массивы

логическое представление:

индексы второго измерения **j**

индексы первого измерения **i**

	0	1	2	3
0	45	-7	12	4
1	10	3	100	-9
2	-2	4	18	9

элемент с индексами $[i][j]$
 $i=1, j=2$

размещение в памяти:

45	-7	12	4	10	3	100	-9	-2	4	18	9
----	----	----	---	----	---	-----	----	----	---	----	---

Многомерные массивы

— реализованы в C++ как **массивы из массивов**

// инициализация многомерных массивов

```
int b[n][m] = { {45, -7, 12, 4}, {10, 3, 100, -9}, {-2, 4, 18, 9} };
```

```
int bb[][m] = { {45, -7, 12, 4}, {10, 3, 100, -9}, {-2, 4, 18, 9} };
```

// ошибка, вторая и последующие размерности должны быть указаны явно

```
//int bb[][] = { {45, -7, 12, 4}, {10, 3, 100, -9}, {-2, 4, 18, 9}};
```

```
int c[3][2][4] = { {{45, -7, 12, 4}, {45, -7, 12, 4}},  
                  {{10, 3, 100, -9}, {10, 3, 100, -9}},  
                  {{-2, 4, 18, 9}, {-2, 4, 18, 9}},  
};
```

Многомерные массивы – обращение к отдельным элементам

– реализованы в C++ как **массивы из массивов**

```
constexpr int n = 3, m = 4;
```

```
int a[n][m];
```

```
a[0][0] = 45;
```

```
a[0][1] = -7;
```

```
a[0][2] = 12;
```

```
a[0][3] = 4;
```

```
a[0][4] = 1000000; // ошибка, выход индекса за границы
```

```
a[1][0] = 10;
```

```
a[1][1] = 3;
```

```
a[1][2] = 100;
```

```
a[1][3] = -9;
```

```
a[2][0] = -2;
```

```
a[2][1] = 4;
```

```
a[2][2] = 18;
```

```
a[2][3] = 9;
```

		индексы второго измерения j			
		0	1	2	3
индексы первого измерения i	0	45	-7	12	4
	1	10	3	100	-9
	2	-2	4	18	9

Многомерные массивы

```
// поэлементная обработка в цикле -  
// однократный перебор всех элементов  
//
```

```
// ВВОД
```

```
for (size_t i = 0; i < n; i++) {  
    for (size_t j = 0; j < m; j++) {  
        cin >> a[i][j];  
    }  
}
```

```
// ВЫВОД
```

```
for (size_t i = 0; i < n; i++, cout << endl) {  
    for (size_t j = 0; j < m; j++) {  
        cout << setw(4) << a[i][j] << " ";  
    }  
}
```

Пример

В течение всего октября через каждый час измеряли температуру воздуха. В программе ввести эти данные и разместить их в двумерном массиве

Вывести

- а) температуру 1 октября в девять утра и девять вечера
- б) когда было холоднее – в полдень 2 октября или 3 октября
- в) день, когда в 9:00 было теплее всего
- г) количество отрицательных температур 30 октября
- д) самую низкую температуру, день и час когда она была зафиксирована
- е) среднесуточную температуру для каждого дня
- ё) среднемесячную температуру
- ж) массив тех же температур в градусах Фаренгейта

Ссылки

Ссылка (reference) — механизм, позволяющий привязать имя к значению;
в частности, ссылка позволяет дать дополнительное имя переменной

Ссылка – псевдоним некоторой переменной

Если **T** некоторый тип и есть переменная типа **T**,
то переменная типа **T&** будет ссылкой на эту переменную, если она
инициализирована этой переменной:

T имя_переменной = значение;

T& имя_ссылочной_переменной = имя переменной;

```
int a = 5;  
int& p = a; //объявляем ссылку. теперь p это псевдоним a  
cout << a << endl; //выведет 5  
cout << p << endl; //выведет 5
```

```
a = 6; //чтение через ссылку  
cout << a << endl; //выведет 6  
cout << p << endl; //выведет 6
```

```
p = 12345678; //запись через ссылку  
cout << a << endl; //12345678  
cout << p << endl; //12345678
```

```
int a = 5;  
int& d; // ошибка -  
// нельзя оставить  
// ссылку  
// неинициализированной  
  
d = a;  
...
```


Ссылки на константы

```
const double gravity = 9.8;
const double& pg = gravity;
cout << pg << endl; //выведет 9.8
pg = 1.62; // ошибка - запись через константную ссылку недоступна

double& pgrav = gravity; // ошибка - для константы нельзя взять
                        // ссылку на переменную, без const
```

Константные ссылки

```
int a = 5;
const int& pca = a;
cout << pca << endl; //5, доступ на чтение возможен
pca = 1000; // ошибка -
           // запись через константную ссылку недоступна
```

Использование ссылок

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

for (int a : arr)
    a = 2 * a; // удваиваем числа в массиве

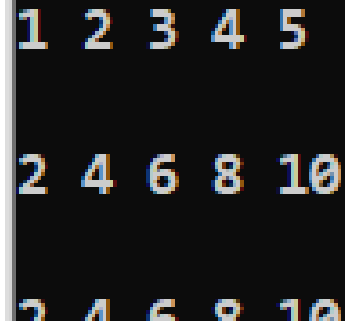
for (int a : arr) // смотрим результат
    cout << a << " "; // ничего не изменилось

cout << "\n\n";
```

```
// теперь a - ссылка на элемент массива
for (int& a : arr)
    a = 2 * a; // удваиваем числа в массиве

for (int a : arr) // результат:
    cout << a << " "; // значения изменились
```

```
// теперь a - const ссылка на элемент массива
for (const int& a : arr)
{
    a = 2 * a; // ошибка - нельзя изменить значение по const-ссылки
    cout << a; // можно + оптимизация (нет копирования в локальную)
}
```



```
1 2 3 4 5
2 4 6 8 10
2 4 6 8 10
```

Операция & - взятие адреса переменной

```
int a = 5, b = -7;
```

```
int& pa = a; //ссылку
```

```
cout << a << endl; //значение a
```

```
cout << b << endl; //значение a
```

```
cout << pa << endl; //значение a  
// через ссылку на нее
```

```
cout << &a << endl; //адрес a
```

```
cout << &b << endl; //адрес b
```

```
cout << &pa << endl; //адрес ссылки на a
```

```
5  
-7  
5  
00000027054FF534  
00000027054FF554  
00000027054FF534
```

Указатели и ссылки

Указатель – тип данных, позволяющих хранить и обрабатывать адреса ячеек памяти

Переменная такого типа, значением которой является адрес, также называется **указателем** (pointer)

Если **T** некоторый тип,
то тип **T** со спецификатором ***** будет типом указателей на тип **T**,
и переменные типа **T *** могут хранить адреса данных типа **T**:

T *имя_указателя = имя_переменной;

T имя_переменной = значение;
имя_указателя = &имя_переменной;

Примеры создания указателей

```
int* pp1; // может хранить адрес целочисленной переменной
double* pp2; // может хранить адрес вещественной переменной
bool* pp3; // может хранить адрес логической переменной
// их значения пока не определены (мусор)
cout << pp1 << " " << pp2 << " " << pp3 << "\n\n"; // в msvc нет
```

```
int* p1 = 0;
int* p2 = NULL;
int* p3 = nullptr;
int* p4{};
```

```
cout << p1 << " " << p2 << " " << p3 << " " << p4 << "\n\n";
```

```
int a = 5, b = -7;
```

```
int *pa = &a, *pb{&b};
```

```
cout << "a = " << a << ", адрес = " << pa << endl;
```

```
cout << "b = " << b << ", адрес = " << pb << endl;
```

```
0x8 0x93e152a33599864a 0x46
```

```
0 0 0 0
```

```
a = 5, адрес = 0x5ffe54
```

```
b = -7, адрес = 0x5ffe50
```

```
PS C:\d\08_2022\cppProjects_2022\
```

Размер переменной указателя

- не зависит от типа указателя.
- зависит от конкретной платформы:
 - на 32-разрядных платформах = 4 байта,
 - на 64-разрядных = 8 байт

```
int* pint{};
```

```
double* pdouble{};
```

```
cout << "*pint size: " << sizeof(pint) << endl;
```

```
cout << "*pdouble size: " << sizeof(pdouble) << endl;
```

```
*pint size: 8
*pdouble size: 8
```

Операции с указателями и ссылками

Присваивание адреса

```
int a = 5, b = -7;  
int *p;  
p = &a;  
//... операции с p (адрес a)  
p = &b;  
//... операции с p (адрес b)
```

Разыменование указателя – операция `*указатель`

```
int a = 5;  
int *p = &a;  
*p = 1000; // записать по адресу новое значение  
int c = *p * 2; // использовать это значение в выражении  
(*p)++; // изменить значение по адресу; != *p++ !!!!!  
cout << "a = " << a << " = " << *p << endl;
```

Операции с указателями

Операции сравнения: >, <, >=, <=, ==, !=

```
int a = 10, b = 20;
int *pa{&a}, *pb{&b};
cout << "pa = " << pa << endl;
cout << "pb = " << pb << endl;
if (pa > pb)
    cout << "pa > pb" << endl;
else
    cout << "pa <= pb" << endl;
```


Адресная арифметика

Операции: +, -, ++, --

```
int a = 5;
```

```
int* pa{ &a };
```

```
pa = pa + 1; // аналогично pa++ или ++pa
```

```
cout << "  a = " << a << ", адрес = " << &a << endl;
```

```
cout << "*pa = " << *pa << ", адрес = " << pa << "\n\n";
```

```
pa = pa - 1; // аналогично pa-- или --pa
```

```
cout << "  a = " << a << ", адрес = " << &a << endl;
```

```
cout << "*pa = " << *pa << ", адрес = " << pa << endl;
```

```
  a = 5,          адрес = 000000838432FCE4
*pa = -858993460, адрес = 000000838432FCE8
```

```
  a = 5, адрес = 000000838432FCE4
*pa = 5, адрес = 000000838432FCE4
```

Адресная арифметика

```
double x = 5.5, y = 1.2;
double* px{ &x }, * py{ &y };
px = px - 1;
py = py - 5;
cout << "  x = " << x << ",\t    адрес = " << &x << endl;
cout << "*px = " << *px << ", адрес = " << px << "\n\n";

cout << "  y = " << y << ",\t    адрес = " << &y << endl;
cout << "*py = " << *py << ", адрес = " << py << "\n\n";

cout << "разница = " << &y - py;
```

```
x = 5.5,          адрес = 00000043555CF5C8
*px = -9.25596e+61, адрес = 00000043555CF5C0

y = 1.2,          адрес = 00000043555CF5E8
*py = -9.25596e+61, адрес = 00000043555CF5C0

разница = 5
```

Указатели и массивы

Имя массива является указателем на его первый по порядку элемент массива.

Обращение к элементу с индексом i :

$*(\text{имя_массива} + i)$

```
double arr[] { 1.2, -2.3, 3.4, -4.5, 5.66666 };
```

```
cout << "первый элемент, индекс == 0" << endl;  
cout << arr[0] << endl;  
cout << *arr << endl;
```

```
cout << "второй элемент, индекс == 1" << endl;  
cout << arr[1] << endl;  
cout << *(arr + 1) << endl;
```

```
cout << "третий элемент, индекс == 2" << endl;  
cout << arr[2] << endl;  
cout << *(arr + 2) << endl;
```

```
первый элемент, индекс == 0  
1.2  
1.2  
второй элемент, индекс == 1  
-2.3  
-2.3  
третий элемент, индекс == 2  
3.4  
3.4
```

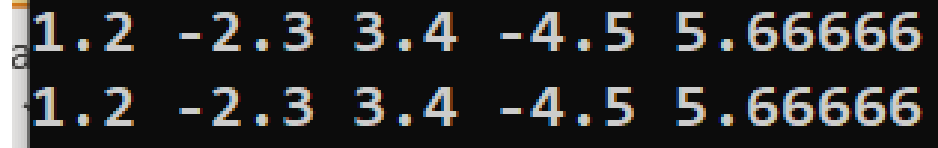
Обход массива через указатели

```
double arr[] { 1.2, -2.3, 3.4, -4.5, 5.66666 };
```

```
int n = std::size(arr);
```

```
for (int i = 0; i < n; i++)  
    cout << *(arr + i) << " ";
```

```
cout << endl;
```



```
1.2 -2.3 3.4 -4.5 5.66666  
1.2 -2.3 3.4 -4.5 5.66666
```

```
for(double* ptr = arr; ptr <= &arr[n - 1]; ptr++)
```

```
    cout << *ptr << " ";
```

Динамические данные в куче (heap)

1. Объявить указатель подходящего типа

```
double* dx;
```

2. Выделить память операцией new

```
dx = new double;
```

3. Использовать указатель для доступа к данным в динамической памяти (куче), выполнить необходимые операции с данными через разыменование указателя

```
*dx = 500;
```

4. Освободить память операцией delete

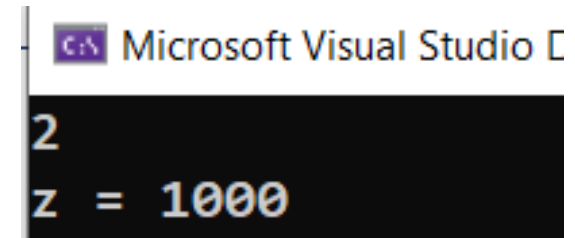
```
delete dx;
```

```
double* dx;  
dx = new double;  
*dx = 500;
```

```
double* dy = new double;  
cin >> *dy;
```

```
double* dz = new double;  
*dz = *dx * *dy;  
cout << "z = " << *dz;
```

```
delete dx;  
delete dy;  
delete dz;
```



Динамические одномерные массивы

1. Объявить указатель подходящего типа

```
double* arr;
```

2. Выделить память операцией new, указать количество элементов в массиве

```
arr = new double[n]; // n – может быть переменной
```

3. Обработать данные как в обычном массиве.

Доступ к элементам через []

```
arr[i]
```

или через указатели

```
*(arr + i)
```

...

4. Освободить память операцией delete[]

```
delete[] arr;
```

Динамические одномерные массивы - пример

```
int n;  
cout << "n = "; cin >> n;  
  
double* arr = new double[n];  
  
for (int i = 0; i < n; i++)  
    cin >> arr[i];  
  
double sum = 0.0;  
for (int i = 0; i < n; i++)  
    sum += arr[i];  
  
cout << "Сумма элементов массива = " << sum;  
  
delete[] arr;
```

```
n = 4  
1.2  
4.5  
2.3  
5.0  
Сумма элементов массива = 13
```

Динамические двумерные массивы

```
int n, m;
cout << "n = "; cin >> n;
cout << "m = "; cin >> m;

// выделить память для указателей на строки
double** arr = new double*[n];

for (int i = 0; i < n; i++)
    arr[i] = new double[m]; // выделить память для каждой строки

for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
        cin >> arr[i][j];

double sum = 0.0;
for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
        sum += arr[i][j];
cout << "Сумма = " << sum;

for (int i = 0; i < n; i++)
    delete[] arr[i]; // освободить память каждой строки

delete[] arr; // освободить набор указателей на строки
```

```
n = 2
m = 3
2 3 -5
1 0 4
Сумма = 5
```