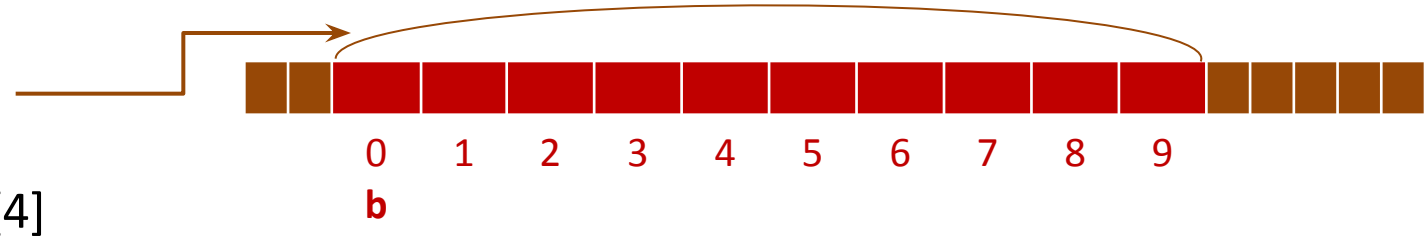
Динамические структуры данных. Массивы и указатели

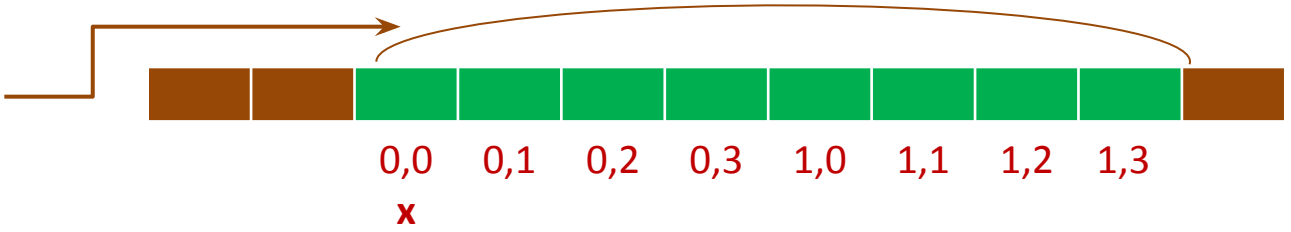
- **Массив – нумерованная последовательность** данных **одного типа**, которые хранятся в непрерывной области памяти друг за другом. Члены последовательности данных называются **элементами массива.**
- Доступ к элементу массива производится путем указания имени массива и номера элемента.
- Нумерация элементов может выполняться одной или несколькими последовательностями целых чисел – индексными последовательностями.
- Если нумерация выполняется одной последовательностью говорят, что массив является **одномерным**, в противном случае – **многомерным**.
- Нумерация элементов массива всегда начинается с **0**, а номер каждого следующего члена больше номера предыдущего на **1**.

Размещение в памяти массивов:

int b [10]



short x [2][4]



Внимание!
Компилятор **гарантирует** размещение элементов массива в смежных ячейках памяти!

Массив определяется:

- именем (идентификатором);
- количеством размерностей - числом номеров, необходимых для указания местонахождения элемента массива;
- размером (диапазоном изменения индексов) по каждой размерности.

Конфигурация массива фиксирована.

- Все элементы массива принадлежат к одному и тому же типу данных.
- Элементами массива могут быть как простые переменные любых типов, так и переменные составных типов (массивов, структур, строк и т.д.).
- В качестве индексов в C++ могут использоваться константы и переменные любых целых типов.

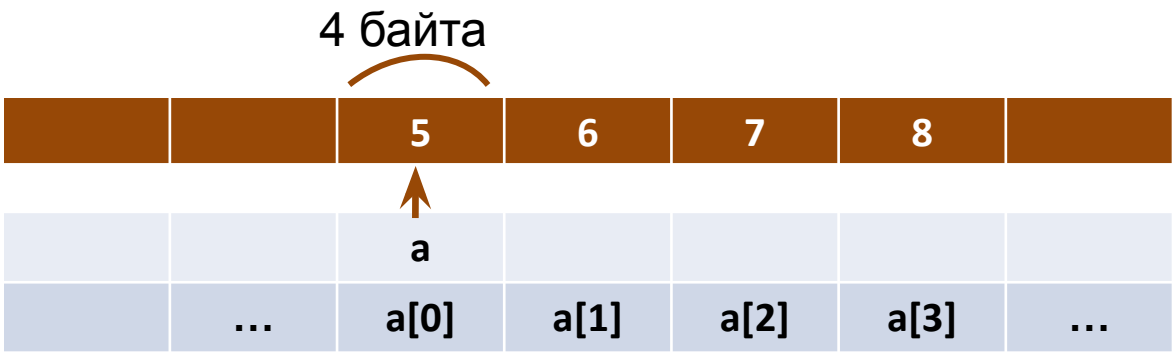
Синтаксис описания одномерного массива

Имя_типа_элемента Имя_массива **[***Размер***];**

Имя_типа_элемента Имя_массива **[***Размер***] = {***список значений***};**

Примеры описаний массивов

```
float  b[20] ;

const int n=4;
int   a[n] = {5, 6,7,8};
```



Примеры описаний массивов

float x[20] = {0} ;

Если список инициализации содержит меньше элементов, чем массив, оставшиеся элементы инициализируются значением 0. Если список содержит больше элементов, чем массив, компилятор генерирует сообщение об ошибке.

double y [ ] = {5.5, 6.8, 8.8, 9.5, 10.3}

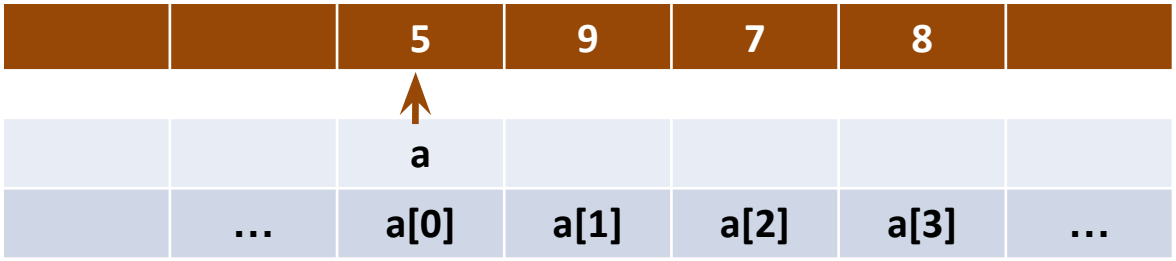
Если значения элементов перечисляются явно, размер массива можно не указывать (компилятор определит его по количеству элементов)

Операция индексирования

Имя_массива [*Индекс*];

Индекс – целочисленное выражение, значение которого изменяется между 0 и (*Размер*-1). Для переменной – индекса лучше выбирать тип целый со знаком, так как контроль выхода значений индекса за допустимый диапазон не ведется.

```
const  int n=4;
int   a[n] = {5, 6,7,8};
int i = 3;
a[1] = a[2]+a[i]-6;
```



Задача.

Написать программу, осуществляющую ввод, сложение, вычитание, скалярное умножение двух n-мерных векторов и вывод результатов.

Структура данных для представления n-мерного вектора: одномерный массив размера n типа double.

Расчетные формулы:

$$\vec{a} = (a_1, a_2, \ldots, a_n); \quad \vec{b} = (b_1, b_2, \ldots, b_n).$$

$$\vec{a} + \vec{b} = (a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n);$$

$$\vec{a} - \vec{b} = (a_1 - b_1, a_2 - b_2, \ldots, a_n - b_n);$$

$$\vec{a}\vec{b} = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n.$$

Решение задачи:

```
int main()
{
    const int n=4;
    double a[n]={0}, b[n]={0}, res[n]={0};

    cout << "Enter..." << endl;
    for (int i=1; i<=n; i++)
      {
          cout << endl << "a" << i << ": ";
          cin >> a[i-1];
          cout << "b" << i << ": ";
          cin >> b[i-1];
      }
```

Если список инициализации содержит меньше значений, чем число элементов массива, оставшиеся элементы будут инициализированы 0

Решение задачи:

```
    for (int i=1; i<=n; i++) res[i-1]=a[i-1]+b[i-1];       //Sum
    cout << endl << "a + b = ( ";
    for (int i=1; i<=n; i++)  cout << res[i-1] << " ";
    cout << ")";

    for (int i=1; i<=n; i++) res[i-1]=a[i-1]-b[i-1];       //Difference
    cout << endl << "a - b = ( ";
    for (int i=1; i<=n; i++)  cout << res[i-1] << " ";
    cout << ")";

    double r=0;
    for (int i=1; i<=n; i++) r+=a[i-1]*b[i-1];             //Product
    cout << endl << "a * b = " << r;

    _getch();
    return 0;
};
```

Возьмем группу студентов из десяти человек.
У каждого из них есть фамилия. Создавать отдельную переменную для каждого студента — не рационально. Создадим массив, в котором будут храниться фамилии всех студентов.

Пример инициализации массива

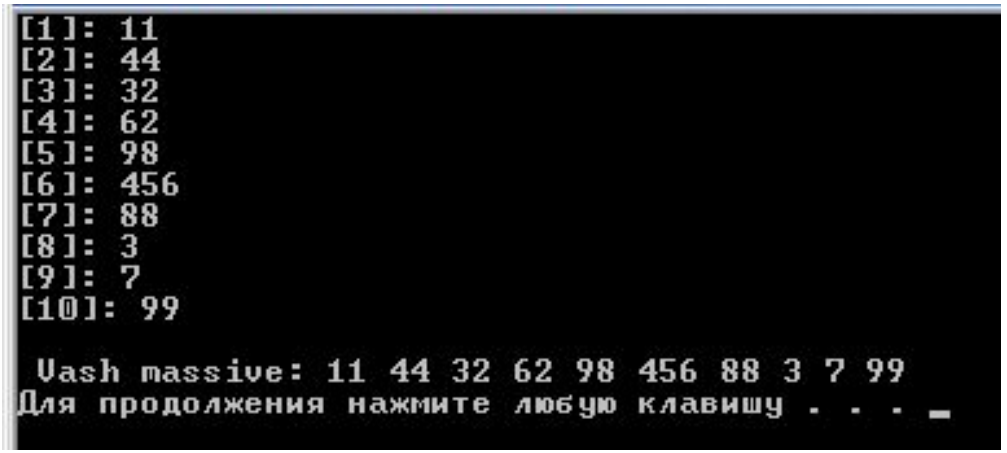
```
string students[10] = {
      "Ivanov", "Petrov", "Sidorov",
      "Ahmetov", "Eremin", "Vasin",
      "Andreev", "Pupkin", "Simakova",
"Lukashin"
   };
```

Описание синтаксиса

Массив создается почти так же, как и обычная переменная.
Для хранения десяти фамилий нам нужен массив, состоящий из **10** элементов. Количество элементов массива задается при его объявлении и заключается в *квадратные скобки*.
Чтобы описать элементы массива сразу при его создании, можно использовать *фигурные скобки*.
В фигурных скобках значения элементов массива перечисляются через *запятую*.
В конце закрывающей фигурной скобки ставится *точка с запятой*.

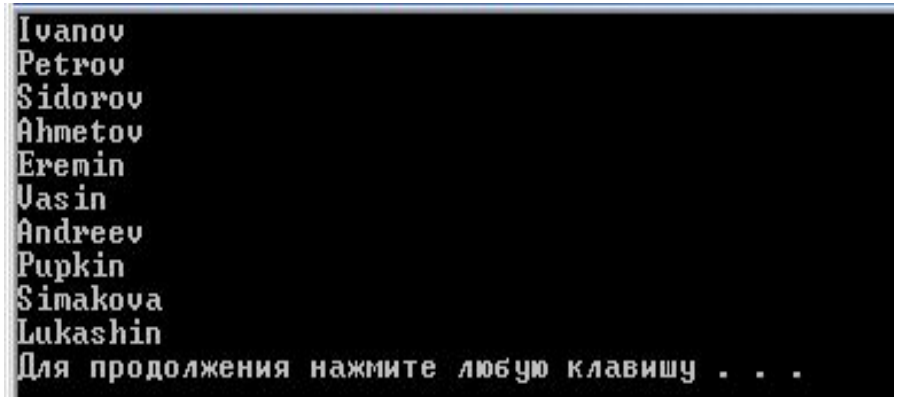
Заполнение массива с клавиатуры

```
#include <iostream>
#include <string>

using std::cout;
using std::cin;
using std::endl;

int main()
{
	int arr[10];
	// Заполняем массив с клавиатуры
	for (int i = 0; i < 10; i++) {
	   cout << "[" << i + 1 << "]" << ": ";
	   cin >> arr[i];
	}

	// И выводим заполненный массив.
	cout << "\n Vash massive: ";

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

	cout << endl;

	return 0;
}
```



Вывод элементов массива через цикл

```
//Вывод элементов массива через цикл
#include <iostream>
#include <string>

int main()
{
    std::string students[10] = {
        "Ivanov", "Petrov", "Sidorov",
        "Ahmetov", "Eremin", "Vasin",
        "Andreev", "Pupkin", "Simakova",
"Lukashin"
    };
    for (int i = 0; i < 10; i++) {
        std::cout << students[i] << std::endl;
    }

    return 0;
}
```



Двумерные массивы

Одномерный массив
int a[6];

Двумерные массивы

Построим массив, состоящий из четырех таких массивов

int a[6];

Двумерные массивы

Построим массив, состоящий из четырех таких массивов

int a[6];

Перенумеруем элементы, используя два индекса и соответственно опишем этот массив:

int a[4][6]

Двумерные массивы

Построим массив, состоящий из четырех таких массивов

int a[6];

Перенумеруем элементы, используя два индекса и соответственно опишем этот массив:

int a[4][6]

В памяти двумерный массив вытянут по строкам

```
int main()  // How to initialize two-dimensional arrays
{
   const int m=4, n =6;          // maximum array dimensions
   int a[m][n] = {{0,1,2,3,4,5},
                  {10,11,12,13,14,15},
                  {20,21,22,23,24,25},
                  {30,31,32,33,34,35}};

int b[m][n] = {0,1,2,3,4,5,10,11,12,13,14,15}; // third & fourth rows are set
to zero

int d[m][n] = {0};   // the whole array is set to zero

int c[][n]={{0,1,2,3,4,5},     // first dimension's size will be calculated by
compiler
              {10,11,12,13,14,15},
              {20,21,22,23,24,25},
              {30,31,32,33,34,35}};
}
```

Многомерные массивы

Трехмерный массив

float x[2][3][4];

```
int main()  // sets elements of 3-D array to sin (i+j*k)
{
   const int l=1, m=2, n=3;
   float x[l+1][m+1][n+1] = {0};

   for (int i=0; i<=l; i++)
      for (int j=0; j<=m; j++)
         for (int k=0; k<=n; k++)
         {
         x[i][j][k] = sin(static_cast <double> (i+j*k));
         cout << setw (10) << x [i][j][k];
         if (k==n) cout << endl;
         if (k==n  & j==m) cout << endl;
         }
_getch(); return 0;
}
```

Динамическое распределение памяти

- Память для глобальных переменных распределяется на этапе компиляции и выделяется при загрузке программы в память.
- Память для локальных переменных выделяется в области стека в момент начала выполнения функции.
- Мы не можем в процессе выполнения программы объявить новые локальные или глобальные переменные, хотя необходимость этого может выясниться только в процессе выполнения.
- Мы не можем работать со структурами данных, размер которых может изменяться в процессе выполнения программы (списки, стеки, очереди, деревья)
- **Механизм динамического распределения памяти позволяет программе получать необходимую для хранения данных память в процессе своего выполнения.**
- Для получения доступа к динамически выделяемым областям памяти и их типизации служат указатели.

Указатели и переменные

Все данные, которые обрабатывает компьютер, хранятся в его памяти. Когда переменной присваивается какое-то значение, оно заполняет собой некоторую часть памяти. Когда значение переменной должно быть использовано, оно считывается из памяти. Таким образом, *каждая переменная — это имя для определенного участка памяти*.

Указатели — это те же обозначения каких-то отдельных фрагментов информации, записанных в памяти. Указатель точно так же, как и переменная, указывает на участок памяти. Поэтому каждый раз, когда вы используете переменную, по сути, вы используете указатель. *Различие между переменными и указателями* заключается в том, что переменные всегда указывают на один и тот же участок памяти, в то время как один и тот же указатель в разные моменты может указывать на разные

Указатели

Что такое указатели

- Каждый байт в памяти ЭВМ имеет адрес по которому можно обратиться к определенному элементу данных
- Указатель – это переменная, которая сохраняет **адрес** другой переменной **определенного типа**.
  - Содержимое указателя содержит **адрес,** начиная с которого размещается переменная, на которую ссылается указатель.
  - По описанию указателя компилятор получает информацию о том, какова длина области памяти, на которую ссылается указатель (которую занимает переменная, на которую он ссылается) и о том, как интерпретировать данные в этой области памяти.

- Таким образом, переменная-указатель обладает именем и имеет тип, определяющий на какого рода данные она может ссылаться.

Информация и ее адрес

Указатели предоставляют доступ к *двум различным фрагментам информации*:

- *Значение, сохраненное самим указателем*. Этим значением всегда будет адрес в памяти, где хранится другая информация. Например, если указатель содержит значение 4, это значит, что он указывает на адрес 4.
- *Значение, на которое указывает указатель*. Например, если в памяти под адресом 4 хранится значение 17, указатель, содержащий значение 4, указывает на значение 17.

Значение, которое содержится в указателе, является просто адресом в памяти компьютера. *Если вывести на экран значение указателя, вы увидите только число, являющееся адресом*. (Поскольку само по себе это число не несет никакой смысловой нагрузки, на экран оно почти никогда не выводится.) Но указатель может также предоставить доступ и к более полезной информации — *к значению, которое хранится под тем адресом, на который он указывает*. Определение значения, на которое ссылается указатель, называется его **разыменованием**. Например, указатель содержит значение 4. Если вы разыменуете его, то получите значение 17, поскольку именно оно хранится в адресе 4.

Например, в повседневной жизни мы сталкиваемся с разыменовыванием. Когда вы набираете номер телефона, автоматически вызов направляется именно к тому абоненту, с которым вы хотите поговорить. Другими словами, номера телефонов в вашей записной книге являются указателями на самых разных людей. И когда вы разыменовываете номер телефона (т.е. просто набираете его), вы получаете доступ к нужному человеку

Как и любая переменная, указатель должен быть ***объявлен***.
При объявлении указателей всегда указывается тип объекта, который будет хранится по данному адресу:
тип имя_переменной;
Например:

int *p *//по адресу, записанному в переменной p,*
//будет хранится переменная типа int
//или, другими словами, p указывает на тип данных int

Звездочка в описании указателя относится непосредственно к имени, поэтому, чтобы объявить несколько указателей, ее ставят перед именем каждого из них:
float *x, y, *z; *//описаны указатели на вещественное число - x и z*
//а также вещественная переменная y

Операции * и & при работе с указателями

Операция получения адреса обозначается знаком &, а операция разадресации *.

Первая возвращает **адрес** своего операнда. Например:

```
float a; //объявлена вещественная переменная a
float *adr_a; //объявлен указатель на тип float
adr_a = &a; //оператор записывает в переменную adr_a
//адрес переменной a
```

Операция *разадресации* * возвращает **значение** переменной, хранящееся в по заданному адресу, то есть выполняет действие, обратное операции &:

```
 float a; //объявлена вещественная переменная a
float *adr_a; //объявлен указатель на тип float
a = *adr_a; //оператор записывает в переменную a
//вещественное значение, хранящиеся по адресу adr_a
```

При работе с указателями можно использовать операции сложения, вычитания и сравнения, причем выполняются они в единицах того типа, на который установлен указатель.

Операции сложения, вычитания и сравнения (больше/меньше) имеют смысл только для последовательно расположенных данных – массивов. Операции сравнения «==» и «!=» имеют смысл для любых указателей, т.е. если два указателя равны между собой, то они указывают на одну и ту же переменную.

Предположим, что значение iNum1 равно 2, а адрес iNum1 — 1000. INum1 будет занимать байты с адресами 1000, 1001, 1002 и 1003. Если значение iNum2 равно 3, то пусть переменная iNum2 занимает ячейки с адресами 1004, 1005, 1006 и 1007. Следовательно, iNum1 начинается с адреса 1000, а iNum2 начинается с адреса 1004.
Однако, хотя iNum1занимает четыре байта, в C/C++ адресом iNum1 называется адрес 1000, а адресом iNum2 называется адрес 1004.
Затем объявим две переменные как указатели — pNum1 и pNum2.
Наша цель - сохранить число 1000 (адрес iNum1) в pNum1 и число 1004 (адрес iNum2) в pNum2.

```
#include <iostream>
using namespace std;
int main()
{
//объявляем три целых переменных:
int iNum1;
int iNum2;
int iResult;
//объявляем две переменные как указатели:
int* pNum1;
int* pNum2;
```

Объявляем две переменные как указатели, используем запись int*. К какому типу относится переменная pNum1? Можно ли сохранить целое значение в pNum1? Нет. В pNum1 можно сохранить **адрес** переменной типа int (т.е. число 1000, поскольку 1000 является адресом iNum1). Так же и для переменной pNum2.

```
iNum1 = 2;
iNum2 = 3;
pNum1 = &iNum1;
pNum2 = &iNum2;
iResult = *pNum1 + *pNum2;
cout << "The result is: ";
cout << iResult << endl;
return 0;
}
```

Эти два оператора сохраняют адрес переменной iNum1 в pNum1 и адрес iNum2 в pNum2

Вопрос.
Каким будет результат выполнения программы?

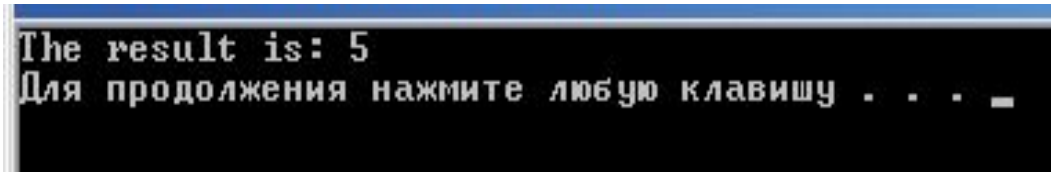
Когда вы используете указатель с символом *, вы извлекаете значение, хранящееся по данному адресу. *pNum1 — это то же, что и *1000, так что программа обращается к значению, хранящемуся по адресу 1000. Поскольку переменная pNum1 была объявлена как int* (а компилятор знает, что целое значение занимает четыре байта памяти), программа обращается к адресам 1000, 1001, 1002 и 1003. Она находит по этим адресам значение 2.

```
iNum1 = 2;
iNum2 = 3;
pNum1 = &iNum1;
pNum2 = &iNum2;
iResult = *pNum1 + *pNum2;
cout << "The result is: ";
cout << iResult << endl;
return 0;
}
```

Эти два оператора сохраняют адрес переменной iNum1 в pNum1 и адрес iNum2 в pNum2

Вопрос.
Каким будет результат выполнения программы?

Когда вы используете указатель с символом *, вы извлекаете значение, хранящееся по данному адресу. *pNum1 — это то же, что и *1000, так что программа обращается к значению, хранящемуся по адресу 1000. Поскольку переменная pNum1 была объявлена как int* (а компилятор знает, что целое значение занимает четыре байта памяти), программа обращается к адресам 1000, 1001, 1002 и 1003. Она находит по этим адресам значение 2.



ССЫЛКИ

Ссылки

- Ссылка (reference) – псевдоним для другой переменной.
- Ссылка имеет имя, которое может использоваться вместо имени переменной. Так как **ссылка – это псевдоним, а не указатель**, переменная, для которой она определяется, должна быть объявлена ранее.
- В отличие от указателя ссылка не может быть изменена, чтобы представлять другую переменную
- Объявление и инициализация:

Базовый_тип &Имя_Ссылки = Имя_Переменной;

```
int  number = 0;
int  &rnumber = number;  // ссылка
int  *pnumber = &number  // указатель
```

- Ссылку, можно использовать вместо имени исходной переменной:

```
rnumber  +=10;
*pnumber +=10;  // требуется разыменование
```

- Ссылка подобна указателю, который уже разыменован и значение которого нельзя изменить.

При объявлении статического массива, ему задается определенный постоянный размер:

```
int n = 10;
int arr[n];
```

Т.е. при объявлении статического массива, его размером должна являться числовая константа, а не переменная.

В большинстве случаев, целесообразно выделять определенное количество памяти для массива, значение которого изначально неизвестно. Например, необходимо создать динамический массив из **N** элементов, где значение **N** задается пользователем.

Мы учились выделять память для переменных, используя указатели. *Выделение памяти для динамического массива имеет аналогичный принцип.*

Динамическим массивом называют *массив* с переменным размером, то есть количество элементов может изменяться во *время выполнения* программы.

При *динамическом распределении памяти* для массивов следует описать соответствующий *указатель*, которому будет присвоено *значение* адреса начала области выделенной памяти.

Указатели и массивы тесно связаны между собой. Идентификатор массива является указателем на его первый элемент, т.е. для массива int ***a***[10], выражения ***a*** и ***a***[0] имеют одинаковые значения, т.к. адрес первого (с индексом 0) элемента массива – это адрес начала размещения его элементов в ОП.

Пусть объявлены – массив из 10 элементов и указатель типа *double*:

double *a*[10], *p;

если *p* = *a*; (установить указатель *p* на начало массива *a*), то следующие обращения: ***a***[i] , *(*a*+i) и *(*p*+i) эквивалентны, т.е. для любых указателей можно использовать две эквивалентные формы доступа к элементам массива: *a*[i] и *(*a*+i). Очевидна эквивалентность следующих выражений:

&*a*[0] ↔ &(**p*) ↔ *p*

Создание динамического массива

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

int main()
{
    int num; // размер массива
    cout << "Enter integer value: ";
    cin >> num; // получение от пользователя размера массива

    int *p_darr = new int[num]; // Выделение памяти для массива
    for (int i = 0; i < num; i++) {
        // Заполнение массива и вывод значений его элементов
        p_darr[i] = i;
        cout << "Value of " << i << " element is " << p_darr[i] << endl;
    }
    delete [] p_darr; // очистка памяти
    return 0;
}
```

Синтаксис выделения памяти для массива имеет вид `указатель = new тип[размер]`.

В качестве размера массива может выступать любое целое положительное значение.

Для создания *двумерного динамического массива* вначале нужно распределить *память* для массива указателей на одномерные массивы, а затем выделить *память* для одномерных массивов.

Объявление двумерных динамических массивов

Под объявлением двумерного *динамического массива* понимают объявление двойного указателя, то есть объявление **указателя на *указатель***.

Синтаксис:

Тип ** ИмяМассива;

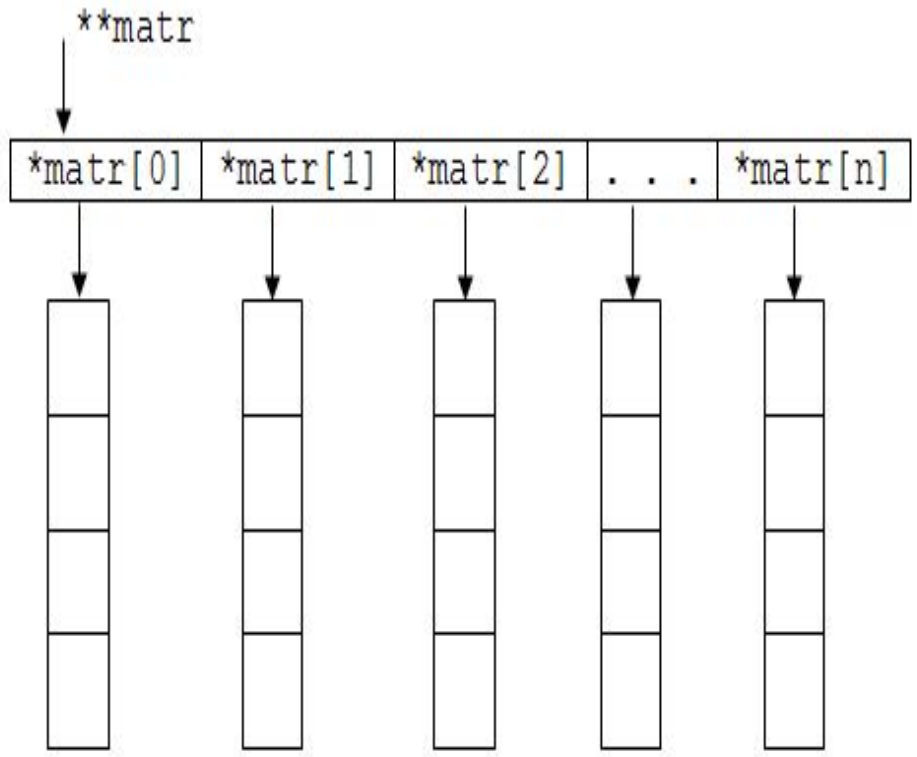
ИмяМассива – *идентификатор* массива, то есть имя двойного указателя для выделяемого *блока памяти*.
Тип – тип элементов объявляемого *динамического массива*.
Элементами *динамического массива* не могут быть функции и элементы типа void.

Например:

```
int **a;
float **m;
```

Выделение памяти под двумерный динамический массив

При формировании двумерного *динамического массива* сначала выделяется *память* для массива указателей на одномерные массивы, а затем в цикле с параметром выделяется *память* под одномерные массивы. На рис. 26.1 представлена схема динамической области памяти, выделенной под *двумерный массив*.



При работе с динамической памятью в языке C++ существует 2 способа выделения памяти под двумерный *динамический массив*.

1) *при помощи операции* `new` , которая позволяет выделить в динамической памяти участок для размещения массива соответствующего типа, но не позволяет его инициализировать.

Синтаксис выделения памяти под *массив* указателей:

```
ИмяМассива = new Тип * [ВыражениеТипаКонстанты];
```

Синтаксис выделения памяти для массива значений:

```
ИмяМассива[ЗначениеИндекса] = new Тип [ВыражениеТипа
Константы];
```

`ИмяМассива` – *идентификатор* массива, то есть имя двойного указателя для выделяемого *блока памяти*.

`Тип` – тип указателя на *массив*.

`ВыражениеТипаКонстанты` – задает количество элементов (*размерность) массива*. *Выражение* константного типа вычисляется на этапе компиляции.

Например:

```
int n, m;//n и m - количество строк и столбцов матрицы
float **matr; //указатель для массива указателей
matr = new float * [n]; //выделение динамической памяти
                        под массив указателей
for (int i=0; i<n; i++)
  matr[i] = new float [m]; //выделение динамической памяти
                           для массива значений
```

При выделении динамической памяти размеры массивов должны быть полностью определены.

2) *при помощи библиотечной функции* `malloc (calloc)` , которая предназначена для выделения динамической памяти.

Синтаксис выделения памяти под *массив* указателей:

```
ИмяМассива = (Тип **) malloc(N*sizeof(Тип *));
```

или

```
ИмяМассива = (Тип **) calloc(N, sizeof(Тип *));
```

Синтаксис выделения памяти для массива значений:

```
ИмяМассива[ЗначениеИндекса]=(Тип*)malloc(M*sizeof(Тип));
```

или

```
ИмяМассива[ЗначениеИндекса]=(Тип*)calloc(M,sizeof(Тип));
```

`ИмяМассива` – *идентификатор* массива, то есть имя двойного указателя для выделяемого *блока памяти*.

`Тип` – тип указателя на *массив*.

`N` – количество строк массива;

`M` – количество столбцов массива.

Например:

```
int n, m;//n и m - количество строк и столбцов матрицы
float **matr; //указатель для массива указателей
matr = (float **) malloc(n*sizeof(float *));
//выделение динамической памяти под массив указателей
for (int i=0; i<n; i++)
  matr[i] = (float *) malloc(m*sizeof(float));
  //выделение динамической памяти для массива значений
```

Так как *функция* `malloc (calloc)` возвращает *нетипизированный указатель* `void *` , то необходимо выполнять его преобразование в *указатель* объявленного типа.

Освобождение памяти, выделенной под двумерный динамический массив

Удаление из динамической памяти двумерного массива осуществляется в порядке, обратном его созданию, то есть сначала освобождается *память*, выделенная под одномерные массивы с данными, а затем *память*, выделенная под *одномерные массив* указателей.

Освобождение памяти, выделенной под двумерный *динамический массив*, также осуществляется 2 способами.

1) *при помощи операции* `delete` , которая освобождает участок памяти ранее выделенной операцией `new` .

Синтаксис освобождения памяти, выделенной для массива значений:

```
delete ИмяМассива [ЗначениеИндекса];
```

Синтаксис освобождения памяти, выделенной под *массив* указателей:

```
delete [] ИмяМассива;
```

`ИмяМассива` – *идентификатор* массива, то есть имя двойного указателя для выделяемого *блока памяти*.

Например:

```
for (int i=0; i<n; i++)
  delete matr [i];
   //освобождает память, выделенную для массива значений
delete [] matr;
//освобождает память, выделенную под массив указателей
```

Квадратные скобки [] означают, что освобождается *память*, занятая всеми элементами массива, а не только первым.

2) *при помощи библиотечной функции* `free` , которая предназначена для освобождения динамической памяти.

Синтаксис освобождения памяти, выделенной для массива значений:

```
free (ИмяМассива[ЗначениеИндекса]);
```

Синтаксис освобождения памяти, выделенной под *массив* указателей:

```
free (ИмяМассива);
```

`ИмяМассива` – *идентификатор* массива, то есть имя двойного указателя для выделяемого *блока памяти*.

Например:

```
for (int i=0; i<n; i++)
  free (matr[i]);
   //освобождает память, выделенную для массива значений
free (matr);
//освобождает память, выделенную под массив указателей
```