# Лабораторная работа №2.
# Динамические массивы в С++

**Цель работы**: изучение и практическое освоение применения динамических массивов в С++.

**Содержание**



## Распределение памяти. Динамическое выделение памяти

### Хранение информации в оперативной памяти

Существует два основных способа хранения информации в оперативной памяти.

Первый способ заключается в использовании глобальных и локальных переменных. В случае глобальных переменных выделяемые под них поля памяти остаются неизменными на все время выполнения программы. Под локальные переменные программа отводит память из стекового пространства. Однако локальные переменные требуют предварительного определения объема памяти, выделяемой для каждой ситуации. Хотя С++ эффективно реализует такие переменные, они требуют от программиста заранее знать, какое количество памяти необходимо для каждой ситуации.

Второй способ, которым С++ может хранить информацию, заключается в использовании системы динамического распределения. При этом способе память распределяется для информации из свободной области памяти по мере необходимости. Область свободной памяти находится между кодом программы с ее постоянной областью памяти и стеком. Динамическое размещение удобно, когда неизвестно, сколько элементов данных будет обрабатываться.

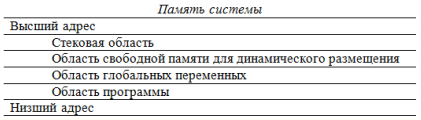


Рисунок 1 - Распределение оперативной памяти для программ на С++

По мере использования программой стековая область увеличивается вниз, то есть программа сама определяет объем стековой памяти. Например, программа с большим числом рекурсивных функций займет больше стековой памяти, чем программа, не имеющая рекурсивных функций, так как локальные переменные и возвращаемые адреса хранятся в стеках. Память под саму программу и глобальные переменные выделяется на все время выполнения программы и является постоянной для конкретной среды.

Память, выделяемая в процессе выполнения программы, называется динамической. После выделения динамической памяти она сохраняется до ее явного освобождения, что может быть выполнено только с помощью специальной операции или библиотечной функции.

Если динамическая память не освобождена до окончания программы, то она освобождается автоматически при завершении программы. Тем не менее, явное освобождение ставшей ненужной памяти является признаком хорошего стиля программирования.

В процессе выполнения программы участок динамической памяти доступен везде, где доступен указатель, адресующий этот участок. Таким образом, возможны следующие три варианта работы с динамической памятью,

выделяемой в некотором блоке (например, в теле неглавной функции).

Указатель (на участок динамической памяти) определен как локальный объект автоматической памяти. В этом случае выделенная память будет недоступна при выходе за пределы блока локализации указателя, и ее нужно освободить перед выходом из блока. Указатель определен как локальный объект статической памяти. Динамическая память, выделенная однократно в блоке, доступна через указатель при каждом повторном входе в блок. Память нужно освободить только по окончании ее использования. Указатель является глобальным объектом по отношению к блоку. Динамическая память доступна во всех блоках, где "виден" указатель. Память нужно освободить только по окончании ее использования.

Все переменные, объявленные в программе размещаются в одной непрерывной области памяти, которую называют сегментом данных. Такие переменные не меняют своего размера в ходе выполнения программы и называются статическими. Размера сегмента данных может быть недостаточно для размещения больших объемов информации. Выходом из этой ситуации является использование динамической памяти. Динамическая память – это память, выделяемая программе для ее работы за вычетом сегмента данных, стека, в котором размещаются локальные переменные подпрограмм и собственно тела программы.

Для работы с динамической памятью используют указатели. С их помощью осуществляется доступ к участкам динамической памяти, которые называются динамическими переменными. Для хранения динамических переменных выделяется специальная область памяти, называемая "кучей".

Динамические переменные создаются с помощью специальных функций и операций. Они существуют либо до конца работы программы, либо до тех пор, пока не будет освобождена выделенная под них память с помощью специальных функций или операций. То есть время жизни динамических переменных – от точки создания до конца программы или до явного освобождения памяти.

В C++ используется два способа работы с динамической памятью:

1. Использование операций ***new*** и ***delete*** ;
2. Использование семейства функций ***malloc*** ( ***calloc*** ) (унаследовано из C).

---

## Работа с динамической памятью с помощью операций ***new*** и ***delete***

---

В языке программирования C++ для динамического распределения памяти существуют операции new и delete. Эти операции используются для выделения и освобождения блоков памяти. Область памяти, в которой размещаются эти блоки, называется свободной памятью.

Операция new позволяет выделить и сделать доступным свободный участок в основной памяти, размеры которого соответствуют типу данных, определяемому именем типа.

Синтаксис:

```
new ИмяТипа;
```

или

```
new ИмяТипа [Инициализатор];
```

В выделенный участок заносится значение, определяемое инициализатором, который не является обязательным элементом. В случае успешного выполнения new возвращает адрес начала выделенного участка памяти. Если участок нужных размеров не может быть выделен (нет памяти), то операция new возвращает нулевое значение адреса (***NULL***).

Синтаксис применения операции:

```
Указатель = new ИмяТипа [Инициализатор];
```

Операция ***new float*** выделяет участок памяти размером 4 байта. Операция ***new int*(15)** выделяет участок памяти 4 байта и инициализирует этот участок целым значением 15. Синтаксис использования операций ***new*** и ***delete*** предполагает применение указателей. Предварительно каждый указатель должен быть объявлен:

```
тип *ИмяУказателя;
```

Например:

```
float *pi;       //Объявление переменной pi
pi=new float;    //Выделение памяти для переменной pi
* pi = 2.25;     //Присваивание значения
```

В качестве типа можно использовать, например, стандартные типы ***int***, ***long***, ***float***, ***double***, ***char***.

Оператор ***new*** чаще всего используется для размещения в памяти данных определенных пользователем типов, например, структур:

```
struct Node {
        char *Name;
        int Value;
        Node *Next
};

Node *PNode;           //объявляется указатель
PNode = new Node;      //выделяется память
PNode->Name = "Ata";   //присваиваются значения
PNode->Value = 1;
PNode->Next = NULL;
```

В качестве имени типа в операции ***new*** может быть использован массив:

```
new ТипМассива
```

При выделении динамической памяти для массива его размеры должны быть полностью определены. Например:

```
ptr = new int [10];//10 элементов типа int или 40 байт
ptr = new int [ ];//неверно, т.к. не определен размер
```

Такая операция позволяет выделить в динамической памяти участок для размещения массива соответствующего типа, но не позволяет его инициализировать. В результате выполнения операция ***new*** возвратит указатель, значением которого служит адрес первого элемента массива. Например:

```
int *n = new int;
```

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.

```
int *b = new int (10);
```

В данном операторе, кроме описанных выше действий, производится инициализация выделенной динамической памяти значением 10.

```
int *q = new int [10];
```

В этом случае операция ***new*** выполняет выделение памяти под 10 величин типа ***int*** (массива из 10 элементов) и записывает адрес начала этого участка в переменную ***q***, которая может трактоваться как имя массива. Через имя можно обращаться к любому элементу массива.

Есть ряд преимуществ использования ***new***. Во-первых, операция ***new*** автоматически вычисляет размер необходимой памяти. Нет необходимости в использовании операции ***sizeof***(). Более важно то, что она предотвращает случайное выделение неправильного количества памяти. Во-вторых, операция ***new*** автоматически возвращает указатель требуемого типа.

Для освобождения выделенного операцией new участка памяти используется операции:

```
delete указатель;
```

Указатель адресует освобождаемый участок памяти, ранее выделенный с помощью операции ***new***. Например:

```
delete x;
```

Для освобождения памяти, выделенной для массива, используется следующая модификация той же операции:

```
delete [ ] указатель;
```

Операцию ***delete*** следует использовать только для указателей на память, выделенную с помощью операции ***new***. Использование ***delete*** с другими типами адресов может породить серьезные проблемы.

---

Пример 1. Демонстрация выполнения операций с динамической памятью

---

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int _tmain(int argc, _TCHAR* argv[]){
  int *pa, *pb;
  pa = new int;
  *pa = 21;
  pb = pa;
  cout << *pa << "  " << *pb << "\n";
  pb = new int;
  *pb = 28;
  cout << *pa << "  " << *pb << "\n";
  delete pa;
  pa = pb;
  cout << *pa << "  " << *pb << "\n";
  delete pa;
  system("pause");
  return 0;
}
```

Результат выполнения программы:

```
21  21
21  28
28  28
```

---

Работа с динамической памятью с помощью библиотечных функций ***malloc*** (***calloc***) и ***free***

Средства для динамического выделения и освобождения памяти описаны в заголовочных файлах ***malloc.h*** и ***stdlib.h*** стандартной библиотеки (файл ***malloc.h*** ).

Функции выделения и освобождения памяти

| Функция | Прототип | Краткое описание |
|---|---|---|
| malloc | void * malloc (unsigned s); | возвращает указатель на начало области (блока) динамической памяти длинной в s байт. При неудачном завершении возвращает значение NULL |
| calloc | void * calloc (unsigned n, unsigned m); | возвращает указатель на начало области (блока) обнуленной динамической памяти, выделенной для размещения n элементов по m байт каждый. При неудачном завершении возвращает значение NULL. |
| realloc | void * realloc (void * bl, unsigned ns); | изменяет размер блока ранее выделенной динамической памяти до размера ns байт, bl – адрес начала изменяемого блока. Если bl равен NULL (память не выделялась), то функция выполняется как malloc |
| free | void * free (void * bl); | освобождает ранее выделенный участок (блок) динамической памяти, адрес первого байта которого равен значению bl. |

Функции ***malloc***(), ***calloc***() и ***realloc***() динамически выделяют память в соответствии со значениями параметров и возвращают адрес начала выделенного участка памяти. Для универсальности тип возвращаемого значения каждой из этих функций есть ***void*** *. Этот указатель можно преобразовать к указателю любого типа с помощью операции явного приведения типа ( тип * ).

Функция ***free***() освобождает память, выделенную перед этим с помощью одной из трех функций ***malloc***(), ***calloc***() или ***realloc***(). Сведения об участке памяти передаются в функцию ***free***() с помощью указателя – параметра типа ***void ****. Преобразование указателя любого типа к типу void * выполняется автоматически, поэтому вместо формального параметра ***void *bl*** можно подставить в качестве фактического параметра указатель любого типа без операции явного приведения типов.

Пример 2. Ввести и напечатать в обратном порядке набор вещественных чисел, количество которых заранее не фиксировано, а вводится до начала ввода самих числовых значений.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int _tmain(int argc, _TCHAR* argv[]){
  float* t; //Указатель для выделяемого блока памяти
  int i,n;
  printf("n=");//n - число элементов
  scanf("%d", &n);
  t=(float *)malloc(n*sizeof(float));
  for (i=0; i=0; i--){
    printf("\nx[%d]=%f",i,t[i]);
  }
  free(t); //освобождает память
  system("pause");
  return 0;
}
```

В программе int n – количество вводимых чисел типа ***float***, ***float* t*** – указатель на начало области, выделяемой для размещения n вводимых чисел. Указатель ***t*** принимает значение адреса области, выделяемой для ***n*** значений типа ***float***. Доступ к участкам памяти выделенной области выполняется с помощью операции индексирования: ***t***[***i***] и ***t***[***i***-1]. Оператор ***free***(***t***); содержит вызов функции, освобождающей выделяемую ранее динамическую память и связанной с указателем ***t***.

**Одномерные динамические массивы**

При использовании многих структур данных достаточно часто бывает, что они должны иметь переменный размер во время выполнения программы. В этих случаях необходимо применять динамическое выделение памяти. Одной из самых распространенных таких структур данных являются массивы, в которых изначально размер не определен и не зафиксирован.

В соответствии со стандартом языка массив представляет собой совокупность элементов, каждый из которых имеет одни и те же атрибуты. Все эти элементы размещаются в смежных участках памяти подряд, начиная с адреса, соответствующего началу массива. То есть общее количество элементов массива и размер памяти, выделяемой для него, получаются полностью и однозначно заданными определением массива. Но это не всегда удобно. Иногда требуется, чтобы выделяемая память для массива имела размеры для решения конкретной задачи, причем ее объем заранее не известен и не может быть фиксирован. Формирование массивов с переменными размерами (динамических массивов) можно организовать с помощью указателей и средств динамического распределения памяти.

Динамический массив – это массив, размер которого заранее не фиксирован и может меняться во время исполнения программы. Для изменения размера динамического массива язык программирования C++, поддерживающий такие массивы, предоставляет специальные встроенные функции или операции. Динамические массивы дают возможность более гибкой работы с данными, так как позволяют не прогнозировать хранимые объемы данных, а регулировать размер массива в соответствии с реально необходимыми объемами.

Объявление одномерного динамического массива

Под объявлением одномерного динамического массива понимают объявление указателя на переменную заданного типа для того, чтобы данную переменную можно было использовать как динамический массив.

Синтаксис:

```
Тип * ИмяМассива;
```

***ИмяМассива*** – идентификатор массива, то есть имя 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;
double *d;
```

В данных примерах a и d являются указателями на начало выделяемого участка памяти. Указатели принимают значение адреса выделяемой области памяти для значений типа int и типа double соответственно.

Таким образом, при динамическом распределении памяти для динамических массивов следует описать соответствующий указатель, которому будет присвоено значение адреса начала области выделенной памяти.

---

## Выделение памяти под одномерный динамический массив

---

Для того чтобы выделить память под одномерный динамический массив в языке C++ существует 2 способа.

1) при помощи операции new, которая выделяет для размещения массива участок динамической памяти соответствующего размера и не позволяет инициализировать элементы массива.

Синтаксис:

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

***ИмяМассива*** – идентификатор массива, то есть имя 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 *mas;
mas = new int [100]; /*выделение динамической памяти
                       размером 100*sizeof(int) байтов*/
double *m = new double [n]; /*выделение динамической
                 памяти размером n*sizeof(double) байтов*/
long (*lm)[4];
lm = new long [2] [4]; /*выделение динамической памяти
                     размером 2*4*sizeof(long) байтов*/
```

При выделении динамической памяти размеры массива должны быть полностью определены.

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

Синтаксис:

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

или

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

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

***Тип*** – тип указателя на массив.

N – количество элементов массива.

Например:

```
float *a;
a=(float *)malloc(10*sizeof(float));
// или
a=(float *)calloc(10,sizeof(float));
/*выделение динамической памяти размером 10*sizeof(float) байтов*/
```

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

---

## Освобождение памяти, выделенной под одномерный динамический массив

---

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

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

Синтаксис:

```
delete [] ИмяМассива;
ИмяМассива – идентификатор массива, то есть имя указателя для выделяемого блока памяти.
```

Например:

```
delete [] mas; /*освобождает память, выделенную под
                 массив, если mas адресует его начало*/
delete [] m;
delete [] lm;
Квадратные скобки [] сообщают оператору, что требуется освободить память, занятую всеми элементами, а не только пер
```

2) при помощи библиотечной функции free, которая служит для освобождения динамической памяти.
Синтаксис:

```
free (ИмяМассива);
```

***ИмяМассива*** – идентификатор массива, то есть имя указателя для выделяемого блока памяти.
Например:

```
free (a); //освобождение динамической памяти
```

---

Обращение к элементам одномерного динамического массива

---

Адресация элементов динамического массива осуществляется аналогично адресации элементов статического массива, то есть с помощью индексированного имени.
Синтаксис:

```
ИмяМассива[ВыражениеТипаКонстанты];
```

или

```
ИмяМассива[ЗначениеИндекса];
```

Например:

```
mas[5] – индекс задается как константа,
sl[i] – индекс задается как переменная,
array[4*p] – индекс задается как выражение.
```

---

Пример 3. Сформировать динамический одномерный массив, заполнить его случайными числами. Преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие в чётных позициях, а во второй половине – элементы, стоявшие в нечётных позициях.

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

int _tmain(int argc, _TCHAR* argv[]){
  int *a, n, i;
  cout << "Введите n: ";
  cin >> n;
  cout << ' ';
  a = new int [n];  //Выделение памяти под массив
  for (i=0; i<n; i++) {
    cout << "Введите a[" << i << "]: ";
    cin >> a[i];
    cout << ' ';
  }

  int *buf = new int [n];
  //Выделение памяти под вспомогательный массив
  int j = 0; //Индекс вспомогательного массива
  for (i=0; i<n; i+=2) {
  //Переписываем элементы с чётным индексом в новый массив
    buf[j] = a[i];
    j++;
  }
  for (i=1; i<n; i+=2) {
  //Переписываем элементы с нечётным индексом в новый массив
    buf[j] = a[i];
    j++;
  }
  cout << "Преобразованный: " << ' ';
  for (i=0; i<n; i++)
    cout << buf[i] << ' ';
  delete [] a;  //Освобождаем память
  delete [] buf;
  system("pause");
  return 0;
}
```

Отметим, что указатель на массив не обязательно должен показывать на начальный элемент некоторого массива. Он может быть сдвинут так, что начальный элемент будет иметь индекс, отличный от нуля, причем он может быть как положительным, так и отрицательным.

---

Пример 4

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int _tmain(int argc, _TCHAR* argv[]){
  float *mas;
  int m;
  scanf("%d",&m);
  mas=(float *)calloc(m,sizeof(float));
  //сейчас указатель q показывает на начало массива
  mas[0]=22.3;
  mas-=5;
  /*теперь начальный элемент массива имеет индекс 5,
    а конечный элемент индекс n-5*/
  mas[5]=1.5;
  /*сдвиг индекса не приводит к перераспределению массива
    в памяти и изменится начальный элемент*/
  mas[6]=2.5; // это второй элемент
  mas[7]=3.5; // это третий элемент
  mas+=5;
  /*теперь начальный элемент вновь имеет индекс 0,
    а значения элементов q[0], q[1], q[2] равны
    соответственно 1.5, 2.5, 3.5*/
  mas+=2;
  /*теперь начальный элемент имеет индекс -2, следующий -1,
    затем 0 и т.д. по порядку*/
  mas[-2]=8.2;
  mas[-1]=4.5;
  mas-=2;
  /*возвращаем начальную индексацию, три первых элемента
    массива q[0],q[1],q[2], имеют значения 8.2, 4.5, 3.5*/
  mas--;
  /*вновь изменим индексацию. Для освобождения области
    памяти, в которой размещен массив q используется
    функция free(q), но поскольку значение указателя q
    смещено, то выполнение функции free(q) приведет к
    непредсказуемым последствиям.
    Для правильного выполнения этой функции указатель q
    должен быть возвращен в первоначальное положение */
  free(++mas);
  system("pause");
  return 0;
}
```

---

Пример 5. Задача Иосифа Флавия или считалка Джозефуса.

Задача в своей основе имеет легенду. Отряд из 41-го сикария, защищавший галилейскую крепость Массада, не пожелал сдаваться в плен блокировавшим его превосходящим силам римлян. Сикарии стали в круг и договорились, что каждые два воина будут убивать третьего, пока не погибнут все. Самоубийство – тяжкий грех, но тот, кто в конце концов останется последним, должен будет его совершить. Иосиф Флавий, командовавший этим отрядом, якобы быстро рассчитал, где нужно стать ему и его другу, чтобы остаться последними. Но не для того, чтобы убить друг друга, а чтобы сдать крепость римлянам. В современной формулировке задачи участвует n воинов и убивают каждого k -го. Требуется определить номера m и t начальных позиций двоих воинов, которые должны будут остаться последними.

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

void kill(int *mass,int n,int i);
void krug(int *mass,int n,int k, int i=0);

int _tmain(int argc, _TCHAR* argv[]){
  int n,k,*mass,i;
  FILE *f;
  f=fopen("input.txt","r");
  fscanf(f,"%d %d",&n,&k);
  fclose(f);
  mass=(int *)malloc(n*sizeof(int));
  for (i=0;i<n;i++) mass[i]=i+1;
  f=fopen("output.txt","w");
  fprintf(f,"Исходная нумерация: \n");
  for (i=0;i<n;i++) fprintf(f,"%d ",mass[i]);
  fclose(f);
```

```
  krug(mass,n,k);
  f=fopen("output.txt","a+");
  fprintf(f,"\nОставшиеся в живых: \n");
  for (i=0;i<k;i++) fprintf(f,"%d ",mass[i]);
  fclose(f);
  free(mass);
  system("pause");
  return 0;
}

void kill(int *mass,int n,int i) {
  int j;
  for (j=i;j<n-1;j++)
    mass[j]=mass[j+1];
}

void krug(int *mass,int n,int k,int i) {
  int ii;
  if (n>k) {
    ii=i+k-1;
    if (ii>=n) ii=ii%n;
    kill(mass,n,ii);
    krug(mass,n-1,k,ii);
  }
}
```

## Двумерные динамические массивы

### 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. 2 представлена схема динамической области памяти, выделенной под двумерный массив.

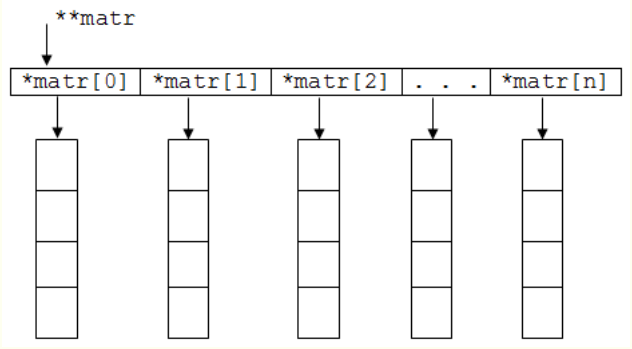


Рисунок 2 - Выделение памяти под двумерный динамический массив

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
```

---

Обращение к элементам двумерного динамического массива

---

Адресация элементов динамического массива осуществляется с помощью индексированного имени.
Синтаксис:

```
ИмяМассива[ВыражениеТипаКонстанты][ВыражениеТипаКонстанты];
```

или

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

Например:

```
mas[5][7] – индекс задается как константа,
sl[i][j] – индекс задается как переменная,
array[4*p][p+5] – индекс задается как выражение.
```

Пример 6. Сформируйте и выведите на экран единичную матрицу с целыми элементами, вводя ее порядок с клавиатуры.

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

int _tmain(int argc, _TCHAR* argv[]){
  int n,i,j;
  int **matr;//указатель для массива указателей
  cout << "Input matrix order:";
  cin >> n;
  matr = new int *[n];
  //выделение памяти под массив указателей
  for(i=0; i<n; i++){
    matr[i] = new int[n];
    //выделение памяти для массива значений
    for (j=0; j<n; j++) //заполнение матрицы
      matr[i][j] = (i==j ? 1 : 0);
  }
  cout << "Result: ";
  for(i=0; i<n; i++){
    cout << "\n";
    for (j=0; j<n; j++)
      cout << " " << matr[i][j];
    delete matr[i];
    //освобождение памяти из-под массива значений
  }
  delete [] matr;
  //освобождение памяти из-под массива указателей
  system("pause");
  return 0;
}
```

Пример 7. Вычислить сумму элементов, лежащих на диагоналях матрицы N x N (обратить внимание на четность-нечетность числа N ). Размер массива должен задаваться пользователем с клавиатуры.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
#include <time.h>
void gen (int nn,int a, int b,int ***mas);
//объявление функции генерации массива
int summa(int nn, int **mas);
/*объявление функции вычисления суммы заданных элементов массива*/
void out (int nn,int **mas);
//объявление функции вывода массива
```

```
int _tmain(int argc, _TCHAR* argv[]){
  int **mass, n;
  int s;
  printf("Введите n: ");
  scanf("%d",&n);
  printf("\nГенерация массива \n");
  gen(n,0,10,&mass);
  s=summa(n,mass);
  out(n,mass);
  printf("\nСумма элементов = %d",s);
  system("pause");
  return 0;
}

void gen(int nn, int a, int b, int ***mas){
  //функция генерации массива
  int i,j;
  srand(time(NULL)*1000);
  *mas=(int**)malloc(nn*sizeof(int*));
  for (i=0;i<nn;i++){
    (*mas)[i]=(int*)malloc(nn*sizeof(int));
    for (j=0;j<nn;j++)
      (*mas)[i][j]=rand()%(b-a)+a;
  }
}

int summa(int nn, int **mas) {
  //функция вычисления суммы элементов диагоналей
  int i,j, sum=0;
  for (i=0;i<nn;i++)
    for (j=0;j<nn;j++) {
      if ((i==j) || (i==nn-j-1)) {
        //нахождение элементов диагоналей
        sum+=mas[i][j];
        //суммирование элементов диагоналей
      }
    }
  return sum;
}

void out (int nn,int **mas){
  //функция вывода массива
  int i,j;
  for (i=0;i<nn;i++) {
    for (j=0;j<nn;j++)
      printf("%4d",mas[i][j]);
    printf("\n");
    free (mas[i]);
    }
  free (mas);
}
```

В языке C++ предусмотрено использование указателя вида ***mass. В данном примере в функцию генерации массива передается не адрес указателя, а его значение. Передача фактического параметра при вызове функции осуществляется через определение адреса указателя **mass.

## 2.1 Задание 1-3

Пересмотреть реализованные в Лабораторной работе №1 классы. Предусмотреть динамические структуры данных. Например:

| № варианта | Задание |
|---|---|
| 1 | Учитывать, что в месяце может быть 28,29,30 и 31 день. |
| 2 | Учитывать, что количество сдаваемых в сессию дисциплин может меняться. |
| 3 | Учитывать, что в месяце может быть 28,29,30 и 31 день. |
| 4 | Учитывать, что количество рейсов в сутки может меняться. |
| 5 | Учитывать, что количество рейсов в разные дни может быть различным. |
| 6 | Учитывать, что в месяце может быть 28,29,30 и 31 день. |
| 7 | Учитывать, что в месяце может быть 28,29,30 и 31 день. |
| 8 | Учитывать, количество рейсов зависит от дня недели. |
| 9 | Количество принимающих в голосовании людей различно за каждого кандидата . |
| 10 | Количество студентов в группе может быть различным. |
| 11 | Учитывать, что в месяце может быть 28,29,30 и 31 день. |
| 12 | Количество команд может быть различным. |