Лабораторная работа №6
Работа с динамическими массивами и функциями

1. Цель работы:

1) Получение практических навыков при работе с массивами.
2) Получение практических навыков при работе с указателями.
3) Получить практические навыки выделения, перераспределения и освобождение памяти при работе с динамическими массивами

2. Краткие теоретические сведения

2.1. Понятие указателя

Указатели являются специальными объектами в программах на C/C++. Указатели предназначены для хранения адресов памяти.

Когда компилятор обрабатывает оператор определения переменной, например, `int i=10;`, то в памяти выделяется участок памяти в соответствии с типом переменной (для `int` размер участка памяти составит 4 байта) и записывает в этот участок указанное значение. Все обращения к этой переменной компилятор заменит адресом области памяти, в которой хранится эта переменная.

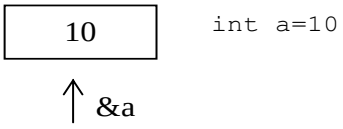


Рис. 3. Значение переменной и ее адрес

Программист может определить собственные переменные для хранения адресов областей памяти. Такие переменные называются указателями. Указатель не является самостоятельным типом, он всегда связан с каким-то другим типом.

В простейшем случае объявление указателя имеет вид:

```
тип* имя;
```

Знак `*`, обозначает указатель и относится к типу переменной, поэтому его рекомендуется ставить рядом с типом, а от имени переменной отделять пробелом, за исключением тех случаев, когда описываются несколько указателей. При описании нескольких указателей знак `*` ставится перед именем переменной-указателя, т. к. иначе будет не понятно, что эта переменная также является указателем.

```
int* i;
double *f, *ff;//два указателя
char* c;
```

Размер указателя зависит от модели памяти. Можно определить указатель на указатель: `int** a;`

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
```

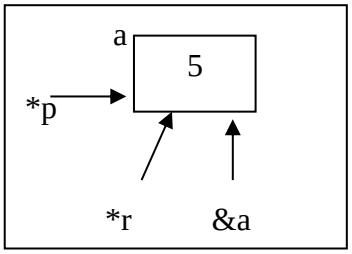
```
const int* pci;     //указатель на целую константу
```

Указатель можно сразу проинициализировать:

```
//указатель на целую переменную
int* pi=&i;
```

Для инициализации указателя существуют следующие способы:

Присваивание адреса существующего объекта:



1) с помощью операции получения адреса
int a=5;
int *p=&a; или int p(&a);
2) с помощью проинициализированного указателя
int *r=p;
3) адрес присваивается в явном виде
char*cp=(char*)0x B800 0000;
где 0x B800 0000 – шестнадцатеричная константа, (char*) – операция приведения типа.
4) присваивание пустого значения:
int*N=NULL;
int *R=0;

С указателями можно выполнять следующие операции:

- разыменование (*);
- присваивание;
- арифметические операции (сложение с константой, вычитание, инкремент ++, декремент --);
- сравнение;
- приведение типов.

Операция разыменования предназначена для получения значения переменной или константы, адрес которой хранится в указателе. Если указатель указывает на переменную, то это значение можно изменять, также используя операцию разыменования.

```
int a; //переменная типа int
int* pa=new int; //указатель и выделение памяти под
//динамическую переменную
*pa=10;//присвоили значение динамической
//переменной, на которую указывает указатель
a=*pa;//присвоили значение переменной a
```

Арифметические операции применимы только к указателям одного типа.

- Инкремент увеличивает значение указателя на величину sizeof(тип).

```
char* pc;
int* pi;
double* pd;
. . . . .
pc++;       //значение увеличится на 1
pi++;       //значение увеличится на 4
pd++;       //значение увеличится на 8
```

- Декремент уменьшает значение указателя на величину sizeof(тип).

- Разность двух указателей – это разность их значений, деленная на размер типа в байтах.
  Суммирование двух указателей не допускается.
- Можно суммировать указатель и константу:

2.2. Динамические переменные

Все переменные, объявленные в программе размещаются в одной непрерывной области памяти, которую называют сегментом данных (64К). Такие переменные не меняют своего размера в ходе выполнения программы и называются статическими. Размера сегмента данных может быть недостаточно для размещения больших массивов 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. Динамические переменные создаются с помощью специальных функций и операций. Они существуют либо до конца работы программ, либо до тех пор, пока не будут уничтожены с помощью специальных функций или операций.

Для создания динамических переменных используют операцию new, определенную в СИ++:

указатель = new имя_типа[инициализатор];

где инициализатор – выражение в круглых скобках.

Операция new позволяет выделить и сделать доступным участок динамической памяти, который соответствует заданному типу данных. Если задан инициализатор, то в этот участок будет занесено значение, указанное в инициализаторе.

int*x=new int(5);

Для удаления динамических переменных используется операция delete, определенная в СИ++:

delete указатель;

где указатель содержит адрес участка памяти, ранее выделенный с помощью операции new.

delete x;

В языке Си определены библиотечные функции для работы с динамической памятью, они находятся в библиотеке <stdlib.h>:

1) void*malloc(unsigned s) – возвращает указатель на начало области динамической памяти длиной s байт, при неудачном завершении возвращает NULL;
2) void*calloc(unsigned n, unsigned m) – возвращает указатель на начало области динамической для размещения n элементов длиной m байт каждый, при неудачном завершении возвращает NULL;
3) void*realloc(void *p,unsigned s) –изменяет размер блока ранее выделенной динамической до размера s байт, p – адрес начала изменяемого блока, при неудачном завершении возвращает NULL;
4) void *free(void *p) – освобождает ранее выделенный участок динамической памяти, p-адрес начала участка.

Пример:

```
int *u=(int*)malloc(sizeof(int)); // в функцию передается количество требуемой памяти в байтах, т. к. функция возвращает значение типа void*, то его необходимо преобразовать к типу указателя (int*).
free(u); //освобождение выделенной памяти
```

2.3. Массивы и указатели

При определении массива ему выделяется память. После этого имя массива воспринимается как константный указатель того типа, к которому относятся элементы массива. Исключением является использование операции `sizeof(имя_массива)` и операции `&имя_массива`.

```
int a[100];

/*определение количества занимаемой массивом памяти, в нашем
случае это 4*100=400 байт*/
int k=sizeof(a);

/*вычисление количества элементов массива*/
int n=sizeof(a)/sizeof(a[0]);
```

Результатом операции `&` является адрес нулевого элемента массива:

```
имя_массива==&имя_массива=&имя_массива[0]
```

Имя массива является указателем-константой, значением которой служит адрес первого элемента массива, следовательно, к нему применимы все правила адресной арифметики, связанной с указателями. Запись `имя_массива[индекс]` это выражение с двумя операндами: имя массива и индекс. Имя_массива – это указатель-константа, а индекс определяет смещение от начала массива. Используя указатели, обращение по индексу можно записать следующим образом: `*(имя_массива+индекс)`.

```
for(int i=0;i<n;i++)     //печать массива
     cout<<*(a+i)<<" ";
     /*к имени (адресу) массива добавляется константа i и
полученное значение разыменовывается*/
```

Так как имя массива является константным указателем, то его невозможно изменить, следовательно, запись `*(a++)` будет ошибочной, а `*(a+1)` – нет.

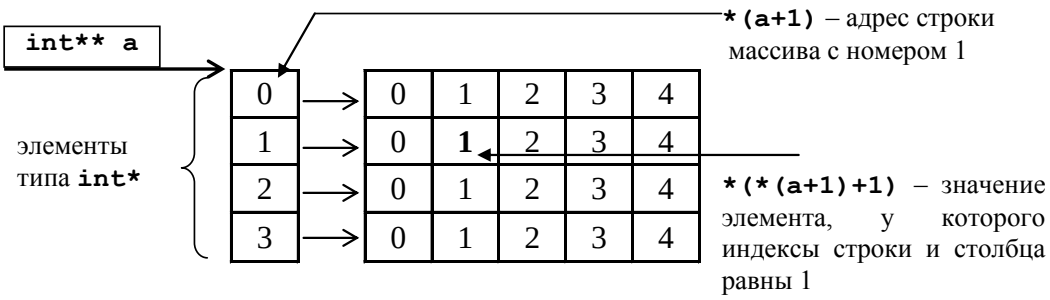
Указатели можно использовать и при определении массивов:

```
int  a[100]={1,2,3,4,5,6,7,8,9,10};

//поставили указатель на уже определенный массив
int* na=a;

/*выделили в динамической памяти место под массив из 100
элементов*/
int b = new int[100];
```

Многомерный массив – это массив, элементами которого служат массивы. Например, массив int a[4][5] – это массив из указателей int*, которые содержат имена одноименных массивов из 5 целых элементов:



Например:
*a[0] == &a[0][0] == a+0*n*sizeof(int);*
*a[1] == &a[1][0] == a+1*n*sizeof(int);*
*a[i] == &a[i][0] == a+i*n*sizeof(int);*
Пример работы с масивом:

```
for(int I=0;I<n;I++)
for(int j=0;j<n;j++)
if(I<j)
{
r[a[I][j];
a[I][j]=a[j][I];
a[j][I]=r;
}
```

2.4 Динамические массивы

Для создания динамических переменных используют операцию new, определенную в C++:

```
указатель = new имя_типа[инициализатор];
```

где инициализатор – выражение в круглых скобках.

Операция `new` позволяет выделить и сделать доступным участок динамической памяти, который соответствует заданному типу данных. Если задан инициализатор, то в этот участок будет занесено значение, указанное в инициализаторе.

```
int* x=new int(5);
```

Для удаления динамических переменных используется операция delete, определенная в C++:

```
delete указатель;
```

где указатель содержит адрес участка памяти, ранее выделенный с помощью операции `new`.

```
delete x;
```

Операция `new` при использовании с массивами имеет следующий формат:

```
new тип_массива
```

Такая операция выделяет для размещения массива участок динамической памяти соответствующего размера, но не позволяет инициализировать элементы массива. Операция new возвращает указатель, значением которого служит адрес первого элемента массива. При выделении динамической памяти размеры массива должны быть полностью определены.

```
//выделение динамической памяти 100*sizeof(int) байт
int* a = new int[100];
```

Примеры:

1. int *a=new int[100];//выделение динамической памяти размером 100*sizeof(int) байтов double *b=new double[10];// выделение динамической памяти размером 10*sizeof(double) байтов

2. long(*ka)[4];//указатель на массив из 4 элементов типа long
ka=new[2][4];//выделение динамической памяти размером 2*4*sizeof(long) байтов

3. int**matr=(int**)new int[5][10];//еще один способ выделения памяти под двумерный //массив

4. int **matr;
matr=new int*[4];//выделяем память под массив указателей int* их n элементов
for(int I=0;I<4;I++)matr[I]=new int[6];//выделяем память под строки массива

Указатель на динамический массив затем используется при освобождении памяти с помощью операции delete.

Примеры:

delete[] a;//освобождает память, выделенную под массив, если a адресует его начало
delete[]b;
delete[] ka;
for(I=0;I<4;I++)delete [] matr[I];//удаляем строки
delete [] matr;//удаляем массив указателей

Для работы с динамической памятью используют указатели. С их помощью осуществляется доступ к участкам динамической памяти, которые называются динамическими переменными. Динамические переменные создаются с помощью специальных функций и операций. Они существуют либо до конца работы программ, либо до тех пор, пока не будут уничтожены с помощью специальных функций или операций.

При формирование матрицы сначала выделяется память для массива указателей на одномерные массивы, а затем в цикле с параметром выделяется память под n одномерных массивов.

```
/*выделение динамической памяти под двумерный динамический
массив*/
int** form_matr(int n,int m)
{
int **matr=new int*[n];//выделение памяти по массив указателей
for(int i=0;i<n;i++)
     //выделение памяти 100*sizeof(int) байт для массива значений
     matr[i]=new int [m];
return matr;//возвращаем указатель на массив указателей
}
```

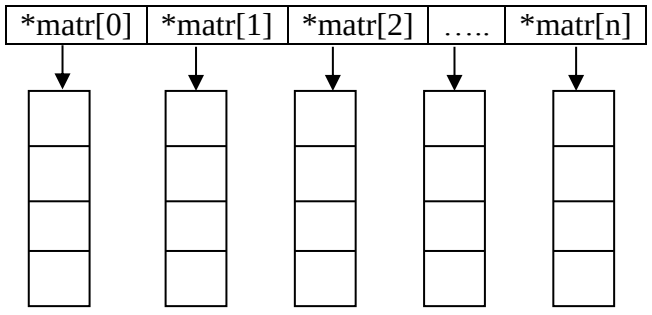


Рис. Выделение памяти под двумерный массив

Изменять значение указателя на динамический массив надо аккуратно, т. к. этот указатель затем используется при освобождении памяти с помощью операции `delete`.

```
/*освобождает память, выделенную под массив, если a адресует его
начало*/
delete[] a;
```

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

При удалении из динамической матрицы строк или столбцов создается новая матрица нужного размера, в которую переписываются данные из старой матрицы. Затем старая матрица удаляется.

```
     int **del(int **matr,int &n,int m)
     {//удаление четных строк
          int k=0,t=0;
          for(int i=0;i<n;i++)
               if(i % 2!=0)k++;//количество нечетных строк
     //выделяем память под новую матрицу
               int **matr2=form_matr(k,m);
               for(i=0;i<n;i++)
                    if(i % 2!=0)
                    {
     //если строка нечетная, то переписываем ее в новую матрицу
                         for(int j=0;j<m;j++)
                         matr2[t][j]=matr[i][j];
                         t++;
                    }
                    n=t;//изменяем количество строк
//возвращаем указатель на новую матрицу как результат функции
     return matr2;
     }
```

Пример1.

```
#include <iostream>
#include <stdlib.h>
using namespace std;
/*выделение динамической памяти под двумерный динамический массив*/
int** form_matr(int n,int m){
 int **matr=new int*[n]; //выделение памяти по массив указателей

 for(int i=0;i<n;i++)
       //выделение памяти 100*sizeof(int) байт для массива значений
       matr[i]=new int [m];

 return matr;//возвращаем указатель на массив указателей
}

void zapolnen (int ** matr,int n,int m){
//заполнение матрицы
       for(int i=0;i<n;i++)
       for(int j=0;j<m;j++)
```

```
                matr[i][j]=rand()%10;//заполнение матрицы
}

void printArray (int ** matr,int n,int m){
//печать сформированной матрицы
        for(int i=0;i<n;i++)
        {
                for(int j=0;j<m;j++)
                        cout<<matr[i][j]<<" ";
                cout<<"\n";

        }
}
int ** dell(int ** matr,int n,int m){
        //удаление старой матрицы
        for(int i=0;i<n;i++)
                delete [] matr[i];
        delete[]matr;
matr=NULL;
return matr;
}


int main()
{
        int n,m;//размерность матрицы
        int i,j;
        cout<<"\nEnter n";
        cin>>n;//строки
        cout<<"\nEnter m";
        cin>>m;//столбцы
        //выделение памяти
        int **matr=form_matr(n,m);
        zapolnen(matr,n,m);
        printArray(matr,n,m);

        //удаление строки с номером к
        int k;
        cout<<"\nEnter k from 0 to"<<n<<endl;
        cin>>k;
        int**temp=form_matr(n-1,m);

                //заполнение новой матрицы
        int t;
        for(i=0,t=0;i<n;i++)
                if(i!=k)
                {
                        for(j=0;j<m;j++)
                                temp[t][j]=matr[i][j];
                        t++;
                }

        matr=dell(matr, n, m);
```

```
n--;
        //печать новой матрицы
       printArray(temp,n,m);
temp=dell(temp, n, m);
        return 0;
}
```

Пример 2. Удалить из массива все элементы, совпадающие с первым элементом, используя динамическое выделение памяти.

```
#include <iostream>
#include <stdlib.h>
using namespace std;
int* form(int&n)
{

        cout<<"\nEnter n";
        cin>>n;
        int*a=new int[n];//указатель на динамическую область памяти
        for(int i=0;i<n;i++)
                a[i]=rand()%100;
        return a;
}
void print(int*a,int n)
{
        for(int i=0;i<n;i++)
                cout<<a[i]<<" ";
        cout<<"\n";
}

int*Dell(int *a,int&n)
{
        int k,j,i;
        for(k=0,i=0;i<n;i++)
                if(a[i]!=a[0])k++;
                int*b;
                b=new int [k];
                for(j=0,i=0;i<n;i++)
                if(a[i]!=a[0])
                {
                        b[j]=a[i];j++;
                }
                n=k;
                return b;
}

int main()
{
        int *a;
        int n;
        a=form(n);
        print(a,n);
        a=Dell(a,n);
        print(a,n);
```

```
        return 0;
    }
```

3. Постановка задачи

1. Сформировать динамический одномерный массив, заполнить его случайными числами и вывести на печать.
2. Выполнить указанное в варианте задание и вывести полученный массив на печать.
3. Сформировать динамический двумерный массив, заполнить его случайными числами и вывести на печать.
4. Выполнить указанное в варианте задание и вывести полученный массив на печать.

4. Варианты

№ варианта	Одномерный массив	Двумерный массив
1	Удалить первый четный элемент	Добавить строку с заданным номером
2	Удалить первый отрицательный элемент	Добавить столбец с заданным номером
3	Удалить элемент с заданным ключом (значением)	Добавить строку в конец матрицы
4	Удалить элемент равный среднему арифметическому элементов массива	Добавить столбец в конец матрицы
5	Удалить элемент с заданным номером	Добавить строку в начало матрицы
6	Удалить N элементов, начиная с номера K	Добавить столбец в начало матрицы
7	Удалить все четные элементы	Добавить K строк в конец матрицы
8	Удалить все элементы с четными индексами	Добавить K столбцов в конец матрицы
9	Удалить все нечетные элементы	Добавить K строк в начало матрицы
10	Удалить все элементы с нечетными индексами	Добавить K столбцов в начало матрицы
11	Добавить элемент в начало массива	Удалить строку с номером K
12	Добавить элемент в конец массива	Удалить столбец с номером K
13	Добавить K элементов в начало массива	Удалить строки, начиная со строки K1 и до строки K2
14	Добавить K элементов в конец массива	Удалить столбцы, начиная со столбца K1 и до столбца K2
15	Добавить K элементов, начиная с номера N	Удалить все четные строки
16	Добавить после каждого отрицательного элемента его модуль	Удалить все четные столбцы
17	Добавить после каждого четного элемента элемент со значением 0	Удалить все строки, в которых есть хотя бы один нулевой элемент

5. Методические указания

1. Для выделения памяти под массивы использовать операцию new, для удаления массивов из памяти – операцию delete.

2. Для выделения памяти, заполнения массивов, удаления и добавления элементов (строк, столбцов) написать отдельные функции. В функции main() должны быть размещены только описания переменных и обращения к соответствующим функциям:

```
int main()
{
      int n;
      cout<<"N?";cin>>n;
      person*mas=form_mas(n);
      init_mas(mas,n);
      print_mas(mas,n);
      return 1;
}
```

3. Для реализации интерфейса использовать текстовое меню:

```
….
do
{
cout<<”1. Формирование массива\n”;
cout<<”2. Печать массива\n”;
cout<<”3. Удаление из массива\n”;
cout<<”4. Добавление в массив\n”;
cout<<”5. Выход\n”;
cin>>k;
switch (k)
{
case 1: mas=form_mas(SIZE);input_mas(mas,SIZE); break;//выделение памяти и заполнение
case 2: print_mas(mas,SIZE); break;//печать
case 3: del_mas(mas,SIZE);break;//удаление
case 4: add_mas(mas,SIZE);break;//добавление
}
while (k!=5);//выход
```

4. При удалении элементов (строк, столбцов) предусмотреть ошибочные ситуации, т. е. ситуации, в которых будет выполняться попытка удаления элемента (строки, столбца) из пустого массива или количество удаляемых элементов будет превышать количество имеющихся элементов (строк, столбцов). В этом случае должно быть выведено сообщение об ошибке.

6. Содержание отчета

1. Постановка задачи (общая и для конкретного варианта).
2. Определения функций для реализации поставленных задач.
3. Определение функции main().

Тесты