Лабораторная работа №3
Работа с массивами

1. Цель работы:

1) Получение практических навыков при работе с массивами.
2) Получение практических навыков при работе с указателями.

2. Краткие теоретические сведения

Массив – это упорядоченная последовательность переменных одного типа. Каждому элементу массива отводится одна ячейка памяти. Элементы одного массива занимают последовательно расположенные ячейки памяти. Все элементы имеют одно имя – имя массива и отличаются индексами – порядковыми номерами в массиве. Количество элементов в массиве называется его размером. Чтобы отвести в памяти нужное количество ячеек для размещения массива, надо заранее знать его размер. Резервирование памяти для массива выполняется на этапе компиляции программы.

2.1. Определение массива в C/C++

Массивы определяются следующим образом:

```
int a[100];//массив из 100 элементов целого типа
```

Операция `sizeof(a)` даст результат 400, т. е. 100 элементов по 4 байта.
Элементы массива всегда нумеруются с 0.

45	352	63		124	значения элементов массива
0	1	2	…..	99	индексы элементов массива

Чтобы обратиться к элементу массива, надо указать имя массива и номер элемента в массиве (индекс):
`a[0]` – индекс задается как константа,
`a[55]` – индекс задается как константа,
`a[i]` – индекс задается как переменная,
`a[2*i]` – индекс задается как выражение.

Элементы массива можно задавать при его определении:

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

2.2. 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.

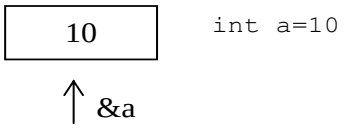


Рис. 3. Значение переменной и ее адрес

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;
```

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:

2.3. 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.

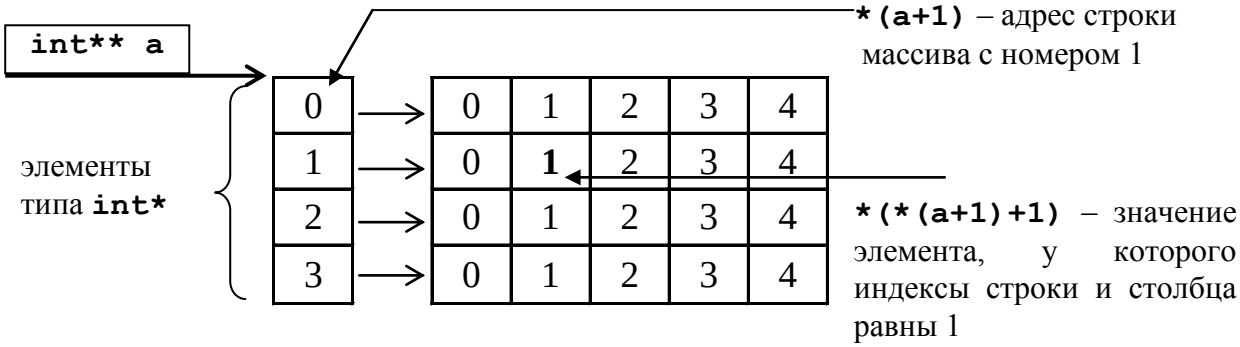
Указатели можно использовать и при определении массивов:

```
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

2.4.1 Динамические массивы в языке C++

В языке C++ 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.

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(*la)[4];//`указатель на массив из 4 элементов типа long

`la=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[] la;
for(i=0;i<4;i++)delete [] matr[i];//удаляем строки
delete [] matr;//удаляем массив указателей
```

Пример

Удалить из матрицы строку с номером K

```
#include <iostream>
#include <string.h>
#include <stdlib.h>
using namespace std;
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=new int* [n];// массив указателей на строки
     for(i=0;i<n;i++)
          matr[i]=new int [m];//память под элементы матрицы
     //заполнение матрицы
     for(i=0;i<n;i++)
     for(j=0;j<m;j++)
          matr[i][j]=rand()%10;//заполнение матрицы
     //печать сформированной матрицы
     for(i=0;i<n;i++)
     {
          for(j=0;j<m;j++)
               cout<<matr[i][j]<<" ";
          cout<<"\n";
     }
     //удаление строки с номером к
     int k;
     cout<<"\nEnter k";
     cin>>k;
     int**temp=new int*[n-1];//формирование новой матрицы
     for(i=0;i<n;i++)
          temp[i]=new int[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++;
            }

            //удаление старой матрицы
      for(i=0;i<n;i++)
            delete matr[i];
      delete[]matr;
n--;
      //печать новой матрицы

      for(i=0;i<n;i++)
      {
            for(j=0;j<m;j++)
                  cout<<temp[i][j]<<" ";
            cout<<"\n";
      }
return 0;
}
```

2.4.1 Динамические массивы в языке С

В С захват и освобождение памяти выполняются с помощью функций стандартной библиотеки #include <stdlib.h> - это так называемый С-стиль работы с динамической памятью.

Для решения задач, работающих с динамической памятью, в С предусмотрено всего четыре функции: malloc, calloc, realloc, free.

Первые две захватывают память (alloc в названии от слова allocation - резервирование). Третья, realloc, позволяет изменять (как правило, увеличивать) размер уже захваченного массива без потери его содержимого. Последняя функция освобождает уже ненужную память.

Функция malloc захватывает область памяти, размер в байтах этой области вы указываете в качестве аргумента. Функция возвращает указатель на адрес захваченной памяти. При этом выделяемая память никак не инициализируется, не очищается.

Формат функции:

```
malloc ( размер памяти в байтах);
```

```
int *m1 = (int * ) malloc ( 100 * sizeof (int)) ;
float *m2 = (float * ) malloc ( 200 * sizeof (float)) ;
```

где m1- переменная-указатель на массив 100 целых значений типа int;
m2 - переменная-указатель на массив 200 целых значений типа float.

Освобождение выделенной памяти осуществляется с помощью функции `free`:

`free` (указатель на массив) ;

```
free(m1);
free(m2);
```

Функции calloc, realloc рассмотреть самостоятельно.

2.5. Перебор элементов массива

1) Элементы массива можно обрабатывать по одному элементу, двигаясь от начала массива к его концу (или в обратном направлении):
   `for(int i=0;i<n;i++) <обработка a[i]>`
2) Элементы массива можно обрабатывать по два элемента, двигаясь с обеих сторон массива к его середине:

```
int i=0,j=n-1;
while (i<j){
<обработка a[I] и a[j]>;
i++;j--;}
```

3) Элементы массива можно обрабатывать по два элемента, двигаясь от начала к концу с шагом 1(т. е. обрабатываются пары элементов a[0]и a[1], a[1]и a[2] и т. д.)

```
for(i=0;i<n-1;i++)
<обработка a[i] и a[i+1]>
```

4) Элементы массива можно обрабатывать по два элемента, двигаясь от начала к концу с шагом 2(т. е. обрабатываются пары элементов a[0]и a[1], a[2]и a[3] и т. д.)

```
i=1;
while(i<n){
<обработка a[i] и a[i+1]>
i=i+2;}
```

3. Постановка задачи

1) Сформировать массив из n элементов с помощью датчика случайных чисел (n задается пользователем с клавиатуры).

2) Распечатать полученный массив.

3) Для задачи1 (в стиле С) и задачи 2(в стиле С++), использовать динамическое выделение памяти. Вывести полученный результат.

4) Для задачи 3 и задачи 4 использовать статический массив, размер указывать явно.

5) Сформировать динамический и статический двумерный массив – задача 5 и задача 6 (таб.2), заполнить его случайными числами и вывести на печать.

6) Вывести полученный результат.

4. Варианты

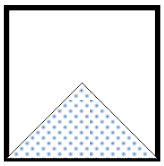
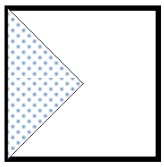
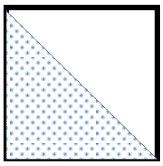
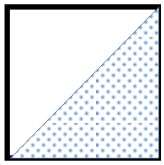
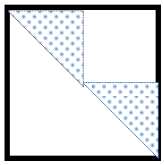
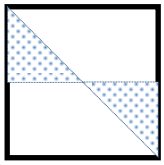
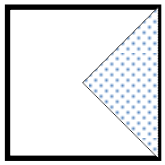
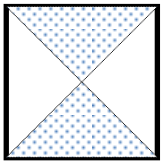
Таблица 1. Одномерные массивы.

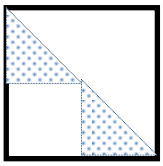
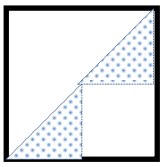
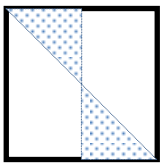
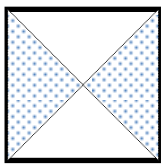
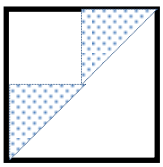
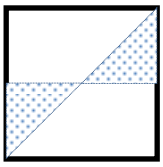
Вариант	**Одномерный динамический массив**		**Одномерный статический массив**	
	Удаление (исключить из последовательности)	**Добавление**	**Перестановка**	**Поиск**
1	Максимальный элемент	К элементов в начало последователь-ности	Перевернуть массив	Первый четный
2	Минимальный элемент	К элементов в конец последователь-ности	Сдвинуть циклически на М элементов вправо	Первый отрицательный
3	Элемент с заданным номером	N элементов, начиная с номера К	Сдвинуть циклически на М элементов влево	Элемент с заданным ключом (значением)
4	N элементов, начиная с номера К	Элемент с номером К	Поменять местами элементы с четными и нечетными номерами	Элемент равный среднему арифметическо му элементов массива
5	Все четные элементы	К элементов в начало	Четные элементы переставить в начало	Первый четный

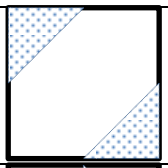
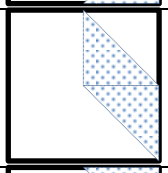
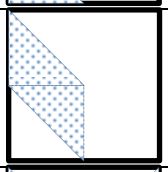
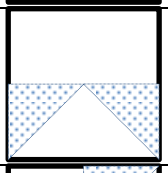
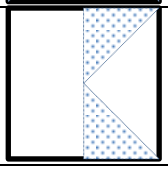
Вариант	Одномерный динамический массив		Одномерный статический массив	
	Удаление (исключить из последовательности)	**Добавление**	**Перестановка**	**Поиск**
		последователь-ности	массива, нечетные - в конец	
6	Все элементы с четными индексами	K элементов в конец последователь-ности	Поменять местами минимальный и максимальный элементы	Первый отрицательный
7	Все нечетные элементы	N элементов, начиная с номера K	Положительные элементы переставить в начало массива, отрицательные - в конец	Элемент с заданным ключом (значением)
8	Все элементы с нечетными индексами	Элемент с номером K	Перевернуть массив	Элемент равный среднему арифметическо му элементов массива
9	Все элементы больше среднего арифметического элементов последовательности	K элементов в начало последователь-ности	Сдвинуть циклически на M элементов вправо	Первый четный
10	Максимальный элемент среди нечетных	K элементов в конец последователь-ности	Сдвинуть циклически на M элементов влево	Первый отрицательный
11	Минимальный элемент среди четных	N элементов, начиная с номера K	Поменять местами элементы с четными и нечетными номерами	Элемент с заданным ключом (значением)
12	Элемент с заданным номером	Элемент с номером K	Четные элементы переставить в начало массива, нечетные - в конец	Элемент равный среднему арифметическо му элементов массива
13	N элементов, начиная с номера K	K элементов в начало последователь-ности	Поменять местами минимальный и максимальный элементы	Первый четный
14	Все четные элементы	K элементов в конец последователь-ности	Положительные элементы переставить в начало массива, отрицательные - в конец	Первый отрицательный
15	Все элементы с четными индексами	N элементов последователь-ности, начиная с номера K	Перевернуть массив	Элемент с заданным ключом (значением)

Вариант	Одномерный динамический массив		Одномерный статический массив	
	Удаление (исключить из последовательности)	**Добавление**	**Перестановка**	**Поиск**
16	Все нечетные элементы	Элемент с номером K	Сдвинуть циклически на M элементов вправо	Элемент равный среднему арифметическому элементов массива
17	Все элементы с нечетными индексами	K элементов в начало последователь-ности	Сдвинуть циклически на M элементов влево	Первый четный
18	Все элементы больше среднего арифметического элементов последовательности	K элементов в конец последователь-ности	Поменять местами элементы с четными и нечетными номерами	Первый отрицательный
19	Максимальный элемент среди кратных 3	N элементов, начиная с номера K	Четные элементы переставить в начало массива, нечетные - в конец	Элемент с заданным ключом (значением)
20	Минимальный элемент среди чисел кратных 5	Элемент с номером K	Поменять местами минимальный и максимальный элементы	Элемент равный среднему арифметическому элементов массива
21	Элемент с заданным номером	K элементов в начало последователь-ности	Положительные элементы переставить в начало массива, отрицательные - в конец	Первый четный
22	N элементов, начиная с номера K	K элементов в конец последователь-ности	Перевернуть массив	Первый отрицательный
23	Все четные элементы	N элементов, начиная с номера K	Сдвинуть циклически на M элементов вправо	Элемент с заданным ключом (значением)
24	Все элементы с четными индексами	Элемент с номером K	Сдвинуть циклически на M элементов влево	Элемент равный среднему арифметическому элементов массива
25	Все нечетные элементы	K элементов в начало последователь-ности	Поменять местами элементы с четными и нечетными номерами	Первый четный

Таблица 2. Двумерные массивы

Вариант	Двумерный статический массив	Двумерный динамический массив
	Найти сумму элементов заштрихованной области	**Выполнить**
1		Добавить строку с заданным номером
2		Добавить столбец с заданным номером
3		Добавить строку в конец матрицы
4		Добавить столбец в конец матрицы
5		Добавить строку в начало матрицы
6		Добавить столбец в начало матрицы
7		Добавить К строк в конец матрицы
8		Добавить К столбцов в конец матрицы
9		Добавить К строк в начало матрицы

Вариант	Двумерный статический массив	Двумерный динамический массив
	Найти сумму элементов заштрихованной области	**Выполнить**
10		Добавить К столбцов в начало матрицы
11		Удалить строку с номером К
12		Удалить столбец с номером К
13		Удалить строки, начиная со строки К1 и до строки К2
14		Удалить столбцы, начиная со столбца К1 и до столбца К2
15		Удалить все четные строки
16		Удалить все четные столбцы
17		Удалить все строки, в которых есть хотя бы один нулевой элемент
18		Удалить все столбцы, в которых есть хотя бы один нулевой элемент

Вариант	Двумерный статический массив	Двумерный динамический массив
	Найти сумму элементов заштрихованной области	**Выполнить**
19		Удалить строку, в которой находится наибольший элемент матрицы
20		Добавить строки после каждой четной строки матрицы
21		Добавить столбцы после каждого четного столбца матрицы
22		Добавить К строк, начиная со строки с номером N
23		Добавить К столбцов, начиная со столбца с номером N
24		Добавить строку после строки, содержащей наибольший элемент
25		Добавить столбец после столбца, содержащего наибольший элемент

5. Методические указания

1. Статические массивы реализуются следующим образом:

1) при определении массива выделяется достаточно большое количество памяти:

```
const int MAX_SIZE=100;//именованная константа
int mas[MAX_SIZE];
```

2) пользователь вводит реальное количество элементов массива меньшее N.

```
int n;
cout<<"\nEnter the size of array<"<<MAX_SIZE<<":";cin>>n;
```

3) дальнейшая работа с массивом ограничивается заданной пользователем размерностью n.

2. Формирование массива осуществляется с помощью датчика случайных чисел. Для этого можно использовать функцию `int rand()`, которая возвращает псевдослучайное число из диапазона `0..RAND_MAX=32767`, описание функции находится в файле `<stdlib.h>`. В массиве должны быть записаны и положительные и отрицательные элементы. Например, оператор `a[I]=rand()%100-50;` формирует псевдослучайное число из диапазона [-50;49].

3. Вывод результатов должен выполняться после выполнения каждого задания. Элементы массива рекомендуется выводить в строчку, разделяя их между собой пробелом.

6. Содержание отчета:

1) Постановка задачи (общая и конкретного варианта).
2) Анализ поставленного задания: определить к какому классу задач относится задача и объяснить почему.
3) Текст программы.
4) Результаты тестов.
5) Решение одной из задач с использованием указателей для доступа к элементам массива.