Лабораторная работа №4

Функции и массивы в C/C++

1. Цель работы:

1) Получение практических навыков при работа со строками, одномерными и двумерными массивами.
2) Получение практических навыков при работе с функциями
3) Получение практических навыков при передаче массивов в функции.
4) Получить практические навыки выделения, перераспределения и освобождение памяти при работе с динамическими массивами

2. Теоретические сведения

Функция – это именованная последовательность описаний и операторов, выполняющая законченное действие, например, формирование массива, печать массива и т. д.

Любая функция должна быть объявлена и определена.

- Объявление функции (прототип, заголовок) задает имя функции, тип возвращаемого значения и список передаваемых параметров.
- Определение функции содержит, кроме объявления, тело функции, которое представляет собой последовательность описаний и операторов.

```
тип имя_функции([список_формальных_параметров])
{ тело_функции}
```

- Тело_функции – это блок или составной оператор. Внутри функции нельзя определить другую функцию.

В теле функции должен быть оператор, который возвращает полученное значение функции в точку вызова. Он может иметь 2 формы:

1) return выражение;
2) return;

Первая форма используется для возврата результата, поэтому выражение должно иметь тот же тип, что и тип функции в определении. Вторая форма используется, если функция не возвращает значения, т. е. имеет тип void. Программист может не использовать этот оператор в теле функции явно, компилятор добавит его автоматически в конец функции перед }.

- Тип возвращаемого значения может быть любым, кроме массива и функции, но может быть указателем на массив или функцию.
- Список формальных параметров – это те величины, которые требуется передать в функцию. Элементы списка разделяются запятыми. Для каждого параметра указывается тип и имя. В объявлении имена можно не указывать.

Для того, чтобы выполнялись операторы, записанные в теле функции, функцию необходимо вызвать. При вызове указываются: имя функции и фактические параметры. Фактические параметры заменяют формальные параметры при выполнении операторов тела функции. Фактические и формальные параметры должны совпадать по количеству и типу.

Объявление функции должно находиться в тексте раньше вызова функции, чтобы компилятор мог осуществить проверку правильности вызова. Если функция имеет тип не void, то ее вызов может быть операндом выражения.

2.1. Параметры функции

Основным способом обмена информацией между вызываемой и вызывающей функциями является механизм параметров. Существует два способа передачи параметров в функцию: по адресу и по значению.

- При передаче по значению выполняются следующие действия:
  - вычисляются значения выражений, стоящие на месте фактических параметров;
  - в стеке выделяется память под формальные параметры функции;

- каждому фактическому параметру присваивается значение формального параметра, при этом проверяются соответствия типов и при необходимости выполняются их преобразования.

```
void Change(int a,int b)//передача по значению
{
     int r=a;
     a=b;
     b=r;
}
int main()
{
int x=1,y=5;
Change(x,y);
cout<<"x="<<x<<" y="<<y; //выведется: x=1 y=5

return 1;
}
```

- При передаче по адресу в стек заносятся копии адресов параметров, следовательно, у функции появляется доступ к ячейке памяти, в которой находится фактический параметр и она может его изменить.

```
void Change(int *a,int *b)//передача по адресу
{
     int r=*a;
     *a=*b;
     *b=r;
}
int main()
{
int x=1,y=5;
Change(&x,&y);
cout<<"x="<<x<<" y="<<y; //выведется: x=5 y=1

return 1;
}
```

Для передачи по адресу также могут использоваться ссылки. Ссылка – это синоним имени объекта, указанного при инициализации ссылки.

Формат объявления ссылки

```
тип & имя =имя_объекта;
```

Ссылка не занимает дополнительного пространства в памяти, она является просто другим именем объекта.

При передаче по ссылке в функцию передается адрес указанного при вызове параметра, а внутри функции все обращения к параметру неявно разыменовываются.

```
void Change(int &a,int &b)
{
     int r=a;
     a=b;
     b=r;
}
int main()
{
int x=1,y=5;
Change(x,y);
cout<<"x="<<x<<" y="<<y; //выведется: x=5 y=1
```

```
return 1;
}
```

Использование ссылок вместо указателей улучшает читаемость программы, т. к. не надо применять операцию разыменовывания. Использование ссылок вместо передачи по значению также более эффективно, т .к. не требует копирования параметров. Если требуется запретить изменение параметра внутри функции, используется модификатор const. Рекомендуется ставить const перед всеми параметрами, изменение которых в функции не предусмотрено (по заголовку будет понятно, какие параметры в ней будут изменяться, а какие нет).

2.2. Локальные и глобальные переменные

- Переменные, которые используются внутри данной функции, называются локальными. Память для них выделяется в стеке, поэтому после окончания работы функции они удаляются из памяти. Нельзя возвращать указатель на локальную переменную, т. к. память, выделенная такой переменной, будет освобождаться.
- Глобальные переменные – это переменные, описанные вне функций. Они видны во всех функциях, где нет локальных переменных с такими именами.

2.3. Передача одномерных массивов как параметров функции

При использовании массива как параметра функции, в функцию передается указатель на его первый элемент, т. е. массив всегда передается по адресу. При этом теряется информация о количестве элементов в массиве, поэтому размерность массива следует передавать как отдельный параметр.

```
void print(int a[100],int n) //вывод массива на печать
{
     for(int i=0;i<n;i++)
          cout<<a[i]<<" ";
     cout<<"\n";
}
```

Так как в функцию передается указатель на начало массива (передача по адресу), то массив может быть изменен за счет операторов тела функции.

2.5. Передача многомерных массивов в функцию

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:

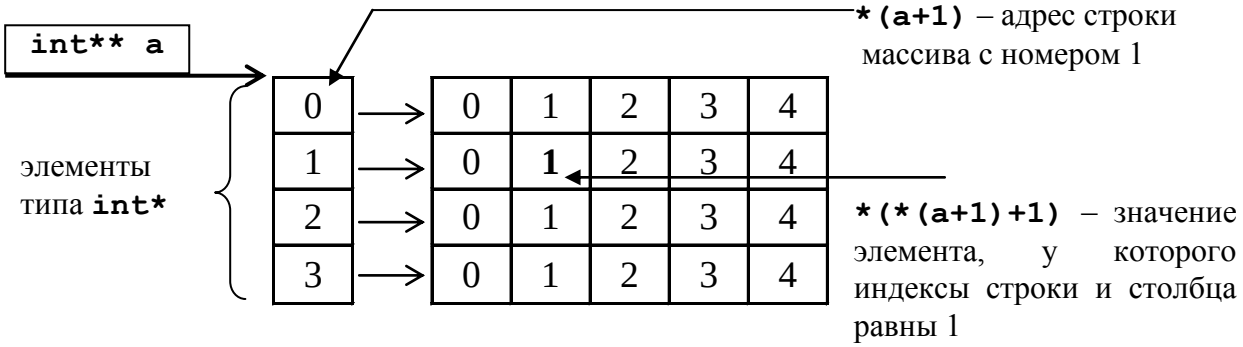

Рис. Выделение памяти под массив, элементами которого являются массивы.

При передаче многомерных массивов в функцию все размерности должны передаваться в качестве параметров.

```
const int N=4;//глобальная переменная
```

```
void transp(int a[][N],int n)// транспонирование матрицы
{
     int r;
     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;
     }
}
```

2.6 Динамические массивы

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 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];
```

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
}
```

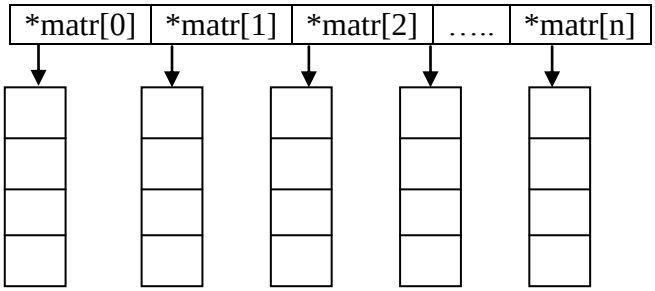

Рис. Выделение памяти под двумерный массив

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.

```
int find(int **matr,int m,int I)
{
     for(int i=0;i<m;i++)
          if(matr[I][i]<0) return 1;
          return 0;
}
```

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;
     }
```

Пример:

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

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

int * creat( int &s)
{
    cout<<"Enter size array";
    cin>>s;
    int * temp= new int[s];
    for (int i=0; i<s; i++) {
        cin>>temp[i];
    }
    return temp;
}
void print (int a[],int size ){
    for (int i=0; i<size; i++) {
        cout<<a[i]<<" ";
    }
    cout<<endl;
}
int* dell(int *a,int &size){
int kol=0;
for (int i=1; i<size; i++) {
    if (a[i]!=a[0]) kol++;
    }

    cout<<"kol="<<kol<<endl;
int *temp=new int[kol];
int j=0;
for (int i=0; i<size; i++) {
    if (a[i]!=a[0])
        {temp[j]=a[i];
        j++;}
}
delete []a;
size=kol;
return temp;
}

int main()
{
    int *a;
    int size;
    a=creat(size);
    cout<<"size="<<size<<endl;
    print(a,size);
    a=dell(a,size);
    print(a,size);
     return 0;
}
```

2.7. Сортировка массивов

Сортировка – это процесс перегруппировки заданного множества объектов в некотором установленном порядке.

2.7.1. Сортировка с помощью включения

Элементы массива делятся на уже готовую последовательность и исходную. При каждом шаге, начиная с I=2, из исходной последовательности извлекается i-ый элемент и вставляется на нужное место готовой последовательности, затем i увеличивается на 1 и т. д.

44	55	12	42	94	18

готовая исходная

В процессе поиска нужного места осуществляются пересылки элементов больше выбранного на одну позицию вправо, т. е. выбранный элемент сравнивают с очередным элементом отсортированной части, начиная с j=i-1. Если выбранный элемент больше a[i], то его включают в отсортированную часть, в противном случае a[j] сдвигают на одну позицию, а выбранный элемент сравнивают со следующим элементом отсортированной последовательности. Процесс поиска подходящего места заканчивается при двух различных условиях:

- если найден элемент a[j]>a[i];
- достигнут левый конец готовой последовательности.

```
int i,j,x;
for(i=1;i<n;i++)
{
x=a[i];//запомнили элемент, который будем вставлять
j=i-1;
while(x<a[j]&&j>=0)//поиск подходящего места
{
      a[j+1]=a[j];//сдвиг вправо
       j--;
}
a[j+1]=x;//вставка элемента
}
```

2.7.2. Сортировка методом простого выбора

Выбирается минимальный элемент массива и меняется местами с первым элементом массива. Затем процесс повторяется с оставшимися элементами и т. д.

44	55	12	42	94	18

1 мин

```
int i,min,n_min,j;
for(i=0;i<n-1;i++)
      {
      min=a[i];n_min=i;//поиск минимального
      for(j=i+1;j<n;j++)
          if(a[j]<min)
      {
          min=a[j];
          n_min=j;
      }
      a[n_min]=a[i];//обмен
      a[i]=min;
      }
```

2.7.3. Сортировка методом простого обмена

Сравниваются и меняются местами пары элементов, начиная с последнего. В результате самый маленький элемент массива оказывается самым левым элементом массива. Процесс повторяется с оставшимися элементами массива.

44	55	12	42	94	18

```
for(int i=1;i<n;i++)
for(int j=n-1;j>=i;j--)
      if(a[j]<a[j-1])
      {
          int r=a[j];
          a[j]=a[j-1];
          a[j-1]=r;}
      }
```

2.7.5 Поиск в отсортированном массиве

В отсортированном массиве используется дихотомический (бинарный) поиск. При последовательном поиске требуется в среднем n/2 сравнений, где n – количество элементов в массиве. При дихотомическом поиске требуется не более m сравнений, если n- m-ая степень 2, если n не является степенью 2, то $n<k=2^m$.

Массив делится пополам `S=(L+R)/ 2+1` и определяется в какой части массива находится нужный элемент X. Так как массив упорядочен, то, если a[S]<X – искомый элемент находится в правой части массива, иначе – находится в левой части. Выбранную часть массива снова надо разделить пополам и т. д., до тех пор, пока границы отрезка L и R не станут равны.

1	3	8	10	11	15	19	21	23	37
0	1	2	3	4	5	6	7	8	9

L S R

```
int b;
cout<<"\nB=?";cin>>b;
int l=0,r=n-1,s;
do
{
     s=(l+r)/2;//средний элемент
     if(a[s]<b)l=s+1;//перенести леую границу
     else r=s;//перенести правую границу
}while(l!=r);
if(a[l]==b)return l;
else return -1;
…
```

3. Постановка задачи

1. Используя функции сформировать с помощью ДСЧ одномерный массив и вывести его на консоль.
2. Выполнить обработку одномерного массива (задача 1, задача 2) в соответствии с вариантом, используя функции, результат вывести на консоль.
3. Сформировать динамический одномерный массив – задача 3, заполнить его случайными числами и вывести на консоль.
4. Выполнить указанное в варианте задание и вывести полученный массив на консоль.

5. Сформировать динамический и псевдодинамический двумерный массив – задача 4, заполнить его случайными числами и вывести на консоль (двумя способами).
6. Выполнить указанное в варианте задание и вывести полученный массив на консоль.

4. Варианты

Таблица 1

Вариант	Одномерный псевдодинамический массив		Динамические массивы	
	Сортировка		Одномерный массив	Двумерный массив
1	Простой обмен	Отсортировать по возрастанию только четные элементы массива.	Удалить первый четный элемент	Перевернуть все четные строки матрицы.
2	Простой выбор	Удалить из массива все четные элементы.	Удалить первый отрицательный элемент	Перевернуть все четные столбцы матрицы.
3	Простое включение	Найти количество простых чисел в массиве.	Удалить элемент с заданным ключом (значением)	Перевернуть все нечетные строки матрицы.
4	Простой обмен	Найти количество чисел Фибоначчи в массиве.	Удалить элемент равный среднему арифметическому элементов массива	Перевернуть все нечетные столбцы матрицы.
5	Простой выбор	Удалить все простые числа из массива.	Удалить элемент с заданным номером	Отсортировать по убыванию все строки матрицы.
6	Простое включение	Удалить из массива все числа Фибоначчи.	Удалить N элементов, начиная с номера K	Отсортировать по убыванию столбцы матрицы.
7	Простой обмен	Отсортировать по возрастанию только положительные элементы массива.	Удалить все четные элементы	Меняя местами строки матрицы, отсортировать по возрастанию ее первый столбец.
8	Простой выбор	Удалить из массива все элементы с четными номерами.	Удалить все элементы с четными индексами	Меняя местами столбцы матрицы, отсортировать по возрастанию ее первую строку.
9	Простое включение	Отсортировать по возрастанию только те элементы массива, которые являются простыми числами.	Удалить все нечетные элементы	Все четные строки матрицы сдвинуть циклически на K элементов вправо.
10	Простой обмен	Удалить из массива все элементы равные min(a[1],a[3],…a[2n-1]).	Удалить все элементы с нечетными индексами	Все нечетные строки матрицы сдвинуть циклически на K элементов влево.
11	Простой выбор	Создать новый массив из номеров элементов, значения которых равны 0.	Добавить элемент в начало массива	Перевернуть все четные строки матрицы.

12	Простое включение	Сформировать массив, в котором будут только элементы исходного массива, заканчивающиеся на цифру К.	Добавить элемент в конец массива	Перевернуть все четные столбцы матрицы.
13	Простой обмен	Отсортировать по возрастанию только четные элементы массива.	Добавить К элементов в начало массива	Перевернуть все нечетные строки матрицы.
14	Простой выбор	Удалить из массива все четные элементы.	Добавить К элементов в конец массива	Перевернуть все нечетные столбцы матрицы.
15	Простое включение	Найти количество простых чисел в массиве.	Добавить К элементов, начиная с номера N	Отсортировать по убыванию все строки матрицы.
16	Простой обмен	Найти количество чисел Фибоначчи в массиве.	Добавить после каждого отрицательного элемента его модуль	Отсортировать по убыванию все столбцы матрицы.
17	Простой выбор	Удалить все простые числа из массива.	Добавить после каждого четного элемента элемент со значением 0	Меняя местами строки матрицы, отсортировать по возрастанию ее первый столбец.
18	Простое включение	Удалить из массива все числа Фибоначчи.	Добавить по К элементов в начало и в конец массива	Меняя местами столбцы матрицы, отсортировать по возрастанию ее первую строку.
19	Простой обмен	Отсортировать по возрастанию только положительные элементы массива.	Добавить элемент с номером К	Все четные строки матрицы сдвинуть циклически на К элементов вправо.
20	Простой выбор	Удалить из массива все элементы с четными номерами.	Удалить элемент с заданным номером	Все нечетные строки матрицы сдвинуть циклически на К элементов влево.
21	Простое включение	Отсортировать по возрастанию только те элементы массива, которые являются простыми числами.	Удалить N элементов, начиная с номера К	Перевернуть все четные строки матрицы.
22	Простой обмен	Удалить из массива все элементы равные min(a[1],a[3],…a[2n-1]).	Удалить все четные элементы	Перевернуть все четные столбцы матрицы.

23	Простой выбор	Создать новый массив из номеров элементов, значения которых равны 0.	Удалить все элементы с четными индексами	Перевернуть все нечетные строки матрицы.
24	Простое включение	Сформировать массив, в котором будут только элементы исходного массива, заканчивающиеся на цифру К.	Удалить все нечетные элементы	Перевернуть все нечетные столбцы матрицы.
25	Простой обмен	Сформировать два массива. В первый массив включить элементы из исходного массива с четными номерами, а во второй с нечетными.	Удалить все элементы с нечетными индексами	Отсортировать по убыванию все строки матрицы.

5. Методические указания

1. Формирование, печать и обработку массивов оформить в виде функции. Массивы передавать как параметры функций.
2. Реализовать массивы в первой и во второй задаче, как псевдодинамические, их размерности передавать как параметры функций.
3. Формирование массивов выполнить с использованием ДСЧ. В массивы записывать и положительные, и отрицательные числа.
4. Сортировку массивов организовать с помощью одного из простых методов сортировки.
5. Функция main() должна содержать только описание массивов и вызовы функций для формирования, печати и обработки массивов.
6. В задаче 2 и 3 для выделения памяти под массивы использовать операцию new, для удаления массивов из памяти – операцию delete.
7. 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;
}
```

1. 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
```

2. 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, используемых для формирования, печати и обработки массивов/строк (для каждой задачи).
3. Определение функции main().
4. Результаты тестов.