Практическое занятие по написанию **программ с использованием одномерных массивов**

Цель работы: освоение программирования обработки массивов с применением циклов, с организацией файлового ввода/вывода, с использованием динамической памяти

Базовый материал: последняя очная лекция (циклы) и последняя лекция при дистанционном формате (одномерные массивы)

Основные этапы выполнения работы:
1) понимание математической сущности задания по перебору элементов одномерного массива в зависимости от индивидуальной формулировки задания
2) подготовка формата представления числовых данных **в файле для ввода** (2 способа: с указанием количества значений для чтения и чтение всех имеющихся) – подробно поясняется в отчете в разделе формат внешнего хранения данных
3) использование **цикла while при вводе** и **цикла for при выводе** содержания одномерного массива
4) подготовка формата представления ВСЕХ элементов массива **при их выводе** – организация вывода **как на экран, так и в файл**
5) организация последовательной **обработки одномерного массива** с помощью цикла while (или for)
6) подготовка **примеров**

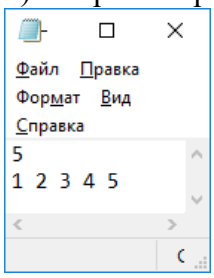
Результаты работы:
- подготовленные входные файлы с числовыми данными в 2 вариантах (с указанием и без указания количества читаемых данных из файла), при чтении сообщения о некорректных ситуациях,
- сформированные программой выходные файлы с контрольным выводом введенных значений (заполненные одномерные массивы) и полученные результаты по итогам обработки одномерных массивов

Требования к выполнению задания:
1) реализовать 1-ю версию с вводом ограниченного числа значений (с контролем размера одномерного массива):
   а) ввод из файла числа вводимых элементов, контроль этого числа на соответствие размера одномерного массива
   б) ввод из файла элементов одномерного массива, контроль исчерпания значений по достижению конца файла (в отчете пояснения по внутреннему хранению данных)
   в) контрольный вывод в файл введенных значений (заполненные одномерные массивы)
   г) выполнение обработки одномерного массива (в отчете пояснения по алгоритму)
   д) вывод в файл результата обработки одномерного массива
   е) использование индексного обращения к элементам массива - A[i]
2) реализовать 2-ю версию с вводом всех имеющихся значений:
   а) проход по файлу для подсчета размещенных в нем числовых значений, выделение динамической памяти для одномерного массива по фактическому количеству числовых значений в файле (в отчете пояснения по внутреннему хранению данных)
   б) ввод из файла элементов одномерного массива
   в) контрольный вывод в файл введенных значений (заполненные одномерные массивы)
   г) выполнение обработки одномерного массива (в отчете пояснения по алгоритму)
   д) вывод в файл результата обработки одномерного массива
   е) освобождение занятой динамической памяти
   ж) использование обращения к элементам массива только с помощью указателя - *(p+i)
3) подготовленный набор данных (скорее всего несколько файлов), которые должны продемонстрировать **реакцию программы на разные входные данные**:
   - отсутствие подходящего согласно заданию
   - подходят все согласно заданию
   - подходит только часть согласно заданию
   - имеются некорректные данные (например, задан отрицательный размер массива)

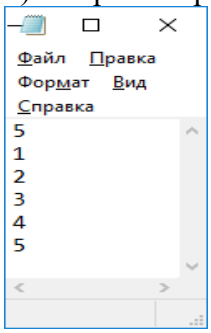
Если по формулировке индивидуального задания используется не один, а два и более одномерных массивов, то для каждого из них осуществляется подготовка отдельных файлов.

Краткие сведения по работе с файлами:
1) работа с файлами организована с использованием файловых потоков ввода/вывода, использование которых возможно при подключении сведений по операциям при вводе/выводе:
#include <fstream.h>
2) для обращения к файлу необходима переменная специального типа:
fstream f;
3) для работы с данными в файле он должен быть открыт в следующих режимах:
- либо для записи:
```
  f.open("results.txt", ios::out); // первым в скобках указано фактическое имя файла для записи
                                   // файл будет помещен в папку с вызванной программой
```
- либо для чтения:
```
  f.open("input.txt", ios::in); // первым в скобках указано фактическое имя файла для записи
                                // файл должен находится в папке, откуда вызывается программа
```
4) при попытке открытия файла могут возникнуть ошибки операции: при чтении нет в папке указанного по имени файла, при записи недостаточно места на устройстве хранения и др. Поэтому необходимо проверить правильность выполнения операции:
```
  if(f.is_open()==0) cout<< "ошибка при открытии файла";
```
5) открытие файла на запись позволяет записывать данные (числовые значения и символьные последовательности):
```
f << x << "запись в файл";
```
6) открытие файла на чтение позволяет читать данные с размещением их в переменных подходящего типа:
```
f >> x;
```
7) в файле находится ограниченное число значений и при чтении они могут закончится (достигнут конец файла), для проверки этого используется специальная операция:
```
if(f.eof()) cout << "достигнут конец файла";
```
Имеется существенная особенность возникновения признака конца файла – возникает не после чтения последнего значения, а при попытке чтения не существующего значения.
Поэтому рекомендуется использовать подобную схему чтения значений до конца файла:
```
f.open("input.txt", ios::in);
if(!f.is_open()==0)
{while(1)
   { f >> x;  if(f.eof()) break; }
}
```
8) если файл больше не нужен, то его надо закрыть:
```
f.close();
```

Возможные форматы оформления исходных данных при реализации 1-го варианта:
1) в первой строке указано количество вводимых значений, во второй строке – сами значения:

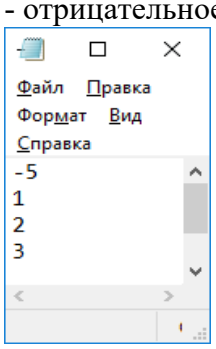


так вроде нагляднее при просмотре строки (задано 5 значений)
2) в первой строке указано количество вводимых значений, в следующих строках – сами значения:

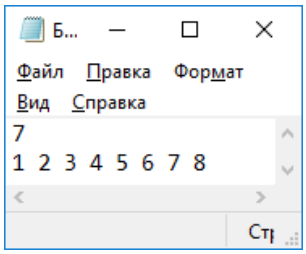


так проще находить по номеру строки необходимое по номеру значение (их тоже 5)

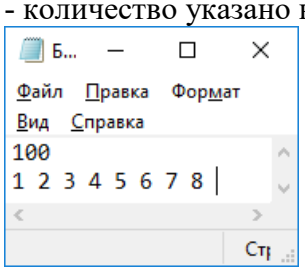
3) естественно в первой строке могут быть неправильные значения, на которые программа должна реагировать в зависимости от ситуации:
- отрицательное количество значений:



Самое простое заменить на 0, т.е. обработке подлежит это количество значений
- количество значений превышает объявленный размер массива (пусть A[4]):

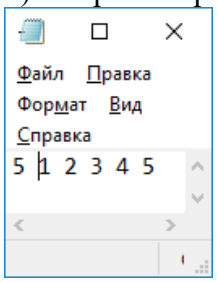


Самое простое заменить 7 на 4, т.е. согласовать ввод с размером массива
- количество указано корректное, но в файле значений меньше:

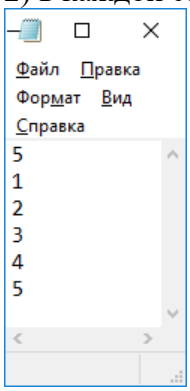


Самое простое при достижении конца файла заменить на фактическое число считанных из файла значений
- могут встретится и иные ситуации

Возможные форматы оформления исходных данных при реализации 2-го варианта:
1) в первой строке указаны все обрабатываемые значения:



так вроде нагляднее при просмотре строки (но тяжело отсчитать)
2) в каждой строке размещено очередное обрабатываемое значение:



так проще находить по номеру строки необходимое по номеру значение

Может возникнут иные разумные мысли по размещению значений в файле для удобства их чтения – это допускается.
Но описание формата файла ОБЯЗАТЕЛЬНО в любом случае.

При реализации программы 1-го варианта используем следующую схему реализации программного кода:

```
const unsigned N=10; // максимальное число элементов в массиве
unsigned a;              // реальное число элементов в массиве
unsigned i;              // индексная переменная
float W[N], tmp; // обращение через W[i]
fstream f;
f.open("inp.txt",ios::in);
 if( f.bad()!=0 )
  { cout<<" Ошибка при открытии файла для ввода !! "<<endl; }
  else
  { f >> a;
    if(a<0) {a=0; cout<<" Некорректный размер скорректирован в 0";} // далее аналогично
    if(a>N) a=N;
    i=0;
    while(1)
      {f>>tmp; if(f.eof()) break; // значения в файле исчерпаны
       W[i]=tmp;
        i++; if(i>=N) break; // не хватает места в массиве
      }
    a=i; // далее будем обрабатывать фактическое количество считанных элементов
    f.close();
    f.open("out.txt",ios::out);
    if( f.bad()!=0 )
     { cout<<" Ошибка при открытии файла для вывода !! "<<endl; }
      else
      { for(i=0; i<a; i++) { f<< W[i]; } // контрольный вывод введенных значений в массив
        for(i=0; i<a; i++) { /* обработка по индивидуальному заданию */}
        for(i=0; i<a; i++) { f<< W[i]; } // вывод результата (размер массива мог быть скорректирован)
      f.close();
     }
  }
```

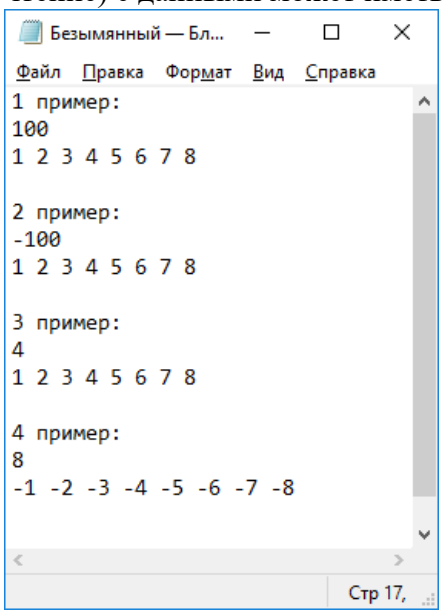
При реализации программы 2-го варианта существенно изменится схема чтения из файла:

```
unsigned a;              // реальное число элементов в массиве
unsigned i;              // индексная переменная
float* p, tmp; // обращение ТОЛЬКО через *(p+i)
fstream f;
f.open("inp.txt",ios::in);
 if( f.bad()!=0 )
  { cout<<" Ошибка при открытии файла для ввода !! "<<endl; }
  else
  { a=0;       // сначала сосчитаем число значений в файле
    while(1)
      {f>>tmp; if(f.eof()) break; // значения в файле исчерпаны
        a++;  }
    f.close();
   p=new float[a]; // будем обрабатывать фактическое количество считанных элементов
   if(p!=NULL)
    {f.open("inp.txt",ios::in);
    a=0;       // теперь заполним динамический массив
    while(1)
      {f>>tmp; if(f.eof()) break; // значения в файле исчерпаны
        *(p+a)=tmp; a++;  }
    f.close();
```

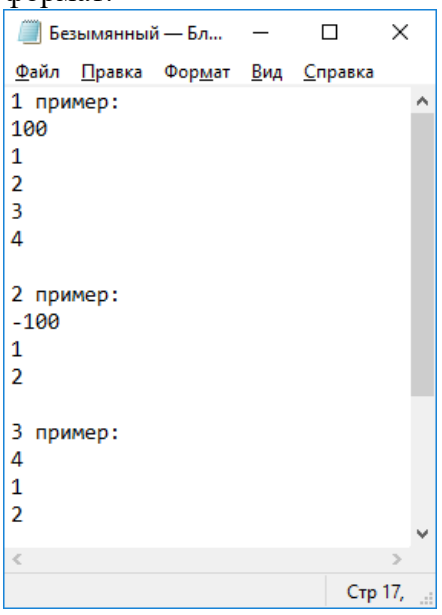
```
  f.open("out.txt",ios::out);
  if( f.bad()!=0 )
   { cout<<" Ошибка при открытии файла для вывода !! "<<endl; }
    else
    { for(i=0; i<a; i++) { f<<*(p+i); } // контрольный вывод введенных значений в массив
       // и далее аналогично 1-му варианту …
    f.close();
   }
   delete [] p;
  }
}
```

К загрузке на vec.etu.ru следует подготовить 3 файла:
- привычный отчет
- привычный файл с текстом программы
- файл со ВСЕМИ наборами входных значений для проверки всех обрабатываемых ситуаций

Прикрепляемый файл (его имя должно соответствовать используемому в программе при открытии на чтение) с данными может иметь такой формат:



ИЛИ



Естественно запуск программы на проверку будет по извлеченной из этого общего файла части строк согласно представленному в отчете формату внешнего хранения данных.

Желающие могут адаптировать программу на обработку этого единого файла, т.е. программа должна загружать данные из заданного формата внешнего хранения частями и сразу их обрабатывать, переходя затем к следующей порции (строке файла) до исчерпания единого файла.
Это очень сложно в обработке и не все пояснения по работе с файлом были предоставлены.
Поэтому только по личной инициативе сдающего – если не будет получаться в обозримом временном интервале, то общий файл со всеми примерами только для загрузки.