Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**«Вычисление арифметических выражений»**

**Выполнил**:

студентка группы 382003-1

Огородникова Валерия Сергеевна

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2020

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 5](#_Toc26962564)

[Описание программной реализации 6](#_Toc26962565)

[Подтверждение корректности 7](#_Toc26962566)

[Результаты экспериментов 8](#_Toc26962567)

[Заключение 9](#_Toc26962568)

[Приложение 10](#_Toc26962569)

# Постановка задачи

Написать программу, с помощью которой можно работать с массивом, элементами являются числа с плавающей запятой(float). Реализовать меню программы, работу с массивом(изменение количества элементов, рандомайзер, ввод с клавиатуры, вывод) и 4 сортировки.

# Метод решения

* Сортировка Шелла:

Идея **сортировки** состоит в том, чтобы сортировать элементы отстоящие друг от друга на некотором расстоянии step. Затем сортировка повторяется при меньших значениях step, и в конце процесс сортировки Шелла завершается при step = 1 (а именно обычной сортировкой вставками).

* Сортировка вставками:

Сортировка вставками использует такой инвариант: первые элементы списка, то есть срез A[:i] уже отсортирован. А вот как устроен алгоритм добавления i-го элемента к уже отсортированной части. Здесь берется элемент A[i] и добавляется к уже отсортированной части списка. Например, пусть i = 5 и срез A[:i] = [1, 4, 6, 8, 8], а значение A[i] == 5. Тогда элемент A[i] == 5 нужно поставить после элемента A[1] == 4, а все элементы, которые больше 5, сдвинуть вправо на 1. Получится cрез A[:i + 1] = [1, 4, 5, 6, 8, 8]. Таким образом, при вставке элемента A[i] в срез A[:i] так, чтобы в результате получился упорядоченный срез, все элементы, которые больше A[i] будут двигаться вправо на одну позицию. А в освободившуюся позицию и будет вставлен элемент A[i].  
При этом значение A[i] нужно сохранить в переменной, т. к. на место элемента A[i], возможно, будет записан элемент A[i – 1].

* Сортировка слиянием:

Алгоритм сортировки слиянием основан на идее, что два отсортированных списка можно слить в один отсортированный список за время, равное суммарной длине этих списков.

Для этого сравним первые элементы данных списков. Тот элемент, который меньше, скопируем в конец результирующего списка (который первоначально пуст) и в этом списке перейдем к следующему элементу. Будем повторять этот процесс (выбираем из начала двух списков наименьший элемент, копируем его в результирующий список), пока один из исходных списков не кончится. После этого оставшиеся элементы (один из двух исходных списков будет непуст) скопируем в результирующий список.

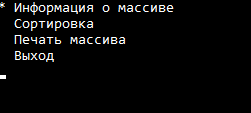
* Сортировка Хоара:

1. Выбираем в массиве некоторый элемент, который будем называть *опорным элементом*.  
2. Операция разделения массива: реорганизуем массив таким образом, чтобы все элементы, меньшие или равные опорному элементу, оказались слева от него, а все элементы, большие опорного — справа от него. Обычный алгоритм этой операции:

* Два индекса — l и r, приравниваются к минимальному и максимальному индексу разделяемого массива соответственно.
* Вычисляется индекс опорного элемента m.
* Индекс l последовательно увеличивается до m до тех пор, пока l-й элемент не превысит опорный.
* Индекс r последовательно уменьшается до m до тех пор, пока r-й элемент не окажется меньше либо равен опорному.
* Если r = l — найдена середина массива — операция разделения закончена, оба индекса указывают на опорный элемент.
* Если l < r — найденную пару элементов нужно обменять местами и продолжить операцию разделения с тех значений l и r, которые были достигнуты. Следует учесть, что если какая-либо граница (l или r) дошла до опорного элемента, то при обмене значение m изменяется на r-й или l-й элемент соответственно.

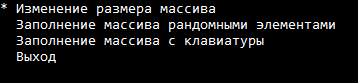
3. Рекурсивно упорядочиваем подмассивы, лежащие слева и справа от опорного элемента.  
4. Базой рекурсии являются наборы, состоящие из одного или двух элементов. Первый возвращается в исходном виде, во втором, при необходимости, сортировка сводится к перестановке двух элементов. Все такие отрезки уже упорядочены в процессе разделения.

# Руководство пользователя

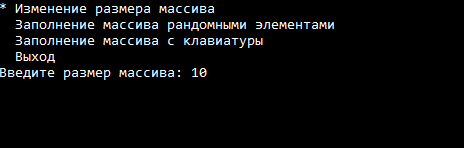


Управление осуществляется с помощью стрелочек вверх, вниз и клавиши “Enter” (в случае нажатия на другие клавиши ничего не будет происходить).

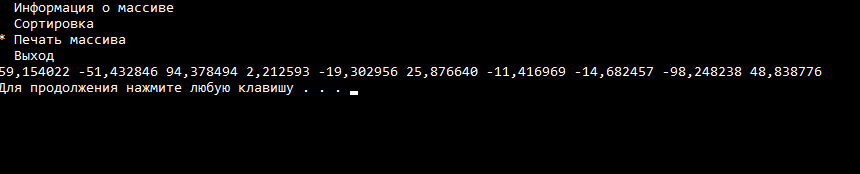
Например, чтобы изменить размер массива и заполнить его случайными числами, нужно перейти в первый пункт меню - “Информация о массиве”.

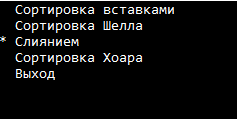


После перехода в следующий пункт меню нажать клавишу “Enter” и ввести размер массива и нажать “Enter” (допустим мы хотим массив из 10 элементов).

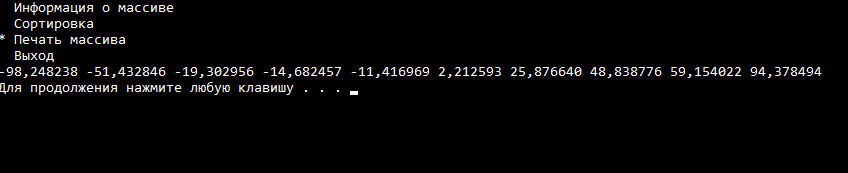


Следующим шагом нужно заполнить этот массив. У вас есть выбор с помощью чего заполнить массив: с помощью рандомайзера(от -100, до 100), с помощью клавиатуры(без ограничений).

Следующим шагом вы можете вывести массив. Чтобы это сделать нужно выйти из этого пункта меню (с помощью кнопки ”esc” или с помощью пункта меню “Выход”).

Далее, у вас есть возможность отсортировать полученный массив. Для этого нужно перейти в пункт меню “Сортировки”. На выбор даётся 4 сортировки. 

Вы можете перейти в главное меню и вывести отсортированный массив.



Чтобы выйти из программы, нужно в главном меню выбрать пункт “Выход” или нажать кнопку “esc”.

# Описание программной реализации

В программе содержится 3 заголовочных файла с прототипами функций:

* ArrayF.h – в нём написаны прототипы функций для работы с массивом.
* ArraySort.h – прототипы сортировок массива.
* Menus.h – прототипы главного и вложенных меню.

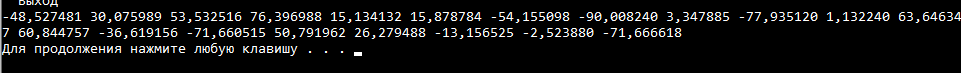
Программа состоит из 4 файлов:

* Main.cpp – в этом файле содержится основная функция main.
* Menu.cpp – реализация меню, определение функций из заголовочного файла Menus.h.
* ArraySort.cpp – определение сортировок из заголовочного файла ArraySort.h.
* ArrayFunc.cpp – определение функций работы с массивом из заголовка ArrayF.h.

# Подтверждение корректности

Для подтверждения корректности в программе была выполнена отладка. Примеры работоспособности программы с разными входными данными:

1. Количество элементов равно 20, элементы сгенерированы рандомно, использовалась сортировка Шелла:

До:

Screenshot 1.1

После: 

Screenshot 1.2

Результат: Программа отработала корректно, свою задачу выполнила.

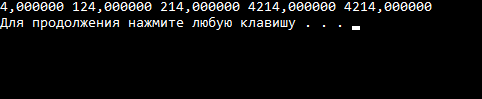
1. Количество элементов равно 5, элементы заполнялись с клавиатуры, использовалась сортировка слиянием:

До:



Screenshot 2.1

После:



Screenshot 2.2

Результат: Программа отработала корректно, свою задачу выполнила.

# Результаты экспериментов

По данным экспериментов видно, что программа работает корректно (см screenshot 1 & 2)

# Заключение

Программа выполнена, отладка проведена, тестирование прошло успешно. Готова к сдаче.

# Приложение

Одна из важнейших частей кода: реализация меню сортировок и сами сортировки.

Меню сортировок:

void menu\_sort(float\* arr, int n)

{

char c;

char men[5][50] = { {"Сортировка вставками"}, {"Сортировка Шелла"}, {"Слиянием"}, {"Сортировка Хоара"}, {"Выход"} };

const int max\_op = 5;

int op = 0;

printf("Размер массива: %d\n", n);

while (true)

{

system("cls");

for (int i = 0; i < max\_op; i++)

{

if (i == op)

{

printf("\* ");

}

else

printf(" ");

printf("%s\n", men[i]);

}

c = \_getch();

if (c == 72)

{

op = (op - 1 + max\_op) % max\_op;

}

if (c == 80)

{

op = (op + 1) % max\_op;

}

if (c == 13)

{

switch (op)

{

case 0:

insert\_sort(arr, n);

break;

case 1:

shell\_sort(arr, n);

break;

case 2:

merge\_sort(arr, 0, n-1);

break;

case 3:

hoar\_sort(arr, 0, n - 1);

break;

case 4:

c = 27;

break;

default:

break;

}

}

if (c == 27)

{

break;

}

}

}