Федеральное государственное образовательное бюджетное учреждение

высшего профессионального образования

«Нижегородский Государственный Университет им.

Н.И.Лобачевского» (ННГУ)

Институт Информационных Технологий Математики и Механики

Отчёт по лабораторной работе

Упорядочивание массивов. Сравнение сортировок.

Выполнил:

студент группы 3821Б1ФИ3

Лупша Е.А

Проверил:

заведующий лабораторией суперкомпьютерных технологий и высокопроизводительных вычислений

Лебедев И.Г

Нижний Новгород

2021г.

# Содержание

[Содержание 1](#_Toc88648576)

[Введение 2](#_Toc88648577)

1. [Постановка задачи 3](#_Toc88648578)

2. [Руководство пользователя 4](#_Toc88648579)

[Первая программа 4](#_Toc88648580)

[Вторая программа 6](#_Toc88648581)

3. [Руководство программиста 9](#_Toc88648582)

[*Описание структуры кода программ* 9](#_Toc88648583)

[Первая программа 9](#_Toc88648584)

[Вторая программа 1](#_Toc88648585)0

[Описание структур данных 1](#_Toc88648586)2

[Описание алгоритмов 1](#_Toc88648587)2

[Сортировка «пузырьком» 1](#_Toc88648588)2

[Сортировка «вставками»](#_Toc88648589) 14

[Быстрая сортировка 16](#_Toc88648590)

4. [Эксперименты](#_Toc88648591) 18

5. [Заключение 2](#_Toc88648592)0

6. [Список литературы 2](#_Toc88648593)1

7. [Приложение 2](#_Toc88648594)2

[Приложение 1.1 22](#_Toc88648595)

[Приложение 1.2 25](#_Toc88648596)

# Введение

Си — это универсальный язык программирования с компактным способом записи выражений, современными механизмами управления структурами данных и богатым набором операторов. Си не является ни языком "очень высокого уровня", ни "большим" языком, не рассчитан он и на какую-то конкретную область применения. Однако благодаря широким возможностям и универсальности для решения многих задач он удобнее и эффективнее, чем предположительно более мощные языки. Первоначально Си был создан Деннисом Ритчи как инструмент написания операционной системы UNIX для машины PDP-11 и реализован в рамках этой операционной системы. И операционная система, и Си компилятор, и, по существу, все прикладные программы системы UNIX написаны на Си. Фирменные Си-компиляторы существуют и на нескольких машинах других типов, среди которых IBM/370, Honeywell 6000 и Interdata 8/32. Си не привязан к конкретной аппаратуре или системе, однако на нем легко писать программы, которые без каких-либо изменений переносятся на другие машины, где осуществляется его поддержка.

Алгоритм сортировки — это алгоритм для упорядочивания элементов в массиве. В случае, когда элемент в массиве имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

# Постановка задачи

Реализовать сортировки массивов данных (тип данных char) задаваемых: обязательно случайно, дополнительно с клавиатуры или из файла.

Реализовать сортировки: пузырьком, вставкой, быстрая. Сравнить время работы, сделать выводы.

Первая программа создает текстовый файл с записанными в него числами. Программа принимает количество чисел n, максимальное и минимальное значение.

Вторая программа читает текстовый файл с набором чисел, выводит консольный интерфейс (печать, сортировка, сброс, выход), выполняет выбранные действия.

# Руководство пользователя

## Первая программа

После запуска программы необходимо выбрать способ, которым будет сгенерирован массив. Пользователю на выбор дано 3 варианта:

1. Массив из случайных чисел
2. Ввод массива вручную
3. Считывание массива из файла

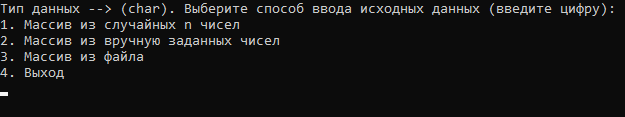


Рис. 1 Программа после запуска

Если пользователь напишет в консоли число 1, то ему необходимо будет ввести размер массива, а также его минимальное и максимальное значение. Затем необходимо ввести название текстового документа (без расширения), под которым будет сохранен файл, который будет заполнен случайными числами, с заданными параметрами (см. рис. 2).

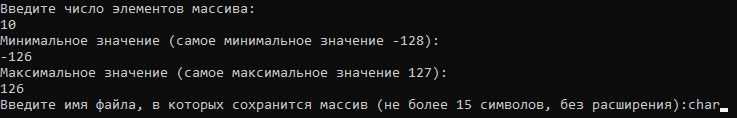


Рис. 2 Генерация массива из случайных чисел.

Если же пользователь напишет в консоли число 2, то после этого указатель в консоли переместиться на новую строку и пользователь сможет начать ввод чисел с клавиатуры. Для начала необходимо ввести N – количество чисел в файле. Затем необходимо по очереди вводить числа, и как только пользователь введет все нужные числа, необходимо будет ввести название файла, в который сохранится массив. После этого массив сразу же будем записан в файл. Пример такой работы пользователя с интерфейсом программы приведен на рисунке 3.

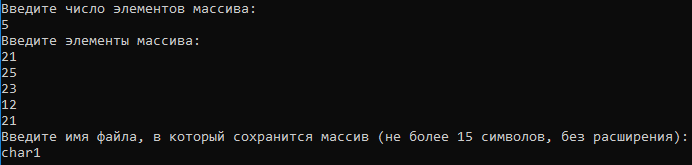


Рис. 3 Ввод массива с клавиатуры.

Если пользователь напишет в консоли число 3, то массив будет скопирован из файла с названием arr.txt и передан в другой - новый файл, название которого пользователь должен будет ввести сам. (см. рис. 4).



Рис. 4 Копирование массива из файла

При любом способе ввода результатом работы программы будет текстовый файл, который наполнен некоторым количеством чисел. Например, для набора из 10 случайных чисел, сгенерированных в диапазоне от 25 до 30, файл будет выглядеть как показано на рисунке 5.



Рис. 5 Текстовый файл с сгенерированными числами

## Вторая программа

При запуске второй программы пользователю будет предложено ввести путь к файлу, в котором расположен массив и который необходимо отсортировать. Как только путь будет указан, пользователю будет предложен консольный интерфейс и возможность выбора: 1. Печать; 2. Сортировка; 3. Сброс; 4. Вычисление норм вектора; 5. Нормировка; 6. Выход;

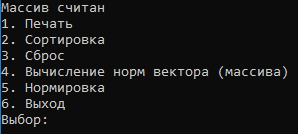


Рис.6 Интерфейс программы

Если пользователь напишет в консоли число 1, то программа выведет массив (см. рис.7).

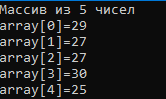


Рис. 7 Печать массива

Если пользователь напишет в консоли число 2, то программа предложит выбрать следующие сортировки: пузырьком, вставками и быстрая. Пользователю необходимо выбрать нужную сортировку (см. рис. 8).

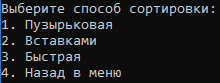


Рис. 8 Выбор сортировки

Например, пользователю нужно выбрать сортировку пузырьком. Тогда необходимо ввести число, соответствующее данной сортировке. Затем на экране будет выведено время данной сортировки (см. рис. 9)



Рис. 9 время сортировки

Затем будет выведено начальное меню и при желании пользователя можно будет убедиться в том, что массив отсортирован, выведя массив (нажать число 1) (см. рис. 10).

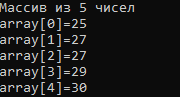


Рис. 10 Числа из файла после сортировки

При вводе команды сброс (число 3) числа из файла вернуться в те позиции, на которых они были при запуске программы (см. рис. 11).

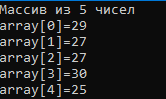


Рис.11 Работа команды сброса

При вводе числа 4 программа подсчитает нормы вектора (массива):

1. Первую норму
2. Вторую норму
3. Гельдерову норму (для расчета этой нормы необходимо ввести степень p)
4. Бесконечную норму

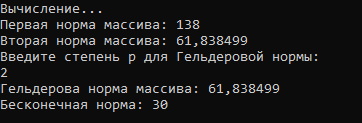
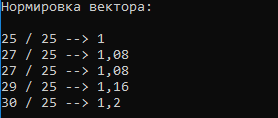


Рис. 12 Подсчет норм

При вводе числа 5 программа совершит нормировку вектора (массива).

 Рис. 13 Нормировка вектора

Для завершения работы программы пользователю достаточно ввести число 6, оно осуществляет выход из программы.

# Руководство программиста

# *Описание структуры кода программ*

# *Первая программа*

1. Подключение библиотек, с которыми предстоит работать.

|  |
| --- |
| #include <stdio.h>  #include <limits.h>  #include <stdlib.h>  #include <time.h>  #include <locale.h>  #include <string.h> |

1. Объявление функции для очищения интерфейса.

|  |
| --- |
| void clear()  {  system("@cls||clear");  } |

1. Объявление функции СreateArr.

Функция для создания массива, заполненного случайными числами.

|  |
| --- |
| void CreateArr(int scan\_count, char\* array, int max, int min) |

1. Объявление функции write\_in\_file.  
   Функция для записи массивов в файл.

|  |
| --- |
| void write\_in\_file(int scan\_count, char\* array, char\* name) |

1. Объявление функции имени файла.   
   Функция совмещает название файла с расширением .txt

|  |
| --- |
| void filename(char\* name) |

1. Объявление функции scanarr().

Функция считывает из файла массив

|  |
| --- |
| void scanarr(int scan\_count, char\* array) |

1. Объявление функции scancount().

Функция считывает количество чисел в файле, чтобы можно было завести динамический массив.

|  |
| --- |
| int scancount() |

1. Объявление функции main().

Объявление переменных, консольный интерфейс и вызов функций в нужный момент.

|  |
| --- |
| int main() |

### Вторая программа

1. Подключение библиотек, с которыми предстоит работать.

|  |
| --- |
| #include <stdio.h>  #include <limits.h>  #include <stdlib.h>  #include <time.h>  #include <locale.h>  #include <string.h>  #include <conio.h>  #include <math.h> |

1. Объявление функции clrscr() для очистки интерфейса.

|  |
| --- |
| void clrscr() |

1. Функция bubbleSort() для сортировки пузырьком.

|  |
| --- |
| void bubbleSort(char\* arr, int count) |

1. Функция qSort() для быстрой сортировки.

|  |
| --- |
| void qSort(char\* arr, int first, int last) |

1. Функция inSort() для сортировки вставками.

|  |
| --- |
| void inSort(char\* arr, int count) |

1. Объявление функции scancount() для считывания количества чисел в массиве.

|  |
| --- |
| int scancount(char\* name) |

1. Функция PrintArray() для печати массива.

|  |
| --- |
| void PrintArray(char\* arr, int count) |

1. Функция resultnorm1() для подсчета первой нормы массива.

|  |
| --- |
| int resultnorm1(char\* arr, int count) |

1. Функция resultnorm2() для подсчета второй нормы массива.

|  |
| --- |
| double resultnorm2(char\* arr, int count) |

1. Функция resultnormG() для подсчета Гельдеровой нормы массива.

|  |
| --- |
| double resultnormG(char\* arr, int count, double p) |

1. Функция infinitynorma() для подсчета бесконечной нормы массива.

|  |
| --- |
| int infinitynorma(char\* arr, int count) |

1. Функция normirovka() для нормировки массива.

|  |
| --- |
| void normirovka(char\* arr, int count) |

1. Функция main(). Объявление всех переменных. Консольный интерфейс, вызов функций в нужное время.

|  |
| --- |
| int main() |

## 

## Описание структур данных

Массив данных располагается в файле с расширением .txt. В первой программе создается динамический массив для хранения, массив из символов char для хранения имена файла. Во второй программе для сортировки создается 2 динамических массива, один из которых сортируется, а второй используется для сброса.

При написании программ были использованы стандартные библиотеки языка «C»: stdlib.h, stdio.h, time.h, limits.h, math.h, locale.h, string.h, conio.h

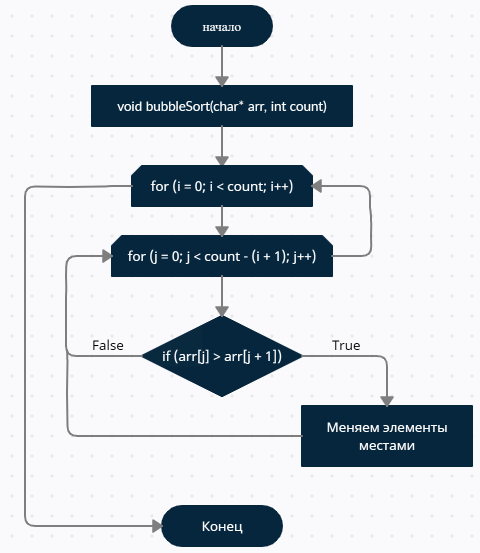
## Описание алгоритмов

### Сортировка «пузырьком»

Упорядоченный массив создается на том же участке памяти, где находится исходная последовательность. Идея метода состоит в том, чтобы попарно сравнивать соседние элементы. Каждый проход начинается с начала последовательности. Сравнивается первый элемент со вторым: если порядок между ними нарушен, то они меняются местами. Затем сравниваются второй с третьим, третий с четвертым и так далее до конца массива; элементы с неправильным порядком в паре меняются местами. В итоге, после первого прохода, максимальный (или минимальный, в зависимости от вида сортировки: по возрастанию/по убыванию) элемент будет находится на последнем месте в массиве, он как бы “всплывет” наверх. Именно поэтому этот метод называется **сортировка пузырьком**. На следующем проходе рассматривается последовательность от 1 до N-1, затем от 1 до N-2, и так до конца. После каждого прохода можно делать проверку: выполнялись ли перестановки элементов. Если не выполнялись, то сортировка завершена.

|  |
| --- |
| void bubbleSort(char\* arr, int count)  {  int i = 0, j = 0;  int tmp = 0;  for (i = 0; i < count; i++)  {  for (j = 0; j < count - (i + 1); j++)  {  if (arr[j] > arr[j + 1])  {  tmp = arr[j];  arr[j] = arr[j + 1];  arr[j + 1] = tmp;  }  }  }  } |

Сортировка Bubble Sort считается наиболее простой и в тоже время наиболее время затратной. Её эффективность оценивается как от размера входных данных, то есть количества элементов в исходном массиве. И в самом деле это можно легко заметить, ведь внешний цикл повторяется n раз и в каждом из таких циклов есть вложенный цикл, который повторяется n раз, отсюда можно сделать вывод, что всего совершается проверок.

 Рис. 14 Алгоритм сортировки «пузырьком»

### Сортировка «вставками»

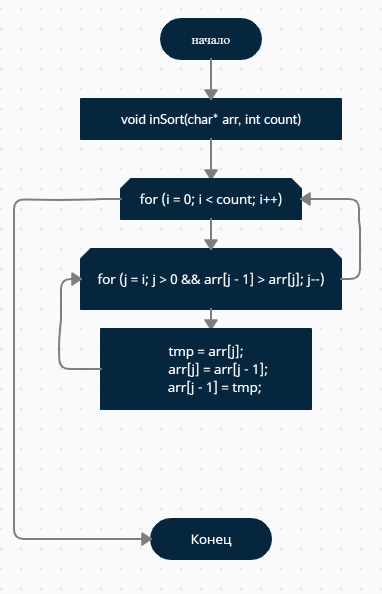
На вход алгоритма подается число элементов в массиве и указатель в начало массива. Главная идея метода состоит в том, что при добавлении нового элемента в уже отсортированный массив его необходимо вставлять на нужное место, вместо того, что бы ставить его в произвольное, и потом заново сортировать всю последовательность.

Рассмотрим алгоритм метода **сортировки вставками** на примере сортировки по возрастанию. Первый элемент в массиве образует уже отсортированную последовательность. Сравниваем второй элемент с первым. Если порядок между ними нарушен, то первый элемент передвигается на одну позицию вправо. Теперь отсортированный массив состоит из двух элементов.

Далее, в течении каждой итерации, берем следующий элемент (третий, четвертый и т.д) и сравниваем его поочередно с другими элементами в уже отсортированном списке, **начиная с конца** этого списка. Если порядок между сравниваемыми элементами нарушен, то меняем их местами, если нет, то “вставка” нового элемента закончена, переходим к следующему.

|  |
| --- |
| void inSort(char\* arr, int count)  {  int i = 0;  int j = 0;  int tmp = 0;  for (i = 0; i < count; i++)  {  for (j = i; j > 0 && arr[j - 1] > arr[j]; j--)  {  tmp = arr[j];  arr[j] = arr[j - 1];  arr[j - 1] = tmp;  }  }  } |

В худшем случае, как и в среднем, сортировка вставками затрачивает времени на свою работу. Данный алгоритм по совей сложности схож с сортировкой пузырьком, ведь в нем производятся схожие по структуре процессы: сравнение элементов и их сдвиги. Однако на практике такой алгоритм нередко оказывается быстрее сортировки пузырьком, так как в ней вне зависимости от того какой был исходный массив выполняется n проверок для каждого элемента с первого до( n – 1)-ого, в то время как сортировки вставкой выполняют проверки для новых элементов до тех пор, пока они не займут своего места и в изначально частично упорядоченном массиве могу работать значительно быстрее.

 Рис.15 Алгоритм сортировки вставками

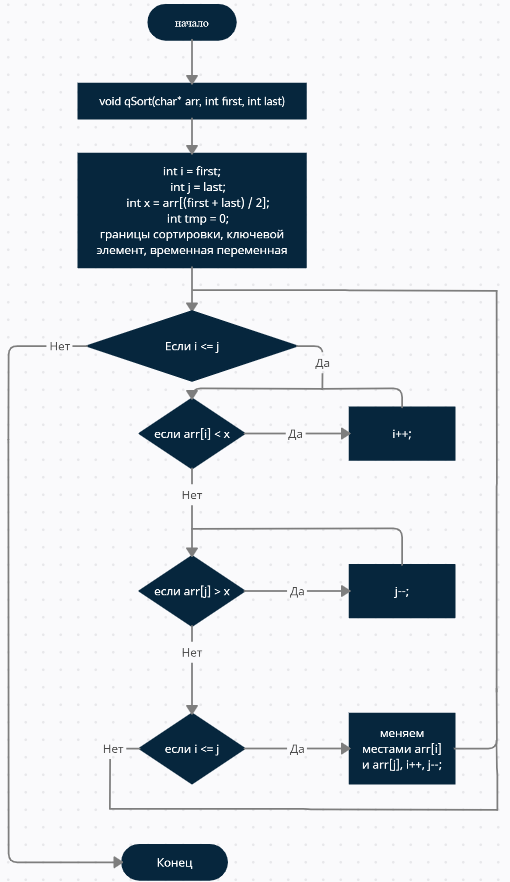
### 

### Быстрая сортировка

Как и в других случаях, на вход этому алгоритму подается число элементов и указатель на начало массива. Однако в отличии от двух других сортировок он работает по совершенно другому принципу. Для начала выбирается средний по положению элемент входного массива – ключ. Затем просматриваются элементы массива до ключа и после него. Как только слева находится элемент больше ключа, и справа находится элемент меньше ключа, то они меняются местами. Это происходит до тек пор, пока все элементы слева от ключа не будут меньше него, а все справа не станут больше. Таким образом мы имеем два массива: массив с числами меньше ключа и массив с числами больше ключа. Затем происходит рекурсивный вызов сортировки для каждого из этих массивов. Этот процесс происходит до тех пор, пока все элементы массива не окажутся отсортированными, то есть пока один из массивов на некоторой итерации не станет одноэлементным.

|  |
| --- |
| void qSort(char\* arr, int first, int last)  {  int i = first;  int j = last;  int x = arr[(first + last) / 2];  int tmp = 0;  while (i <= j)  {  while (arr[i] < x)  i++;  while (arr[j] > x)  j--;  if (i <= j)  {  tmp = arr[i];  arr[i] = arr[j];  arr[j] = tmp;  i++;  j--;  }  }  if (first < j)  qSort(arr, first, j);  if (last > i)  qSort(arr, i, last);  } |

Данный алгоритм принципиально отличается от двух других. Это касается и его сложности. В сравнении с квадратичными сортировками (пузырьковая, вставками), быстрая сортировка более чем оправдывает свое название, ведь её худшее время работы , в то время как среднее время работы составляет , что быстрее чем в случае с квадратичными сортировками.

 Рис 16. Алгоритм быстрой сортировки

Формула р-нормы (Гельдерова норма) вектора:

(1)

где x – исходный массив, i – индекс вектора (номер элемента вектора), n – количество элементов вектора.

Норму при p = 1 называют *норма-сумма*, при р = 2 будет вторая норма (евклидова норма) вектора. А при p → ∞ будет формула:xi

(2)

Формула вычисления нормировки вектора (каждый элемент вектора разделим на результат данной формулы):

(3)

# (4)

# Эксперименты

В качестве эксперимента посмотрим на практике какой из алгоритмов сортировки работает быстрее с набором случайной сгенерированных чисел.

Для этого с помощью первой программы создадим текстовый файл с большим количеством чисел (см. рис. 17).

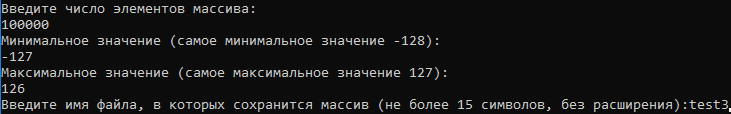


Рис. 17 Параметры генерации чисел

Затем запустим программу два и с её помощью отсортируем элементы в текстовом файле с помощью сортировки пузырьком (см. рис. 18)



Рис. 18 Запуск и завершение сортировки

Сортировка пузырьком продлилась примерно 24 секунды. После её завершения возвращаем файл к первоначальному виду и сортируем его с помощью сортировки вставкой. Затем повторяем это процедуру для быстрой сортировки.

Полученные в экспериментах данные приведены в таблице 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **Размер входных данных** | **Пузырьком**  **время работы, c.** | **Вставками**  **время работы, c.** | **Быстрая**  )  **время работы, c.** |
| 25000 | 1,25 | 0,72 | 0,002 |
| 100000 | 24 | 11,5 | 0,006 |
| 500000 | 601 | 289 | 0,034 |
| 1000000 | N/A | N/A | 0,066 |

Таблица 1. Сравнение времени работы сортировок

Все эти данные были получены на одной и той же машине, при одном и том же диапазоне генерации случайных чисел.

# Заключение

В ходе данной лабораторной работы, на языке программирования «С» были написаны две программы, позволяющие работать с текстовыми файлами: заполнять их числами и получать числа из них. Реализованы три алгоритма сортировки числовых массивов: сортировка пузырьком, вставками и быстрая, а также был проведен эксперимент по сравнению их быстродействия.

По результатам экспериментов можно сделать вывод:  
**Пузырьковая сортировка:**  
Преимущества:  
Простая реализация  
Быстро работает для почти отсортированных массивов  
Недостатки:  
Медленно работает в остальных случаях  
**Сортировка вставками:**Преимущества:  
Алгоритм эффективен при работе с массивами  
Хорошо справляется с массивами небольшого размера  
Недостатки:  
Медленно работает в остальных случаях  
**Быстрая сортировка:**Преимущества:  
Очень эффективный алгоритм для сортировки больших массивов данных.  
Недостатки:  
Сложен в реализации  
Малоэффективен на массивах состоящих из небольшого числа элементов.

# Список литературы

1. Лекции Сысоева Александра Владимировича, 2021.
2. Практики и лабораторные Лебедева Ильи Геннадьевича, 2021.
3. Брайан Керниган, Деннис Ритчи. Язык программирования C. — Москва: Вильямс, 2015. — 304 с.
4. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ — 2-е. — М.: Вильямс, 2005. — 1296 с.
5. Кнут Д. Э. Искусство программирования. Том 3. Сортировка и поиск / под ред. В. Т. Тертышного (гл. 5) и. В. Красикова (гл. 6). — 2-е изд. — Москва: Вильямс, 2007. — Т. 3. — 832 с.
6. Алгоритмы / С. Дасгупта, Х. Пападимитриу, У. Вазирани; Пер. с англ. под ред. А. Шеня. –– М.: МЦНМО, 2014. –– 320 с.

# Приложение 1

## Приложение 1.1

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include <stdio.h>  #include <limits.h>  #include <stdlib.h>  #include <time.h>  #include <locale.h>  #include <string.h>  const float RAND\_MAX\_F = RAND\_MAX;  void clrscr() {  system("@cls||clear");  }  // Создание случайного массива  void CreateArr(int scan\_count, char\* array, int max, int min)  {  int i = 0;  char element = 0;  srand(time(NULL));  for (i = 0; i < scan\_count; i++)  {  element = ((rand() / RAND\_MAX\_F) \* (max-min) + min);  array[i] = element;  }  }  // Запись массива в файл  void write\_in\_file(int scan\_count, char\* array, char\* name)  {  int i = 0;  FILE\* file = fopen(name, "w");  if (file == NULL)  printf("Ошибка при открытии файла\n");  else  {  for (i = 0; i < scan\_count; i++)  {  fprintf(file, "%d\n", array[i]);  }  }  fclose(file);  }  // Имя файла  void filename(char\* name)  {  while (strlen(name) >= 15) {  printf("Название слишком длинное");  scanf("%s", &name);  }  strcat(name, ".txt");  }  // Считывание массива из файла  void scanarr(int scan\_count, char\* array)  {  int i = 0;  FILE\* file = fopen("arr.txt", "r+");  if (file == NULL)  printf("Ошибка при открытии файла\n");  else  {  for (i = 0; i < scan\_count; i++)  {  fscanf(file, "%d", &array[i]);  }  }  }  int scancount()  {  int count = 0;  FILE\* file = fopen("arr.txt", "r");  if (file == NULL) {  printf("Ошибка при открытии файла\n");  exit;  }  else  {  while (!feof(file))  {  if (fgetc(file) == '\n')  count++;  }  }  fclose(file);  return count;  }  int main()  {  setlocale(LC\_ALL, "Rus");  int scan\_count = 0, flag = 0, i = 0, min = 0, max = 0;  char element = 0;  char\* array = 0;  char\* name[16] = { 0 };  do {  printf("Тип данных --> (char). Выберите способ ввода исходных данных (введите цифру):\n");  printf("1. Массив из случайных n чисел\n");  printf("2. Массив из вручную заданных чисел\n");  printf("3. Массив из файла\n");  printf("4. Выход\n");  do {  scanf\_s("%d", &flag);  if (flag < 1 || flag > 4)  printf("\nНеверно введен номер, введите снова.\n");  } while (flag < 1 || flag > 4);  switch (flag) {  case 1:  clrscr();  printf("Введите число элементов массива:\n");  scanf\_s("%d", &scan\_count);  array = (char\*)malloc(scan\_count \* sizeof(char));  printf("Минимальное значение (самое минимальное значение -128):\n");  scanf("%d", &min);  printf("Максимальное значение (самое максимальное значение 127):\n");  scanf("%d", &max);  CreateArr(scan\_count, array, min, max);  printf("Введите имя файла, в которых сохранится массив (не более 15 символов, без расширения):");  scanf("%s", &name);  filename(name);  write\_in\_file(scan\_count, array, name);  free(array);  clrscr();  break;  case 2:  clrscr();  printf("Введите число элементов массива:\n");  scanf\_s("%d", &scan\_count);  array = (char\*)malloc(scan\_count \* sizeof(char));  printf("Введите элементы массива:\n");  for (i = 0; i < scan\_count; i++)  {  scanf("%d", &element);  if (element >= -128 && element <= 127)  {  array[i] = element;  }  else  {  printf("Введено неверное значение, диапазон [0;255]\n");  i--;  }  }  printf("Введите имя файла, в который сохранится массив (не более 15 символов, без расширения):\n");  scanf("%s", &name);  filename(name);  write\_in\_file(scan\_count, array, name);  free(array);  clrscr();  break;  case 3:  clrscr();  scan\_count = scancount();  array = (char\*)malloc((scan\_count + 1) \* sizeof(char));  printf("Сохраните массив в файл с названием arr.txt\n");  scanarr((scan\_count+1), array);  printf("Введите имя файла, в который сохранить массив (не более 15 символов, без расширения):\n");  scanf("%s", &name);  filename(name);  write\_in\_file((scan\_count+1), array, name);  clrscr();  break;  case 4:  return 0;  default:  printf("Введено неверное значение.\n");  break;  }  } while (flag == 1 || flag == 2 || flag == 3 || flag == 4);  free(array);  } |

## Приложение 1.2

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include <stdio.h>  #include <limits.h>  #include <stdlib.h>  #include <time.h>  #include <locale.h>  #include <string.h>  #include <conio.h>  #include <math.h>  // Очистка консоли  void clrscr() {  system("@cls||clear");  }  // Сортировки массивов  void bubbleSort(char\* arr, int count)  {  int i = 0, j = 0;  int tmp = 0;  for (i = 0; i < count; i++)  {  for (j = 0; j < count - (i + 1); j++)  {  if (arr[j] > arr[j + 1])  {  tmp = arr[j];  arr[j] = arr[j + 1];  arr[j + 1] = tmp;  }  }  }  }  void qSort(char\* arr, int first, int last)  {  int i = first;  int j = last;  int x = arr[(first + last) / 2];  int tmp = 0;  while (i <= j)  {  while (arr[i] < x)  i++;  while (arr[j] > x)  j--;  if (i <= j)  {  tmp = arr[i];  arr[i] = arr[j];  arr[j] = tmp;  i++;  j--;  }  }  if (first < j)  qSort(arr, first, j);  if (last > i)  qSort(arr, i, last);  }  void inSort(char\* arr, int count)  {  int i = 0;  int j = 0;  int tmp = 0;  for (i = 0; i < count; i++)  {  for (j = i; j > 0 && arr[j - 1] > arr[j]; j--)  {  tmp = arr[j];  arr[j] = arr[j - 1];  arr[j - 1] = tmp;  }  }  }  // Считывание размера массива из файла  int scancount(char\* name)  {  int count = 0;  FILE\* file = fopen(name, "r");  if (file == NULL) {  printf("Ошибка при открытии файла\n");  exit;  }  else  {  while (!feof(file))  {  if (fgetc(file) == '\n')  count++;  }  }  fclose(file);  return count;  }  // Печать массива  void PrintArray(char\* arr, int count)  {  int i = 0;  for (i = 0; i < count; i++)  {  printf("array[%d]=%d\n", i, arr[i]);  }  }  // Вычисление норм  int resultnorm1(char\* arr, int count)  {  int i = 0;  int result = 0;  for (i = 0; i < count; i++)  {  result = result + abs(arr[i]);  }  return(result);  }  double resultnorm2(char\* arr, int count)  {  int i = 0;  int sum = 0;  double result = 0;  for (i = 0; i < count; i++)  {  sum = sum + (abs(arr[i])\* abs(arr[i]));  }  result = sqrt(sum);  return(result);  }  double resultnormG(char\* arr, int count, double p)  {  int i;  int sum = 0;  double result = 0;  for (i = 0; i < count; i++)  {  sum = sum + pow(abs(arr[i]), p);    }  result = pow(sum, pow(p, -1));  return(result);  }  int infinitynorma(char\* arr, int count)  {  int i = 0;  int max = -9999;  for (i = 0; i < count; i++)  {  if (abs(arr[i]) > max)  max = abs(arr[i]);  }  return(max);  }  void normirovka(char\* arr, int count)  {  int i = 0;  int sum = 0;  double sqrtb = 0;  for (i = 0; i < count; i++)  {  sum = sum + (abs(arr[i]) \* abs(arr[i]));  sqrtb = sqrt(sum);  printf("Нормировка вектора: \n\n");  for (i = 0; i < count; i++)  {  printf("%d / %g --> %g\n", arr[i], sqrtb, arr[i] \* 1.0 / sqrtb);  }  printf("\n\n\n");  }  }  int main()  {  int flag, sortflag, count, i = 0;  char\* name[300] = { 0 };  double startTime, endTime, p, norm2, normG = 0;  int norm1, normI = 0;  setlocale(LC\_ALL, "Rus");  printf("Введите путь к текстовому файлу с массивом:\n");  scanf("%s", &name);  count = scancount(name);  char\* array = (char\*)malloc(count \* sizeof(char));  char\* array2 = (char\*)malloc(count \* sizeof(char));  if (count == 0 || count == 1) {  printf("Нет смысла сортировать этот массив\n");  return 0;  }  FILE\* file = fopen(name, "r");  if (file == NULL) {  printf("Ошибка при открытии файла\n");  exit;  }  else  {  for (i = 0; i < count; i++)  {  fscanf(file, "%d\n", &array[i]);  }  }  clrscr();  printf("Массив считан\n");  memcpy(array2, array, sizeof(char) \* count);  do {  printf("1. Печать\n2. Сортировка\n3. Сброс\n4. Вычисление норм вектора (массива)\n5. Нормировка\n6. Выход\nВыбор:");  do {  scanf("%d", &flag);  if (flag < 1 || flag > 6)  printf("\nНеверно введен номер, введите снова.\n");  } while (flag < 1 || flag > 6);  switch (flag) {  case(1):  clrscr();  printf("Массив из %d чисел\n", count);  PrintArray(array, count);  break;  case(2):  clrscr();  printf("Выберите способ сортировки:\n1. Пузырьковая\n2. Вставками\n3. Быстрая\n4. Назад в меню\n");  do  {  scanf("%d", &sortflag);  if (sortflag < 1 || sortflag > 4)  printf("\nВведен неверный номер. Введите снова:\n");  } while (sortflag < 1 || sortflag > 4);  switch (sortflag) {  case(1):  clrscr();  printf("Сортировка в процессе выполнения...");  startTime = clock();  bubbleSort(array, count);  endTime = clock();  printf("Время выполнения: %lf\n", (endTime - startTime) / 1000);  printf("Сортировка завершена\n\n");  break;  case(2):  clrscr();  printf("Сортировка в процессе выполнения...\n");  startTime = clock();  inSort(array, count);  endTime = clock();  printf\_s("Время выполнения %lf\n", (endTime - startTime) / 1000);  printf\_s("Сортировка завершена\n\n");  break;  case(3):  clrscr();  printf("Сортировка в процессе выполнения...");  startTime = clock();  qSort(array, 0, count - 1);  endTime = clock();  printf\_s("Время выполнения %lf\n", (endTime - startTime) / 1000);  printf\_s("Cортировка завершена\n\n");  break;  case(4):  clrscr();  break;  }  break;  case(3):  clrscr();  memcpy(array, array2, sizeof(char) \* count);  printf\_s("Сортировка сброшена\n\n");  break;  case(4):  clrscr();  printf("Вычисление...\n");  norm1 = resultnorm1(array, count);  printf("Первая норма массива: %d\n", norm1);  norm2 = resultnorm2(array, count);  printf("Вторая норма массива: %lf\n", norm2);  printf("Введите степень p для Гельдеровой нормы:\n");  scanf("%lf", &p);  normG = resultnormG(array, count, p);  printf("Гельдерова норма массива: %lf\n", normG);  normI = infinitynorma(array, count);  printf("Бесконечная норма: %d\n\n", normI);  break;  case(5):  normirovka(array, count);  break;  case(6):  printf\_s("\nВыход из программы...\n");  return 0;  break;  }  } while (flag == 1 || flag == 2 || flag == 3 || flag == 4 || flag == 5 || flag == 6);  fclose(file);  free(array);  free(array2);  } |