Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Нижегородский государственный архитектурно-строительный университет»

(ННГАСУ)

Институт технологий бизнеса

Кафедра прикладной информатики и статистики

Курсовая работа

по дисциплине «Программирование»

Тема: «Разработка модуля сортировки файлов»

|  |  |
| --- | --- |
| Выполнил: | Студент группы ПРИ21.21/1  Очкаленко А.А. |
| Проверил: | Доцент, доктор технических наук  Платов А.Ю. |

Нижний Новгород

2022 г

**Содержание**

**Оглавление**

[**1. Введение.** 2](#_Toc103795516)

[**2. Реализация функций сортировки.** 4](#_Toc103795517)

[2.1. Сортировка простого обмена 4](#_Toc103795518)

[2.2. Сортировка простой вставки 4](#_Toc103795519)

[2.3. Сортировка простого обмена 4](#_Toc103795520)

[2.4. Сортировка расчёской 5](#_Toc103795521)

[2.5. Сортировка Шелла 5](#_Toc103795522)

[2.6. Быстрая сортировка 6](#_Toc103795523)

[2.7. Сортировка слиянием 6](#_Toc103795524)

[2.8. Пирамидальная сортировка 7](#_Toc103795525)

[**3.Другие файлы** 8](#_Toc103795526)

[3.1. sort.hpp 8](#_Toc103795527)

[3.2. test.cpp 13](#_Toc103795528)

[3.3. main.cpp 20](#_Toc103795529)

[**4. Заключение.** 21](#_Toc103795530)

# **1. Введение.**

Задачи:

1. Реализация функций сортировки: простого обмена, простой вставки, простого выбора, расчёской, Шелла, быстрой сортировки, слиянием, пирамидальной.
2. Реализовать вызов функции сортировки через единственную интерфейсную функцию.
3. Реализовать выбор метода сортировки через специальный параметр (SORT\_METHOD).
4. Реализовать выбор направления сортировки по убыванию-возрастанию (SORT\_DIRECTION).
5. Предусмотреть в библиотеке «говорящие» константы для набора значений обоих параметров.
6. Предусмотреть обработку исключений.
7. Добавить к каждой функции описание.
8. Создать файл README с описанием интерфейса библиотеки на английском языке в стиле страниц MAN.
9. Разработать тесты для прогона всех методов

# **2. Реализация функций сортировки.**

# 2.1. Сортировка простого обмена

*template* <*typename* T>

*void* bubble\_sort(T \*array, *int* n, *int* SORT\_DIRECTION) {

*bool* swapped = false;

*for*(*int* i = 0; i < n-1; i++) {

        swapped = false;

        T temp;

*for*(*int* j = 0; j < n-i-1; j++) {

*if*(comp(&array[j], &array[j+1]) == SORT\_DIRECTION) {

                move(&temp, &array[j], sizeof(T));

                move(&array[j], &array[j+1], sizeof(T));

                move(&array[j+1], &temp, sizeof(T));

                swapped = true;

            }

        }

*if*(!swapped) *break*;

    }

}

# 2.2. Сортировка простой вставки

*template* <*typename* T>

*void* insertion\_sort(T \*array,*int* n, *int* SORT\_DIRECTION) {

    T temp;

*for*(*int* i=1;i<n;i++){

*for*(*int* j=i; j>0 && comp(&array[j-1], &array[j]) == SORT\_DIRECTION;j--) {

            move(&temp, &array[j], sizeof(T));

            move(&array[j], &array[j-1], sizeof(T));

            move(&array[j-1], &temp, sizeof(T));

        }

    }

}

# 2.3. Сортировка простого обмена

*template*<*typename* T>

*void* selection\_sort(T \*array, *int* n, *int* SORT\_DIRECTION) {

    T temp;

*for* (*int* i = 0; i < n; i++) {

        move(&temp, &array[0], sizeof(T));

*for* (*int* j = i + 1; j < n; j++) {

*if* (comp(&array[i], &array[j]) == SORT\_DIRECTION) {

            move(&temp, &array[i], sizeof(T));

            move(&array[i], &array[j], sizeof(T));

            move(&array[j], &temp, sizeof(T));

            }

        }

    }

}

# 2.4. Сортировка расчёской

*template* <*typename* T>

*void* comb\_sort(T \*arr, *int* n, *int* SORT\_DIRECTION) {

*int* i, h = n;

*double* shrink = 1.33;

*while* (h > 1){

        h = h / shrink;

*if* (h <= 1) *break*;

*for* (i = 0; i + h < n; i++){

*if* (comp(&arr[i], &arr[i+h]) == SORT\_DIRECTION) {

                T temp;

                move(&temp, &arr[i], sizeof(T));

                move(&arr[i], &arr[i+h], sizeof(T));

                move(&arr[i+h], &temp, sizeof(T));

            }

        }

    }

    bubble\_sort(arr, n, SORT\_DIRECTION);

}

# 2.5. Сортировка Шелла

*template* <*typename* T>

*void* shell\_sort(T \*arr, *int* n, *int* SORT\_DIRECTION) {

*int* i, j, h;

*double* shrink = 1.88;

*for*(h = n/shrink; h > 0; h/=shrink) {

*for*(i = h; i < n; i ++){

            T temp;

            move(&temp, &arr[i], sizeof(arr[i]));

*for*(j=i; j>=h; j-=h){

*if* (comp( &arr[j-h], &temp) == SORT\_DIRECTION) {

                arr[j] = arr[j-h];

                move(&arr[j], &arr[j-h], sizeof(T));

                }

*else* *break*;

            }

            move(&arr[j], &temp, sizeof(T));

        }

    }

}

# 2.6. Быстрая сортировка

*template* <*typename* T>

*int* partition(T \*arr, *int* n, *int* SORT\_DIRECTION) {

    T x = arr[n/2];

    T temp;

*int* i = 0, j = n - 1;

*while* (1){

*while* (comp( &x, &arr[i]) == SORT\_DIRECTION) {

        i++;

    }

*while* (comp( &arr[j], &x) == SORT\_DIRECTION) {

        j--;

    }

*if* (i >= j) *break*;

    move(&temp, &arr[i], sizeof(T));

    move(&arr[i], &arr[j], sizeof(T));

    move(&arr[j], &temp, sizeof(T));

    i++;

    j--;

    }

*return* i;

}

*template* <*typename* T>

*void* quick\_sort(T \*arr, *int* n, *int* SORT\_DIRECTION) {

*if* (n < 2) *return*;

*int* i = partition(arr, n, SORT\_DIRECTION);

    quick\_sort(arr, i, SORT\_DIRECTION);

    quick\_sort(arr + i, n - i, SORT\_DIRECTION);

}

# 2.7. Сортировка слиянием

*template* <*typename* T> *void* merge\_sort(T \*array, *int* n, *int* SORT\_DIRECTION) {

*bool* swapped = false;

*for*(*int* i = 0; i < n-1; i++) {

        swapped = false;

        T temp;

*for*(*int* j = 0; j < n-i-1; j++) {

*if*(comp(&array[j], &array[j+1]) == SORT\_DIRECTION) {

                move(&temp, &array[j], sizeof(T));

                move(&array[j], &array[j+1], sizeof(T));

                move(&array[j+1], &temp, sizeof(T));

                swapped = true;

            }

        }

*if*(!swapped) *break*;

    }

}

# 2.8. Пирамидальная сортировка

*template* <*typename* T>

*void* heap\_s(T \*array, *int* len\_a, *int* k, *int* SORT\_DIRECTION) {

*int* nmax = k, l = 2\*k + 1, r = 2\*k + 2;

    T temp;

*if*(l < len\_a && comp(&array[l], &array[nmax]) == SORT\_DIRECTION) nmax = l;

*if*(r < len\_a && comp(&array[r], &array[nmax]) == SORT\_DIRECTION) nmax = r;

*if*(nmax != k) {

        move(&temp, &array[k], sizeof(T));

        move(&array[k], &array[nmax], sizeof(T));

        move(&array[nmax], &temp, sizeof(T));

        heap\_s(array, len\_a, nmax, SORT\_DIRECTION);

    }

}

*template* <*typename* T>

*void* heap\_sort(T \*array, *int* len\_a, *int* SORT\_DIRECTION) {

*int* k;

    T temp;

*for*(k = len\_a/2-1; k >= 0; k--) heap\_s(array, len\_a, k, SORT\_DIRECTION);

*for*(k = len\_a-1; k >= 0; k--){

        move(&temp, &array[0], sizeof(T));

        move(&array[0], &array[k], sizeof(T));

        move(&array[k], &temp, sizeof(T));

        heap\_s(array,k,0, SORT\_DIRECTION);

    }

}

# **3.Другие файлы**

# 3.1. sort.hpp

*#pragma* once

*#include* <iostream>

*#include* <cstring>

*#include* <string>

*enum* {

    \_short\_int, \_unsigned\_short\_int, \_int, \_unsigned\_int, \_float, \_double

} DATA\_TYPE;

*enum* methods {

    bubble, insertion, selection, comb, shell, merge, quick, heap

} SORT\_METOD;

*enum* directions {

    descending = -1, ascending = 1

} SORT\_DIRECTION;

*/\**

*Так как шаблон - это не функция а образец, по которому компилятор сделает функцию,*

*необходимо было перенести определение шаблона из .cpp в .hpp,*

*поскольку в точке использования должен быть доступ к определению шаблона,*

*иначе как компилятор узнает каким образом делать для нас функцию?*

*Однако есть несколько нюансов, но ничего плохого в них я не вижу.*

*\*/*

*/\**

*Функция сравнения двух элементов.*

*Возвращает -1, если  x1 < x2*

*Возвращает 0, если  x1 == x2*

*Возвращает 1, если  x1 > x2.*

*\*/*

*template* <*typename* T> *int* comp(T \*x1, T \*x2) {

*if* ( \*(T\*)x1 < \*(T\*)x2 ) *return* -1;

*else* *if* ( \*(T\*)x1 > \*(T\*)x2 ) *return* 1;

*else* *return* 0;

}

*/\**

*Функция пересылки b в a.*

*\*/*

*void* move(*void* \*a, *void* \*b, *int* elemsize) {

    std::memcpy(a, b, elemsize);

}

*template* <*typename* T> *void* print\_array(T \*arr, *int* n) {

*for*(*int* i = 0; i < n; i++) std::cout << arr[i] << " ";

}

*template* <*typename* T>

*void* bubble\_sort(T \*array, *int* n, *int* SORT\_DIRECTION) {

*bool* swapped = false;

*for*(*int* i = 0; i < n-1; i++) {

        swapped = false;

        T temp;

*for*(*int* j = 0; j < n-i-1; j++) {

*if*(comp(&array[j], &array[j+1]) == SORT\_DIRECTION) {

                move(&temp, &array[j], sizeof(T));

                move(&array[j], &array[j+1], sizeof(T));

                move(&array[j+1], &temp, sizeof(T));

                swapped = true;

            }

        }

*if*(!swapped) *break*;

    }

}

*template* <*typename* T>

*void* insertion\_sort(T \*array,*int* n, *int* SORT\_DIRECTION) {

    T temp;

*for*(*int* i=1;i<n;i++){

*for*(*int* j=i; j>0 && comp(&array[j-1], &array[j]) == SORT\_DIRECTION;j--) {

            move(&temp, &array[j], sizeof(T));

            move(&array[j], &array[j-1], sizeof(T));

            move(&array[j-1], &temp, sizeof(T));

        }

    }

}

*template*<*typename* T>

*void* selection\_sort(T \*array, *int* n, *int* SORT\_DIRECTION) {

    T temp;

*for* (*int* i = 0; i < n; i++) {

        move(&temp, &array[0], sizeof(T));

*for* (*int* j = i + 1; j < n; j++) {

*if* (comp(&array[i], &array[j]) == SORT\_DIRECTION) {

            move(&temp, &array[i], sizeof(T));

            move(&array[i], &array[j], sizeof(T));

            move(&array[j], &temp, sizeof(T));

            }

        }

    }

}

*template* <*typename* T>

*void* comb\_sort(T \*arr, *int* n, *int* SORT\_DIRECTION) {

*int* i, h = n;

*double* shrink = 1.33;

*while* (h > 1){

        h = h / shrink;

*if* (h <= 1) *break*;

*for* (i = 0; i + h < n; i++){

*if* (comp(&arr[i], &arr[i+h]) == SORT\_DIRECTION) {

                T temp;

                move(&temp, &arr[i], sizeof(T));

                move(&arr[i], &arr[i+h], sizeof(T));

                move(&arr[i+h], &temp, sizeof(T));

            }

        }

    }

    bubble\_sort(arr, n, SORT\_DIRECTION);

}

*template* <*typename* T>

*void* shell\_sort(T \*arr, *int* n, *int* SORT\_DIRECTION) {

*int* i, j, h;

*double* shrink = 1.88;

*for*(h = n/shrink; h > 0; h/=shrink) {

*for*(i = h; i < n; i ++){

            T temp;

            move(&temp, &arr[i], sizeof(arr[i]));

*for*(j=i; j>=h; j-=h){

*if* (comp( &arr[j-h], &temp) == SORT\_DIRECTION) {

                arr[j] = arr[j-h];

                move(&arr[j], &arr[j-h], sizeof(T));

                }

*else* *break*;

            }

            move(&arr[j], &temp, sizeof(T));

        }

    }

}

*template* <*typename* T>

*int* partition(T \*arr, *int* n, *int* SORT\_DIRECTION) {

    T x = arr[n/2];

    T temp;

*int* i = 0, j = n - 1;

*while* (1){

*while* (comp( &x, &arr[i]) == SORT\_DIRECTION) {

        i++;

    }

*while* (comp( &arr[j], &x) == SORT\_DIRECTION) {

        j--;

    }

*if* (i >= j) *break*;

    move(&temp, &arr[i], sizeof(T));

    move(&arr[i], &arr[j], sizeof(T));

    move(&arr[j], &temp, sizeof(T));

    i++;

    j--;

    }

*return* i;

}

*template* <*typename* T>

*void* quick\_sort(T \*arr, *int* n, *int* SORT\_DIRECTION) {

*if* (n < 2) *return*;

*int* i = partition(arr, n, SORT\_DIRECTION);

    quick\_sort(arr, i, SORT\_DIRECTION);

    quick\_sort(arr + i, n - i, SORT\_DIRECTION);

}

*template* <*typename* T> *void* merge\_sort(T \*array, *int* n, *int* SORT\_DIRECTION) {

*bool* swapped = false;

*for*(*int* i = 0; i < n-1; i++) {

        swapped = false;

        T temp;

*for*(*int* j = 0; j < n-i-1; j++) {

*if*(comp(&array[j], &array[j+1]) == SORT\_DIRECTION) {

                move(&temp, &array[j], sizeof(T));

                move(&array[j], &array[j+1], sizeof(T));

                move(&array[j+1], &temp, sizeof(T));

                swapped = true;

            }

        }

*if*(!swapped) *break*;

    }

}

*template* <*typename* T>

*void* heap\_s(T \*array, *int* len\_a, *int* k, *int* SORT\_DIRECTION) {

*int* nmax = k, l = 2\*k + 1, r = 2\*k + 2;

    T temp;

*if*(l < len\_a && comp(&array[l], &array[nmax]) == SORT\_DIRECTION) nmax = l;

*if*(r < len\_a && comp(&array[r], &array[nmax]) == SORT\_DIRECTION) nmax = r;

*if*(nmax != k) {

        move(&temp, &array[k], sizeof(T));

        move(&array[k], &array[nmax], sizeof(T));

        move(&array[nmax], &temp, sizeof(T));

        heap\_s(array, len\_a, nmax, SORT\_DIRECTION);

    }

}

*template* <*typename* T>

*void* heap\_sort(T \*array, *int* len\_a, *int* SORT\_DIRECTION) {

*int* k;

    T temp;

*for*(k = len\_a/2-1; k >= 0; k--) heap\_s(array, len\_a, k, SORT\_DIRECTION);

*for*(k = len\_a-1; k >= 0; k--){

        move(&temp, &array[0], sizeof(T));

        move(&array[0], &array[k], sizeof(T));

        move(&array[k], &temp, sizeof(T));

        heap\_s(array,k,0, SORT\_DIRECTION);

    }

}

*/\**

*Функция выбора типа сортировки и её направления.*

*\*/*

*template* <*typename* T>

*void* sort(T \*array, *int* size, *int* elemsize, *int* SORT\_DIRECTION, *int* SORT\_METHOD) {

*if*(\*array == 0) {

        std::cout << "Массив не содержит элементов" << std::endl;

        exit(0);

    }

*else* *if*(size < 1) {

        std::cout << "Количество элементов массива меньше единицы" << std::endl;

        exit(0);

    }

*else* *if*(size == 1) {

        std::cout << "Массив из одного элемента уже отсортирован" << std::endl;

        exit(0);

    }

*switch* (SORT\_METHOD) {

*case* 0:

            bubble\_sort(array, size, SORT\_DIRECTION);

*break*;

*case* 1:

            insertion\_sort(array, size, SORT\_DIRECTION);

*break*;

*case* 2:

            selection\_sort(array, size, SORT\_DIRECTION);

*break*;

*case* 3:

            comb\_sort(array, size, SORT\_DIRECTION);

*break*;

*case* 4:

            shell\_sort(array, size, SORT\_DIRECTION);

*break*;

*case* 5:

            merge\_sort(array, size, SORT\_DIRECTION);

*break*;

*case* 6:

            quick\_sort(array, size, SORT\_DIRECTION);

*break*;

*case* 7:

            heap\_sort(array, size, SORT\_DIRECTION);

*break*;

*default*:

            std::cout << "SOMETHING WENT WRONG" << std::endl;

    }

*return* print\_array(array, size);

}

## 3.2. test.cpp

*#include* "sort.hpp"

*#include* <limits>

*#include* <random>

using std::numeric\_limits;

*template* <*typename* T>

*void* findError\_ascending(T a[], *int* n){

*int* err = 0;

*for*(*int* i = 0; i < n-1; i++){

*if*(a[i] > a[i+1]) err++;

    }

*if*(!err) std::cout << n << "\t- массив отсортирован" << std::endl;

*else* std::cout << n << "\t- массив не отсортирован" << std::endl;

}

*template* <*typename* T>

*void* findError\_descending(T a[], *int* n){

*int* err = 0;

*for*(*int* i = 0; i < n-1; i++){

*if*(a[i] < a[i+1]) err++;

    }

*if*(!err) std::cout << n << "\t- массив отсортирован" << std::endl;

*else* std::cout << n << "\t- массив не отсортирован" << std::endl;

}

*void* test(*int* types){

*switch*(types){

*case* \_int:

*goto* \_int\_test;

*break*;

*case* \_short\_int:

*goto* \_short\_int\_test;

*break*;

*case* \_unsigned\_short\_int:

*goto* \_unsigned\_short\_int\_test;

*break*;

*case* \_unsigned\_int:

*goto* \_unsigned\_int\_test;

*break*;

*case* \_float:

*goto* \_float\_test;

*break*;

*case* \_double:

*goto* \_double\_test;

*break*;

*default*:

            std::cout << "WRONG DATA TYPE FOR TEST" << std::endl;

    }

        \_double\_test:

*for*(*int* i = 0; i <= 7; i++, SORT\_METOD = methods(i)){

*for*(*int* arraySize = 8; arraySize <= 268435456; arraySize \*= 2){

*double* \*array = new *double*[arraySize];

                std::random\_device rd;

                std::default\_random\_engine eng(rd());

                std::uniform\_real\_distribution<*double*> distr(-

*for*(*int* arrElem = 0; arrElem < arraySize; arrElem++) array[arrElem] = distr(rd);

*switch*(SORT\_METOD){

*case* bubble:

                    bubble\_sort(array, arraySize, ascending);

                    std::cout << "bubble\t";

                    findError\_ascending(array, arraySize);

*case* insertion:

                    insertion\_sort(array, arraySize, descending);

                    std::cout << "insertion\t";

                    findError\_descending(array, arraySize);

*case* selection:

                    selection\_sort(array, arraySize, ascending);

                    std::cout << "selection\t";

                    findError\_ascending(array, arraySize);

*case* comb:

                    comb\_sort(array, arraySize, descending);

                    std::cout << "comb\t";

                    findError\_descending(array, arraySize);

*case* shell:

                    shell\_sort(array, arraySize, ascending);

                    std::cout << "shell\t";

                    findError\_ascending(array, arraySize);

*case* quick:

                    quick\_sort(array, arraySize, descending);

                    std::cout << "quick\t";

                    findError\_descending(array, arraySize);

*case* heap:

                    heap\_sort(array, arraySize, ascending);

                    std::cout << "heap\t";

                    findError\_ascending(array, arraySize);

*case* merge:

                    merge\_sort(array, arraySize, descending);

                    std::cout << "merge\t";

                    findError\_descending(array, arraySize);

                }

                delete [] array;

            }

        }

        \_float\_test:

*for*(*int* i = 0; i <= 7; i++, SORT\_METOD = methods(i)){

*for*(*int* arraySize = 8; arraySize <= 268435456; arraySize \*= 2){

*float* \*array = new *float*[arraySize];

                std::random\_device rd;

                std::default\_random\_engine eng(rd());

                std::uniform\_real\_distribution<*float*> distr(-numeric\_limits<*float*>::max(), numeric\_limits<*float*>::max());

*for*(*int* arrElem = 0; arrElem < arraySize; arrElem++) array[arrElem] = distr(rd);

*switch*(SORT\_METOD){

*case* bubble:

                    bubble\_sort(array, arraySize, ascending);

                    std::cout << "bubble\t";

                    findError\_ascending(array, arraySize);

*case* insertion:

                    insertion\_sort(array, arraySize, descending);

                    std::cout << "insertion\t";

                    findError\_descending(array, arraySize);

*case* selection:

                    selection\_sort(array, arraySize, ascending);

                    std::cout << "selection\t";

                    findError\_ascending(array, arraySize);

*case* comb:

                    comb\_sort(array, arraySize, descending);

                    std::cout << "comb\t";

                    findError\_descending(array, arraySize);

*case* shell:

                    shell\_sort(array, arraySize, ascending);

                    std::cout << "shell\t";

                    findError\_ascending(array, arraySize);

*case* quick:

                    quick\_sort(array, arraySize, descending);

                    std::cout << "quick\t";

                    findError\_descending(array, arraySize);

*case* heap:

                    heap\_sort(array, arraySize, ascending);

                    std::cout << "heap\t";

                    findError\_ascending(array, arraySize);

*case* merge:

                    merge\_sort(array, arraySize, descending);

                    std::cout << "merge\t";

                    findError\_descending(array, arraySize);

                }

                delete [] array;

            }

        }

        \_unsigned\_int\_test:

*for*(*int* i = 0; i <= 7; i++, SORT\_METOD = methods(i)){

*for*(*int* arraySize = 8; arraySize <= 268435456; arraySize \*= 2){

*unsigned* *int* \*array = new *unsigned* *int*[arraySize];

                std::random\_device dev;

                std::mt19937 rng(dev());

                std::uniform\_int\_distribution<std::mt19937::result\_type> dist6(0, numeric\_limits<*int*>::max());

*for*(*int* arrElem = 0; arrElem < arraySize; arrElem++) array[arrElem] = dist6(rng);

*switch*(SORT\_METOD){

*case* bubble:

                    bubble\_sort(array, arraySize, ascending);

                    std::cout << "bubble\t";

                    findError\_ascending(array, arraySize);

*case* insertion:

                    insertion\_sort(array, arraySize, descending);

                    std::cout << "insertion\t";

                    findError\_descending(array, arraySize);

*case* selection:

                    selection\_sort(array, arraySize, ascending);

                    std::cout << "selection\t";

                    findError\_ascending(array, arraySize);

*case* comb:

                    comb\_sort(array, arraySize, descending);

                    std::cout << "comb\t";

                    findError\_descending(array, arraySize);

*case* shell:

                    shell\_sort(array, arraySize, ascending);

                    std::cout << "shell\t";

                    findError\_ascending(array, arraySize);

*case* quick:

                    quick\_sort(array, arraySize, descending);

                    std::cout << "quick\t";

                    findError\_descending(array, arraySize);

*case* heap:

                    heap\_sort(array, arraySize, ascending);

                    std::cout << "heap\t";

                    findError\_ascending(array, arraySize);

*case* merge:

                    merge\_sort(array, arraySize, descending);

                    std::cout << "merge\t";

                    findError\_descending(array, arraySize);

                }

                delete [] array;

            }

        }

        \_unsigned\_short\_int\_test:

*for*(*int* i = 0; i <= 7; i++, SORT\_METOD = methods(i)){

*for*(*int* arraySize = 8; arraySize <= 268435456; arraySize \*= 2){

*unsigned* *short* \*array = new *unsigned* *short*[arraySize];

                std::random\_device dev;

                std::mt19937 rng(dev());

                std::uniform\_int\_distribution<std::mt19937::result\_type> dist6(0, numeric\_limits<*short* *int*>::max());

*for*(*int* arrElem = 0; arrElem < arraySize; arrElem++) array[arrElem] = dist6(rng);

*switch*(SORT\_METOD){

*case* bubble:

                    bubble\_sort(array, arraySize, ascending);

                    std::cout << "bubble\t";

                    findError\_ascending(array, arraySize);

*case* insertion:

                    insertion\_sort(array, arraySize, descending);

                    std::cout << "insertion\t";

                    findError\_descending(array, arraySize);

*case* selection:

                    selection\_sort(array, arraySize, ascending);

                    std::cout << "selection\t";

                    findError\_ascending(array, arraySize);

*case* comb:

                    comb\_sort(array, arraySize, descending);

                    std::cout << "comb\t";

                    findError\_descending(array, arraySize);

*case* shell:

                    shell\_sort(array, arraySize, ascending);

                    std::cout << "shell\t";

                    findError\_ascending(array, arraySize);

*case* quick:

                    quick\_sort(array, arraySize, descending);

                    std::cout << "quick\t";

                    findError\_descending(array, arraySize);

*case* heap:

                    heap\_sort(array, arraySize, ascending);

                    std::cout << "heap\t";

                    findError\_ascending(array, arraySize);

*case* merge:

                    merge\_sort(array, arraySize, descending);

                    std::cout << "merge\t";

                    findError\_descending(array, arraySize);

                }

                delete [] array;

            }

        }

        \_short\_int\_test:

*for*(*int* i = 0; i <= 7; i++, SORT\_METOD = methods(i)){

*for*(*int* arraySize = 8; arraySize <= 268435456; arraySize \*= 2){

*short* \*array = new *short*[arraySize];

                std::random\_device dev;

                std::mt19937 rng(dev());

                std::uniform\_int\_distribution<std::mt19937::result\_type> dist6(-numeric\_limits<*short* *int*>::max(), numeric\_limits<*short* *int*>::max());

*for*(*int* arrElem = 0; arrElem < arraySize; arrElem++) array[arrElem] = dist6(rng);

*switch*(SORT\_METOD){

*case* bubble:

                    bubble\_sort(array, arraySize, ascending);

                    std::cout << "bubble\t";

                    findError\_ascending(array, arraySize);

*case* insertion:

                    insertion\_sort(array, arraySize, descending);

                    std::cout << "insertion\t";

                    findError\_descending(array, arraySize);

*case* selection:

                    selection\_sort(array, arraySize, ascending);

                    std::cout << "selection\t";

                    findError\_ascending(array, arraySize);

*case* comb:

                    comb\_sort(array, arraySize, descending);

                    std::cout << "comb\t";

                    findError\_descending(array, arraySize);

*case* shell:

                    shell\_sort(array, arraySize, ascending);

                    std::cout << "shell\t";

                    findError\_ascending(array, arraySize);

*case* quick:

                    quick\_sort(array, arraySize, descending);

                    std::cout << "quick\t";

                    findError\_descending(array, arraySize);

*case* heap:

                    heap\_sort(array, arraySize, ascending);

                    std::cout << "heap\t";

                    findError\_ascending(array, arraySize);

*case* merge:

                    merge\_sort(array, arraySize, descending);

                    std::cout << "merge\t";

                    findError\_descending(array, arraySize);

                }

                delete [] array;

            }

        }

        \_int\_test:

*for*(*int* i = 0; i <= 7; i++, SORT\_METOD = methods(i)){

*for*(*int* arraySize = 8; arraySize <= 268435456; arraySize \*= 2){

*int* \*array = new *int*[arraySize];

                std::random\_device dev;

                std::mt19937 rng(dev());

                std::uniform\_int\_distribution<std::mt19937::result\_type> dist6(-numeric\_limits<*int*>::max(), numeric\_limits<*int*>::max());

*for*(*int* arrElem = 0; arrElem < arraySize; arrElem++) array[arrElem] = dist6(rng);

*switch*(SORT\_METOD){

*case* bubble:

                    bubble\_sort(array, arraySize, ascending);

                    std::cout << "bubble\t";

                    findError\_ascending(array, arraySize);

*case* insertion:

                    insertion\_sort(array, arraySize, descending);

                    std::cout << "insertion\t";

                    findError\_descending(array, arraySize);

*case* selection:

                    selection\_sort(array, arraySize, ascending);

                    std::cout << "selection\t";

                    findError\_ascending(array, arraySize);

*case* comb:

                    comb\_sort(array, arraySize, descending);

                    std::cout << "comb\t";

                    findError\_descending(array, arraySize);

*case* shell:

                    shell\_sort(array, arraySize, ascending);

                    std::cout << "shell\t";

                    findError\_ascending(array, arraySize);

*case* quick:

                    quick\_sort(array, arraySize, descending);

                    std::cout << "quick\t";

                    findError\_descending(array, arraySize);

*case* heap:

                    heap\_sort(array, arraySize, ascending);

                    std::cout << "heap\t";

                    findError\_ascending(array, arraySize);

*case* merge:

                    merge\_sort(array, arraySize, descending);

                    std::cout << "merge\t";

                    findError\_descending(array, arraySize);

                }

                delete [] array;

            }

        }

}

*int* main(){

    test(\_int);

*return* 0;

}

# 3.3. main.cpp

*#include* "sort.hpp"

*int* main() {

*char* arr[] = {'a', 'c', 'e', 'b', 'd'};

    SORT\_METOD = heap;

    SORT\_DIRECTION = ascending;

    sort(arr, sizeof(arr)/sizeof(*char*), sizeof(*double*), SORT\_DIRECTION, SORT\_METOD);

}

# **4. Заключение.**

В данной курсовой работе на выходе получилась программа(модуль) сортировки файлов. Цель, поставленная вначале работы достигнута, задачи выполнены.