ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ АРХИТЕКТУРНО-СТРОИТЕЛЬНЫЙ УНИВЕРСИТЕТ»

Институт технологий бизнеса

Кафедра прикладной информатики и статистики

**Курсовая работа**

Выполнил

студент 1 курса гр. ПРи 23/1 Пронин А. А

Проверил:

Зав. кафедрой, доктор тех. наук, доцент

Платов А.Ю

Нижний Новгород

2024

**Содержание**

Оглавление

[Задание 3](file:///C:\Users\Артём\Downloads\Telegram%20Desktop\Курсовая%20прога.docx#_Toc167743769)

[1. Описание: 4](file:///C:\Users\Артём\Downloads\Telegram%20Desktop\Курсовая%20прога.docx#_Toc167743770)

[2. Выбор метода сортировки: 4](file:///C:\Users\Артём\Downloads\Telegram%20Desktop\Курсовая%20прога.docx#_Toc167743771)

[3. Направление сортировки: 4](file:///C:\Users\Артём\Downloads\Telegram%20Desktop\Курсовая%20прога.docx#_Toc167743772)

[4. Интерфейсная функция: 5](file:///C:\Users\Артём\Downloads\Telegram%20Desktop\Курсовая%20прога.docx#_Toc167743773)

[5. Константы: 7](file:///C:\Users\Артём\Downloads\Telegram%20Desktop\Курсовая%20прога.docx#_Toc167743774)

[6. Пример программы 7](file:///C:\Users\Артём\Downloads\Telegram%20Desktop\Курсовая%20прога.docx#_Toc167743775)

[7. Набор тестов: 8](file:///C:\Users\Артём\Downloads\Telegram%20Desktop\Курсовая%20прога.docx#_Toc167743776)

[Приложение 1. 9](file:///C:\Users\Артём\Downloads\Telegram%20Desktop\Курсовая%20прога.docx#_Toc167743777)

# Задание

Требования:

1) Реализация функций сортировки: простого обмена, простой вставки, простого выбора, расчёской, Шелла, быстрой сортировки, слиянием, пирамидальной.

2) Вызов функции сортировки осуществляется через единственную интерфейсную функцию   
int sort(void\*array,int size,int elemsize,int(\*comp)(void\*a1,void\*a2)); Здесь array -- сортируемый массив, size -- размер массива, comp -- функция для сравнения элементов, которая возвращает -1, если a1 < a2, 0, если a1==a2 и 1, если a1>a2 elemsize -- размер элемента

3) Выбор метода сортировки выполняется через специальный параметр. SORT\_METHOD

4) Выбор направления сортировки (по убыванию-возрастанию) выполняется через специальный параметр. SORT\_DIRECTION

5) Предусмотреть в библиотеке «говорящие» константы для набора значений обоих параметров.

6) Предусмотреть обработку исключений

7) Код оформить в стиле 1TBS. https://ru.wikipedia.org/wiki/Отступ (програмирование)

8) Добавить к каждой функции описание

9) Создать файл README с описанием интерфейса библиотеки на английском языке в стиле страниц MAN.

10) Разработать тесты для прогона всех методов

# 1. Описание:

Библиотека “sort.h” предоставляет реализации сортировки различных типов данных (char, int, double) алгоритмами:

Пузырьковая сортировка

Сортировка вставкой

Сортировка по выбору

Сортировка расчёской

Сортировка Шелла

Быстрая сортировка

Сортировка слиянием

Пирамидальная сортировка

# 2. Выбор метода сортировки:

Метод сортировки выбирается с помощью специального сеттера

Int sort\_set\_method(int method) – проверяет правильность выбранного метода и присваивает глобальной статической переменной значение метода. В противном случае установит метод по умолчанию (пузырьковая сортировка), напишет ошибку и продолжит работу (в этом случае функция вернёт значение 1).

# 3. Направление сортировки:

Метод сортировки выбирается с помощью специального сеттера

Int sort\_set\_direction(int direction) – проверяет правильность выбранного направления и присваивает глобальной статической переменной значение. В противном случае установит направление по умолчанию (по возрастанию), напишет ошибку и продолжит работу (в этом случае функция вернёт значение 1).

Направление не влияет на сами сортировки, а всего лишь “разворачивает” уже отсортированный массив, что впоследствии повлечёт отсутствие необходимости дополнительного тестирования этой функции.

# 4. Интерфейсная функция:

int sort(void \*array, int size, int elemsize, int((\*comp)(void\* a1, void\* a2));

Параметры:

array: указатель на начало массива, который нужно отсортировать.

len: количество элементов в массиве.

elemsize: тип данных, для каждого элемента в массиве, для целочисленных он соответствует их размеру в байтах.

**Ошибки:**

Если передан пустой указатель вернёт -1 и завершит сортировку.

Если длина массива 1 и меньше вернёт -1 и завершит сортировку.

Если передан некорректный метод сортровки вернёт -2 и завершит сортировку.

В случае удовлетворительных аргументов выполнится сортировка.

**Суть**

Предварительно создастся глобальная переменная metod, куда передаётся назначенный для сортировки тип. Затем локальная переменная size, означающая размер типа данных в байтах.

При помощи switch-case выбирается метод сортировки, указанный в глобальной статической переменной. Происходит сортировка, выбранным методом. Рассмотрим на примере пузырьковой:

bubblesort(array, size, elemsize, comp);

В двух вложенных циклах происходит универсальное сравнение

if (1 == comp(array + j\*elemsize, array + (j+1)\*elemsize))

равносильно

if (array[j] > array[j+1])

Сравнение происходит при помощи указателей на нужные элементы для этого позиция j умножается на размер одного элемента size.

Встроенная функция compare сравнивает два элемента, на которые ссылаются указатели, в зависимости от типа type, используя switch-case происходит выбор типа данных для сравнения и непосредственно сравнение.

Например, char выглядит как:

if( \*((char\*)a) < \*((char\*)b) ) return -1;  
if( \*((char\*)a) > \*((char\*)b) ) return 1;  
return 0;

Void указатель преобразуется в тип данных, выбранный пользователем. По этому шаблону работают остальные сравнения. Если первые число больше возвращает 1, если второе -1, если равны 0.

Пользователь может заменить функцию compare и передавать вместо неё в качестве аргумента свою функцию. Это нужно например в случае, когда представленных типов данных недостаточно.

Функция swap так же принимает два указателя и размер данных в байтах. При помощи memcpy присваивает временной переменной значение и меняет местами значения указателей.

swap(array + j\*elemsize, array + j\*elemsize + 1\*elemsize, elemsize);

После сортировки происходит проверка направления, если оно восходящее, функция заканчивается, если нисходящее – записывает массив в обратном порядке после чего прекращает сортировку. В обоих случаях считается, что сортировка прошла успешно, функция возвращает 0.

# 5. Константы:

Обозначающие направление сортировки:

SORT\_DIRECTION\_ASCENDING 1   
SORT\_DIRECTION\_DESCENDING 2

Обозначающие методы сортировок

SORT\_METHOD\_EXCHANGE 1

SORT\_METHOD\_INSERTION 2

SORT\_METHOD\_SELECTION 3

SORT\_METHOD\_COMB 4

SORT\_METHOD\_SHELL 5

SORT\_METHOD\_QUICK 6

SORT\_METHOD\_MERGE 7

SORT\_METHOD\_HEAP 8

Для выбора типа сортируемых данных

INT CHAR  
DOUBLE

# 6. Пример программы

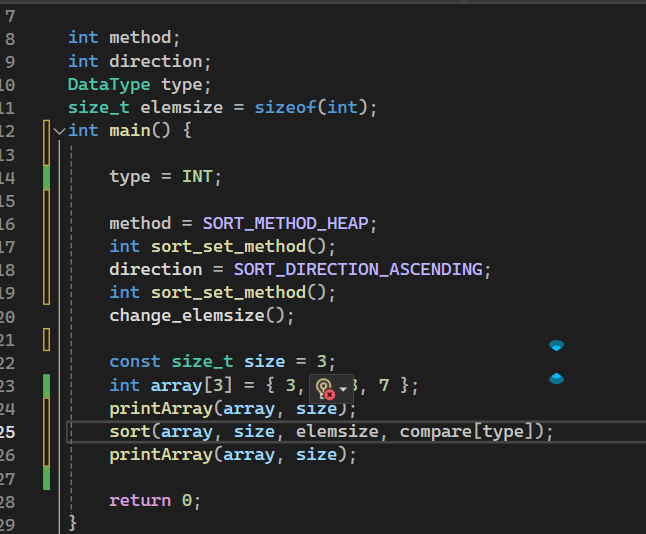


Рис. 1. Пример программы

# 7. Набор тестов:

“test.h” Эта тестовая программа разработана для проверки работоспособности различных методов сортировки на тестовых данных. (Приложение 1.) Программа и использует циклы for для прогона каждого метода сортировки.

Функция test(int) принимает как аргумент длину массива и с помощью двух вложенных циклов поочерёдно передаёт все комбинации различных методов сортировки и типов данных. Про отсутствие необходимости проверки писал в пункте “Направление сортировки”.

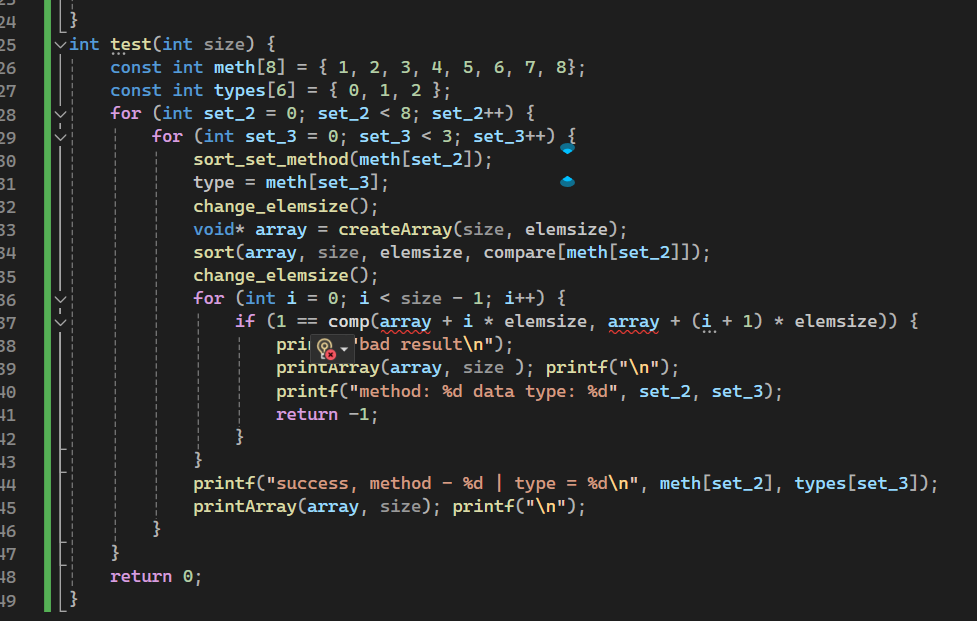
Тестовая программа создает массив тестовых данных поочерёдно меняя тип данных и перезаполняя его, который затем подается на вход каждому методу сортировки. Затем сортируется с помощью текущего метода сортировки.

Результаты сортировки каждого метода выводятся на экран. Это позволяет наглядно проверить правильность работы каждого метода сортировки на тестовых данных. Если проверка не пройдена функция теста выводит не прошедшие тест данные и останавливается.

Такой подход обеспечивает простой способ добавления новых методов сортировки и оценки их производительности на тестовых данных. Также он упрощает анализ результатов и сравнение различных методов сортировки.

В приложении 2 представлен тест массивов длинной 5. Все прогоны пройдены успешно, а значит можно считать что библиотека “sort.h ” рабочая.

# Приложение 1.



# Приложение 2.

