МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Алгоритмы и структуры данных»
Тема: Алгоритмы сортировки

Студент гр. 9381 Матвеев А. Н.

Преподаватель Фирсов М. А.

Санкт-Петербург
2020

Цель работы.

Изучение и реализация алгоритмов сортировки.

Задание.

Вариант 4.

Пузырьковая сортировка оптимизированная; сортировка чёт-нечет.

Описание работы алгоритмов.

1. **Оптимизированный алгоритм пузырьковой сортировки.**

Базовый алгоритм пузырьковой сортировки состоит в повторяющихся проходах по сортируемому массиву. На каждой итерации последовательно сравниваются соседние элементы, и, если порядок в паре неверный, то элементы меняют местами. За каждый проход по массиву **хотя бы один элемент встает на свое место**, поэтому необходимо совершить не более n−1 проходов (n - размер массива), чтобы отсортировать массив.

В реализации алгоритма используются 2 цикла, один из которых вложенный. Внешний цикл отвечает за количество проходов по массиву, а внутренний за каждый конкретный проход.

Оптимизация 1.

Заметим, что после i-ой итерации внешнего цикла i последних элементов уже находятся на своих местах в отсортированном порядке, поэтому нет смысла сравнивать их друг с другом. Следовательно, внутренний цикл можно выполнять не до n−2 (включительно), а **до n−i−2** (включительно).

Оптимизация 2.

Также заметим, что если после выполнения внутреннего цикла не произошло ни одного обмена, то массив уже отсортирован, и продолжать дальше бессмысленно. Поэтому внешний цикл можно выполнять не n−1 раз (как это происходит в базовой реализации), **а до тех пор, пока во внутреннем цикле происходят обмены.** Это контролируется при помощи флага.

Достоинства и недостатки алгоритма.

Достоинства:

-В лучшем случае сложность $O(n)$;

-Большие относительно других элементов массива быстро передвигаются к концу.

Недостатки:

-Маленькие относительно других элементов массива медленно передвигаются к началу.

-В худшем и среднем случае сложность $O(n^2)$.

2. Сортировка чёт-нечет.

Сортировка чет-нечет — модификация пузырьковой сортировки, основанная на сравнении элементов стоящих на четных и нечетных позициях независимо друг от друга.

На первом проходе элементы с чётным индексом сравниваются с соседями, расположенными на нечётных местах (0-й сравнивается со 1-м, затем 2-й с 3-м и т. д.). Затем наоборот – элементы с нечётным индексом сравниваются (и при необходимости меняются) с элементами с чётным индексом). Аналогично на следующих итерациях. Процесс сортировки заканчивается тогда, когда после двух проходов **подряд** по массиву («чётно-нечётному» и «нечётно-чётному») **не произошло ни одного обмена**. Для переключения между видами прохода используется флаг.

Достоинства и недостатки алгоритма.

Достоинства:

-В лучшем случае сложность $O(n)$;

-На нескольких процессорах она выполняется быстрее, чем пузырьковая так как четные и нечетные индексы сортируются параллельно;

-При одном проходе сразу все крупные относительно остальных элементов массива одновременно передвигаются вправо на одну позицию, в отличие от пузырьковой сортировки, где во время каждого прохода текущий максимум планомерно смещается в конец массива.

Недостатки:

-В худшем и среднем случае сложность $O(n^2)$.

Описание структур и пространств имён.

Для удобства создана структура **Array**, представляющая собой массив. Имеет поле **array** типа int * (указатель на начало динамического массива) и поле **size** типа int (количество элементов в массиве).

Пространство имён **file** содержит статические переменные inputFile типа ifstream и outputFile типа ofstream. (Эти типы данных объявлены в <fstream>). Это потоки ввода из файла и вывода в файл соответственно.

Описание функций.

(Примечание: используется целочисленный массив типа int, шаблоны не использовались, поскольку были использованы в прошлой работе).

void bubbleSortOptimized(int * array, int & arrSize, ostream & out) - функция, производящая оптимизированную сортировку пузырьком. Принимает на вход указатель на массив целых чисел, ссылку на целочисленную переменную (размер массива) и ссылку на поток вывода. Ничего не возвращает. Создана переменная-флаг flag, которая отвечает за выход из внешнего цикла, о котором будет сказано далее. Запускается цикл while, который прекращает повторяться, когда flag равен 0. В начале тела цикла flag всегда устанавливается в 0. Далее запускается вложенный в while цикл for, который совершает обход массива. Он работает до тех пор, пока текущий индекс массива j < arrSize - i - 1, где i - количество уже сделанных обходов. В цикле for каждый элемент сравнивается с последующим, и, если последний больше, они меняются местами при помощи библиотечной функции std::swap. Если во внутреннем цикле произошёл хотя бы 1 обмен, flag устанавливается в 1. После завершения внутреннего обхода i увеличивается на 1. Если за весь обход не произошло ни одного обмена, flag останется 0 и внешний цикл прекратит работу. Таким образом, массив сортируется алгоритмом оптимизированной сортировки пузырьком.

void oddEvenSorting(int * array, int & arrSize, ostream & out) - функция, производящая сортировку чёт-нечёт. Принимает на вход указатель на массив целых чисел, ссылку на целочисленную переменную (размер массива) и

ссылку на поток вывода. Ничего не возвращает. Создаются целочисленные переменные delta и checkChanges, от которых будет зависеть выход из внешнего цикла. Изначально они равны 0. Также создан флаг even, который необходим для переключения состояния обхода (“чёт-нечёт” или “нечёт-чёт”). Запускается внешний цикл while. Сначала в его теле запускается цикл for, который идёт до (arrSize - 2)-го элемента включительно с шагом 2. Флаг even определяет, начиная с какого индекса будет идти цикл на текущем обходе: с 0-го (если even установлен в true) или с 1-го (если even установлен в false). Значение even меняется на противоположное после каждого обхода массива. В цикле for аналогично предыдущей функции проверяется условие, что текущий элемент больше следующего и, при положительном исходе они меняются местами. Однако в этой функции в данном условии после этого также инкрементируется переменная checkChanges. После каждого обхода инкрементируется и delta, после чего проверяется условие выхода из внешнего цикла: delta == 2 и checkChanges == 0. Это объясняется тем, что если 2 обхода подряд элементы ни одной пары не поменялись местами, то сортировка завершена. Если же условие выхода не выполнено, но delta равна 2, то обе переменные обнуляются, чтобы обеспечить правильность дальнейших аналогичных проверок.

В функциях сортировки присутствуют отладочные выводы позволяющие понять работу алгоритмов.

void printArray(int * arr, int & size, ostream & out) - функция печати массива на экран. Принимает на вход указатель на массив целых чисел, ссылку на целочисленную переменную (размер массива) и ссылку на поток вывода. Ничего не возвращает. Выводит в консоль элементы массива через пробел.

Array *getArrayFromString(string &arrSeq) - функция, преобразующая корректно введённую строку в экземпляр структуры Array. Получает на вход ссылку на поданную ей строку. Возвращает либо нулевой указатель (если строка пустая) , либо указатель на экземпляр структуры Array. Если строка не пуста, то создаётся указатель на экземпляр структуры Array (под него

динамически выделяется память). Также создаётся строковый поток myStream типа stringstream. Этот тип данных находится в заголовочном файле <sstream>. В него записывается введённая строка. После этого подсчитывается количество пробелов в массиве (все элементы массива должны быть разделены строго одним пробелом). Таким образом размер массива становится известен. Далее, под поле array экземпляра Array выделяется память в size ячеек, после чего в цикле for каждая заполняется соответствующим значением при помощи оператора ввода из myStream. Здесь запись в поток из строки и из потока в строку служат способом выделения из строки целых чисел и преобразования их в int. Таким образом, массив заполнен и готов к использованию.

bool isCorrect(string & arrSeq) - функция, проверяющая корректность введённой строки. Принимает на вход ссылку на введённую строку. Помимо этого выводит в поток ошибок информацию об ошибках. Возвращает логическое значение, показывающее, корректна ли строка или нет. Строка некорректна, если в ней присутствуют символы, которые: не число, не пробел и не знак “минус” одновременно. При наличии двух и более идущих подряд пробелов или пробела в конце строка тоже считается некорректной. При проверке последнего условия значительную роль играет метод find() из класса string (заголовочный файл <cstring>) , который возвращает индекс первого вхождения подстроки в строку. В случае отсутствия подстроки возвращается -1. Если индекс вхождения подстроки “ “ не -1, значит формат неверный.

Array * copyArray(Array * obj) - функция, копирующая массив (Экземпляр Array). Принимает на вход указатель на экземпляр структуры Array, выделяет память под новый и копирует все данные в него. Возвращает либо nullptr, либо заполненную копию исходного экземпляра.

void destroy(Array * obj) - функция, очищающая память, выделенную под экземпляр структуры Array. Принимает на вход указатель на экземпляр, память которого должна быть очищена и освобождает её. Функция ничего не возвращает.

bool isEqual(Array * first, Array * second) - функция проверки массивов на идентичность. Принимает на вход 2 указателя на структуру Array (2 массива и поэлементно сравнивает их). Если все количество элементов равно и они совпадают, (или если оба указателя нулевые) то функция возвращает true, в противном случае - false. Используется при сравнении массивов, отсортированных реализованными алгоритмами и библиотечным.

В функции **main()** пользователь вводит три флага: inputFlag отвечает, откуда вводить(консоль/файл), outputFlag отвечает за вывод данных (консоль/файл) и sortFlag отвечает за метод сортировки: (оптимизированная пузырьковая / чёт-нечёт). После ввода флагов производится проверка на их корректность, затем пользователь сначала вводит либо элементы массива через пробел, либо путь до файла, откуда предполагается считать массив. Ключевую роль здесь играет функция std::getline, позволяющая считывать строку из потока. В случае с файлом при корректном вводе открывается на чтение поток inputFile и данные считываются оттуда.

Ещё до определения *куда* будет выведена информация, строка проверяется на корректность. В случае любого некорректного ввода выводится сообщение об ошибке и программа завершается. Создается ссылка на поток вывода типа ostream и инициализируется либо ссылкой на поток cout, либо на outputFile, в зависимости от флага. После этого создаются переменные-указатели на экземпляр структуры Array: arrayObject (основной массив, который будет обрабатываться) и его копия copy (необходима для функции библиотечной сортировки, чтобы проверить работоспособность реализованных алгоритмов). arrayObject инициализируется через функцию getArrayFromString, а copy - через функцию copyArray. Если была введена пустая строка, выводится соответствующее сообщение и программа завершится.

Наконец, по соответствующему флагу устанавливается алгоритм сортировки и массив сортируется соответствующим образом. В алгоритмах сортировки представлены подробные отладочные выводы, позволяющие понять

их принцип. После этого отсортированный массив выводится в нужный поток, и производится работа с копией исходного массива: указатель на первый и конец массива передаются в библиотечную функцию std::sort() из библиотеки <algorithm> и копия сортируется встроенным алгоритмом. После этого и она выводится, после чего отсортированные массивы сравниваются в функции isEqual, выводится результат сравнения, выделенная память освобождается.

Исходный код программы смотреть в Приложении А.

Тестирование.

Результаты тестирования представлены в таблицах 1.1 и 1.2.

Таблица 1.1. – Результаты тестирования (для оптимизированной пузырьковой сортировки)

№ п/п	Входные данные	Выходные данные
1.		Введена пустая строка. Сортировать нечего
2.	-5	Запуск оптимизированной сортировки пузырьком. Состояние массива: (-5) Массив состоит из одного элемента. Конец сортировки Отсортированный массив: (-5) Выполним проверку библиотечной функцией std::sort() Исходный массив: (-5) Массив после обработки std::sort(): (-5) Массивы идентичны.
3.	9 20 -73 87 59	Запуск оптимизированной сортировки пузырьком. Состояние массива: (9 20 -73 87 59) Обмен элементов (9, индекс 0) и (20, индекс 1) не требуется ! Элементы (-73, индекс 1) и (20, индекс 2) меняются местами Состояние массива: (9 -73 20 87 59)

		Обмен элементов (20, индекс 2) и (87, индекс 3) не требуется ! Элементы (59, индекс 3) и (87, индекс 4) меняются местами Состояние массива: (9 -73 20 59 87) ! Элементы (-73, индекс 0) и (9, индекс 1) меняются местами Состояние массива: (-73 9 20 59 87) Обмен элементов (9, индекс 1) и (20, индекс 2) не требуется Обмен элементов (20, индекс 2) и (59, индекс 3) не требуется Обмен элементов (-73, индекс 0) и (9, индекс 1) не требуется Обмен элементов (9, индекс 1) и (20, индекс 2) не требуется При текущем обходе не произошло ни одного обмена. (Условие конца сортировки) Сортировка закончена. Отсортированный массив: (-73 9 20 59 87) Выполним проверку библиотечной функцией std::sort() Исходный массив: (9 20 -73 87 59) Массив после обработки std::sort(): (-73 9 20 59 87) Массивы идентичны
4.	-74 -54 11 58 -63 -55	Запуск оптимизированной сортировки пузырьком. Состояние массива: (-74 -54 11 58 -63 -55) Обмен элементов (-74, индекс 0) и (-54, индекс 1) не требуется

		Обмен элементов (-54, индекс 1) и (11, индекс 2) не требуется Обмен элементов (11, индекс 2) и (58, индекс 3) не требуется ! Элементы (-63, индекс 3) и (58, индекс 4) меняются местами Состояние массива: (-74 -54 11 -63 58 -55) ! Элементы (-55, индекс 4) и (58, индекс 5) меняются местами Состояние массива: (-74 -54 11 -63 -55 58) Обмен элементов (-74, индекс 0) и (-54, индекс 1) не требуется Обмен элементов (-54, индекс 1) и (11, индекс 2) не требуется ! Элементы (-63, индекс 2) и (11, индекс 3) меняются местами Состояние массива: (-74 -54 -63 11 -55 58) ! Элементы (-55, индекс 3) и (11, индекс 4) меняются местами Состояние массива: (-74 -54 -63 -55 11 58) Обмен элементов (-74, индекс 0) и (-54, индекс 1) не требуется ! Элементы (-63, индекс 1) и (-54, индекс 2) меняются местами Состояние массива: (-74 -63 -54 -55 11 58)

		! Элементы (-55, индекс 2) и (-54, индекс 3) меняются местами Состояние массива: (-74 -63 -55 -54 11 58) Обмен элементов (-74, индекс 0) и (-63, индекс 1) не требуется Обмен элементов (-63, индекс 1) и (-55, индекс 2) не требуется При текущем обходе не произошло ни одного обмена. (Условие конца сортировки) Сортировка закончена. Отсортированный массив: (-74 -63 -55 -54 11 58) Выполним проверку библиотечной функцией std::sort() Исходный массив: (-74 -54 11 58 -63 -55) Массив после обработки std::sort(): (-74 -63 -55 -54 11 58) Массивы идентичны.
5.	1 1 3 5 7	Запуск оптимизированной сортировки пузырьком. Состояние массива: (1 1 3 5 7) Обмен элементов (1, индекс 0) и (1, индекс 1) не требуется Обмен элементов (1, индекс 1) и (3, индекс 2) не требуется Обмен элементов (3, индекс 2) и (5, индекс 3) не требуется Обмен элементов (5, индекс 3) и (7, индекс 4) не требуется При текущем обходе не произошло ни одного обмена. (Условие конца сортировки) Сортировка закончена.

		Отсортированный массив: (1 1 3 5 7) Выполним проверку библиотечной функцией std::sort() Исходный массив: (1 1 3 5 7) Массив после обработки std::sort(): (1 1 3 5 7) Массивы идентичны.
6.	0 0 -1	Запуск оптимизированной сортировки пузырьком. Состояние массива: (0 0 -1) Обмен элементов (0, индекс 0) и (0, индекс 1) не требуется ! Элементы (-1, индекс 1) и (0, индекс 2) меняются местами Состояние массива: (0 -1 0) ! Элементы (-1, индекс 0) и (0, индекс 1) меняются местами Состояние массива: (-1 0 0) При текущем обходе не произошло ни одного обмена. (Условие конца сортировки) Сортировка закончена. Отсортированный массив: (-1 0 0) Выполним проверку библиотечной функцией std::sort() Исходный массив: (0 0 -1) Массив после обработки std::sort(): (-1 0 0) Массивы идентичны.

Таблица 1.2. - Результаты тестирования (для сортировки чёт-нечёт)

№ п/п	Входные данные	Выходные данные
1.	-135	Запуск сортировки чёт-нечёт

		Состояние массива: (-135) Массив состоит из одного элемента. Конец сортировки Отсортированный массив: (-135) Выполним проверку библиотечной функцией std::sort() Исходный массив: (-135) Массив после обработки std::sort(): (-135) Массивы идентичны.
2.	-18 79 -54 67 94	Запуск сортировки чёт-нечёт Состояние массива: (-18 79 -54 67 94) Состояние ЧЁТ-нечёт: Обмен элементов (-18, индекс 0) и (79, индекс 1) не требуется Обмен элементов (-54, индекс 2) и (67, индекс 3) не требуется Состояние НЕЧЁТ-чёт: ! Элементы (79, индекс 1) и (-54, индекс 2) меняются местами Состояние массива: (-18 -54 79 67 94) Обмен элементов (67, индекс 3) и (94, индекс 4) не требуется Состояние ЧЁТ-нечёт: ! Элементы (-18, индекс 0) и (-54, индекс 1) меняются местами Состояние массива: (-54 -18 79 67 94) ! Элементы (79, индекс 2) и (67, индекс 3) меняются местами Состояние массива: (-54 -18 67 79 94) Состояние НЕЧЁТ-чёт:

		Обмен элементов (-18, индекс 1) и (67, индекс 2) не требуется Обмен элементов (79, индекс 3) и (94, индекс 4) не требуется Состояние ЧЁТ-нечёт: Обмен элементов (-54, индекс 0) и (-18, индекс 1) не требуется Обмен элементов (67, индекс 2) и (79, индекс 3) не требуется Состояние НЕЧЁТ-чёт: Обмен элементов (-18, индекс 1) и (67, индекс 2) не требуется Обмен элементов (79, индекс 3) и (94, индекс 4) не требуется После двух проходов по массиву подряд (ЧЁТ-нечёт и НЕЧЁТ-чёт) не произошло ни одного обмена (Условие конца сортировки) Сортировка закончена Отсортированный массив: (-54 -18 67 79 94) Выполним проверку библиотечной функцией std::sort() Исходный массив: (-18 79 -54 67 94) Массив после обработки std::sort(): (-54 -18 67 79 94) Массивы идентичны.
3.	28 73 -54 46 83 52 12	Запуск сортировки чёт-нечёт Состояние массива: (28 73 -54 46 83 52 12) Состояние ЧЁТ-нечёт:

		Обмен элементов (28, индекс 0) и (73, индекс 1) не требуется Обмен элементов (-54, индекс 2) и (46, индекс 3) не требуется ! Элементы (83, индекс 4) и (52, индекс 5) меняются местами Состояние массива: (28 73 -54 46 52 83 12) Состояние НЕЧЁТ-чёт: ! Элементы (73, индекс 1) и (-54, индекс 2) меняются местами Состояние массива: (28 -54 73 46 52 83 12) Обмен элементов (46, индекс 3) и (52, индекс 4) не требуется ! Элементы (83, индекс 5) и (12, индекс 6) меняются местами Состояние массива: (28 -54 73 46 52 12 83) Состояние ЧЁТ-нечёт: ! Элементы (28, индекс 0) и (-54, индекс 1) меняются местами Состояние массива: (-54 28 73 46 52 12 83) ! Элементы (73, индекс 2) и (46, индекс 3) меняются местами Состояние массива: (-54 28 46 73 52 12 83) ! Элементы (52, индекс 4) и (12, индекс 5) меняются местами Состояние массива: (-54 28 46 73 12 52 83) Состояние НЕЧЁТ-чёт: Обмен элементов (28, индекс 1) и (46, индекс 2) не требуется ! Элементы (73, индекс 3) и (12, индекс 4) меняются местами Состояние массива: (-54 28 46 12 73 52 83)

		Обмен элементов (52, индекс 5) и (83, индекс 6) не требуется Состояние ЧЁТ-нечёт: Обмен элементов (-54, индекс 0) и (28, индекс 1) не требуется ! Элементы (46, индекс 2) и (12, индекс 3) меняются местами Состояние массива: (-54 28 12 46 73 52 83) ! Элементы (73, индекс 4) и (52, индекс 5) меняются местами Состояние массива: (-54 28 12 46 52 73 83) Состояние НЕЧЁТ-чёт: ! Элементы (28, индекс 1) и (12, индекс 2) меняются местами Состояние массива: (-54 12 28 46 52 73 83) Обмен элементов (46, индекс 3) и (52, индекс 4) не требуется Обмен элементов (73, индекс 5) и (83, индекс 6) не требуется Состояние ЧЁТ-нечёт: Обмен элементов (-54, индекс 0) и (12, индекс 1) не требуется Обмен элементов (28, индекс 2) и (46, индекс 3) не требуется Обмен элементов (52, индекс 4) и (73, индекс 5) не требуется Состояние НЕЧЁТ-чёт: Обмен элементов (12, индекс 1) и (28, индекс 2) не требуется Обмен элементов (46, индекс 3) и (52, индекс 4) не требуется

		Обмен элементов (73, индекс 5) и (83, индекс 6) не требуется После двух проходов по массиву подряд (ЧЁТ-нечёт и НЕЧЁТ-чёт) не произошло ни одного обмена (Условие конца сортировки) Сортировка закончена Отсортированный массив: (-54 12 28 46 52 73 83) Выполним проверку библиотечной функцией std::sort() Исходный массив: (28 73 -54 46 83 52 12) Массив после обработки std::sort(): (-54 12 28 46 52 73 83) Массивы идентичны.
4.	-2 -2 -2	Запуск сортировки чёт-нечёт Состояние массива: (-2 -2 -2) Состояние ЧЁТ-нечёт: Обмен элементов (-2, индекс 0) и (-2, индекс 1) не требуется Состояние НЕЧЁТ-чёт: Обмен элементов (-2, индекс 1) и (-2, индекс 2) не требуется После двух проходов по массиву подряд (ЧЁТ-нечёт и НЕЧЁТ-чёт) не произошло ни одного обмена (Условие конца сортировки) Сортировка закончена Отсортированный массив: (-2 -2 -2) Выполним проверку библиотечной функцией std::sort() Исходный массив: (-2 -2 -2)

		Массив после обработки std::sort(): (-2 -2 -2) Массивы идентичны.
5.	0 15	Запуск сортировки чёт-нечёт Состояние массива: (0 15) Состояние ЧЁТ-нечёт: Обмен элементов (0, индекс 0) и (15, индекс 1) не требуется Состояние НЕЧЁТ-чёт: После двух проходов по массиву подряд (ЧЁТ-нечёт и НЕЧЁТ-чёт) не произошло ни одного обмена (Условие конца сортировки) Сортировка закончена. Отсортированный массив: (0 15) Выполним проверку библиотечной функцией std::sort() Исходный массив: (0 15) Массив после обработки std::sort(): (0 15) Массивы идентичны.

Тестирование на некорректных данных.

Результаты тестирования на некорректных данных представлены в табл. 2.

Таблица 2 - результаты тестирования на некорректных данных

№ п/п	Входные данные	Выходные данные
1.	23 d 6	Присутствуют запрещенные символы Некорректный ввод
2.	-1 8 9 63	Присутствуют лишние пробелы Некорректный ввод

3.	+23 -15 f	Присутствуют запрещенные символы Некорректный ввод

Выводы.

Были изучены и реализованы пузырьковая сортировка оптимизированная и сортировка чёт-нечёт.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

```
#include <iostream>
#include <cstdlib>
#include <ostream>
#include <fstream>
#include <cstring>
#include <sstream>
#include <algorithm>

using namespace std;
struct Array{
    int * array;
    int size;
};
void printArray(int *arr, int &size, ostream &out); // функция печати массива на экран
void bubbleSortOptimized(int *array, int &arrSize, ostream &out); // оптимизированная сортировка пузырьком
void oddEvenSorting(int *array, int &arrSize, ostream &out); // сортировка чёт-нечёт
bool isCorrect(string &arrSeq); // проверка на корректность введённой строки
Array * copyArray(Array * obj); // функция копирования массива
Array *getArrayFromString(string &arrSeq); // получение массива из строки
void destroy(Array * obj); // удаление экземпляра структуры Array
bool isEqual(Array * first, Array * second); // проверка массивов на равенство

namespace file{
    ifstream inputFile;
    ofstream outputFile;
}

int main(){
```

```
    string path; // путь до файла ввода
    string arraySequence; // вводимая строка-массив
    stringstream myStream; // вспомогательный поток
    char inputFlag; // управляющий флаг для ввода
    char sortFlag; // управляющий флаг для сортировки
    char outputFlag; // управляющий флаг для вывода
    int size; // количество элементов массива
     cout << "Первый введённый Вами символ - выбор ввода, второй - выбор вывода, третий - выбор алгоритма сортировки\n"
           "1) 0 - ввод с консоли\n   1 - ввод с файла\n"
           "2) 0 - вывод в консоль\n   1 - вывод в файл\n"
            "3) B - оптимизированная сортировка пузырьком (английская буква)\n   "
           "E - сортировка чёт-нечёт (английская буква)\n   "
           "Примеры начального ввода: 00E, 01B, 10E, 11B\n"
            "\n!Порядок ввода!\nЧисла должны быть введены строго через один пробел.\nПосле того, как был введён последний элемент, пробелы вводить нельзя\n";
    cin >> inputFlag;
    cin >> outputFlag;
    cin >> sortFlag;
     if((inputFlag != '0' && inputFlag != '1') || (sortFlag != 'B' && sortFlag != 'E') || (outputFlag != '1' && outputFlag != '0'))
    {
        cerr << "Некорректный ввод\n";
        return 0;
    }

       switch(inputFlag){  //  определение,  откуда  будет  производиться считывание
        case '0':
            cout << "Введите через пробел:\n";
            cin.ignore();
            getline( cin, arraySequence); // считывание строки
            break;
        case '1':
            cout << "Введите путь до входного файла:\n";
            cin >> path;
```

```
                file::inputFile.open(path); // открытие потока вывода в
файл
            if(!file::inputFile.is_open()){
                cout << "Невозможно открыть файл на чтение\n";
                return 0;
            }
            getline(file::inputFile, arraySequence);
            file::inputFile.close();
            break;
        default:
            cerr << "Неизвестная ошибка\n";
            exit(1);
    }

    if(!isCorrect(arraySequence)){ //проверка строки на корректность
        cerr << "Некорректный ввод\n";
        return 0;
    }

    if(outputFlag == '1'){
        cout << "Введите путь до выходного файла:\n";
        cin >> path;
        file::outputFile.open(path);
    }
    ostream & out = outputFlag == '0' ? cout : file::outputFile;


       Array * arrayObject = getArrayFromString(arraySequence); //
пребразование строки в массив
    if(!arrayObject) {
        out << "Введена пустая строка. Сортировать нечего\n";
        return 0;
    }
      Array * copy = copyArray(arrayObject); // создание копии для
проверки библиотечной функцией std::sort() (нужно в тестировании)
    switch(sortFlag){
       case 'E':
            oddEvenSorting(arrayObject->array, arrayObject->size, out);
            break;
```

```
       case 'B':
             bubbleSortOptimized(arrayObject->array, arrayObject->size,
out);
            break;
            default:
            cerr << "Неизвестная ошибка\n";
            exit(1);
        }
        out << "\nОтсортированный массив: ";
            printArray(arrayObject->array, arrayObject->size, out); //
вывод массива на экран

                out << "\nВыполним проверку библиотечной функцией
std::sort()\n";
        out << "Исходный массив: ";
        printArray(copy->array, copy->size, out);
        out << "\n";
        std::sort(copy->array, copy->array + copy->size); // сортировка
библиотечной функцией
        out << "Массив после обработки std::sort(): ";
        printArray(copy->array, copy->size, out);
        out << "\n";
        if(isEqual(arrayObject, copy))
            out << "Массивы идентичны.\n";
        else
            out << "Массивы НЕ идентичны\n";

        destroy(arrayObject); // освобождение памяти
        destroy(copy); // освобождение памяти

    return 0;
}

void bubbleSortOptimized(int *array, int &arrSize, ostream &out) {
    out << "Запуск оптимизированной сортировки пузырьком.\n";
       int i = 0; // вспомогательный счётчик, который нужен для
оптимизированной версии алгоритма
     bool flag = true; // флаг, по которому устанавливается, завершена
ли сортировка
```

```
    out << "Состояние массива: ";
    printArray(array, arrSize, out);
    out << "\n\n";
    if(arrSize == 1){
        out << "Массив состоит из одного элемента. Конец сортировки\n";
        return;
    }
     while(flag){  // цикл, зависящий от флага, который устанавливается в 1 если во внутреннем цикле произошёл хоть один обмен
        flag = false;

        for(int j = 0; j < arrSize - i - 1; j++) {
            if (array[j] > array[j + 1]) { // условие обмена
                  std::swap(array[j], array[j+1]); // элементы меняются местами
                flag = true;
                  out << "! Элементы (" << array[j] << ", индекс " << j << ") и " << "(" << array[j+1] << ", индекс " << j+1 <<") меняются местами\n";
                out << "\nСостояние массива: ";
                printArray(array, arrSize, out);
                out << "\n\n";
            }
            else{
                  out << "Обмен элементов (" << array[j] << ", индекс " << j << ") и " << "(" << array[j+1] << ", индекс " << j+1 <<") не требуется\n";
            }
        }
        i++;
        if(!flag){
             out << "При текущем обходе не произошло ни одного обмена. (Условие конца сортировки)\n";
        }
    }
    out << "Сортировка закончена.\n";
}

void oddEvenSorting(int *array, int &arrSize, ostream &out) {
```

```
    // две переменные ниже необходимы для определения конца сортировки и служат для оптимизации алгоритма
    int checkChanges = 0; // переменная, фиксирующая количество обменов
    int delta = 0; // переменная, фиксирующая количество итераций
    out << "Запуск сортировки чёт-нечёт\n";
    out << "Состояние массива: ";
    printArray(array, arrSize, out);
    out << "\n\n";
    if(arrSize == 1){
        out << "Массив состоит из одного элемента. Конец сортировки";
        return;
    }
    char even = true;
    while(true) {
        switch(even){
            case true:
                out << "\nСостояние ЧЁТ-нечёт:\n";
                break;
            case false:
                out << "\nСостояние НЕЧЁТ-чёт:\n";
                break;
            default:
                break;
        }
            for (int j = even ? 0 : 1; j < arrSize-1; j += 2) { // состояния ЧЁТ-нечёт И НЕЧЁТ-чёт чередуются на каждой итерации внешнего цикла
            if (array[j] > array[j + 1]) { // условие обмена
                  out << "! Элементы (" << array[j] << ", индекс " << j << ") и " << "(" << array[j+1] << ", индекс " << j+1 <<") меняются местами\n";
                  std::swap(array[j], array[j+1]); // элементы меняются местами
                out << "Состояние массива: ";
                printArray(array, arrSize, out);
                out << "\n";
                checkChanges++;
            }
            else
```

```
                out << "Обмен элементов (" << array[j] << ", индекс "
<< j << ") и " << "(" << array[j+1] << ", индекс " << j+1 <<") не
требуется\n";

        }

        delta++;

         if(delta == 2 && checkChanges == 0){ // если после подряд двух
проходов по массиву подряд (ЧЁТ-нечёт и НЕЧЁТ-чёт) не произошло ни
одного обмена, следует завершить сортировку
             out << "\nПосле двух проходов по массиву подряд (ЧЁТ-нечёт
и НЕЧЁТ-чёт) не произошло ни одного обмена (Условие конца
сортировки)\n";
            break;
        }
          else if(delta == 2){ /*важно в случае продолжения сортировки
обнулить переменные-контроллеры, чтобы можно при следующих двух обходах
 * можно было понять, требуется ли продожать внешний цикл */
            delta = 0;
            checkChanges = 0;
        }
        if(even)
            even = false;
        else
            even = true;

    }
    out << "Сортировка закончена.\n";
}

bool isEqual(Array * first, Array * second){
      if(first == nullptr && second == nullptr) // если оба нулевые
указатели
        return true;
      else if ((first == nullptr && second != nullptr) || (second ==
nullptr && first != nullptr)) { //если один нулевой указатель, а другой
нет
        return false;
```

```
    }
    else {
         if (first->size != second->size) { // если размеры отличаются,
массивы однозначно не равны
            return false;
        }
    }
    bool equal = true;
    for(int i = 0; i < first->size; i++){ // поэлементное сравнение
        if(first->array[i] != second->array[i]){
            equal = false;
            break;
        }
    }
    return equal;
}

Array *getArrayFromString(string &arrSeq) {
    if(arrSeq.empty())
        return nullptr;
    auto * arrayObj = new Array;
    stringstream myStream; // открывается строковый поток
    myStream << arrSeq; // в него записывается введённая строка
    int indentCounter = 0;
    int size = arrSeq.size();
      for(int i = 0; i < size; i++) { // размер массива формируется
исходя из количества пробелов между элементами
        if (arrSeq[i] == ' ')
            indentCounter++;
    }
    arrayObj->size = ++indentCounter;
    arrayObj->array = new int[arrayObj->size];

     for(int i = 0; i < arrayObj->size; i++){  // из строкового потока
происходит поэлементная запись в массив
        myStream >> arrayObj->array[i];
    }
    return arrayObj;
}
```

```
bool isCorrect(string &arrSeq) {
    if(arrSeq.find("  ") != -1 || arrSeq[arrSeq.size() - 1] == ' ') {
        cerr << "Присутствуют лишние пробелы\n";
        return false;
    }
    int size = arrSeq.size();
    for(int i = 0; i < size; i++)
    {
        if(!isdigit(arrSeq[i]) && arrSeq[i] != ' ' && arrSeq[i] != '-')
{
            cerr << "Присутствуют запрещенные символы\n";
            return false;
        }
    }
    return true;
}

void printArray(int *arr, int &size, ostream &out) {
    out << "(  ";
    for(int i = 0; i < size; i++)
        out << arr[i] << "  ";
    out << ")";
}

Array * copyArray(Array * obj){
    if(obj == nullptr) // если пустой экземпляр, возвращается nullptr
        return nullptr;
    auto * copy = new Array; // иначе создаётся экземпляр
    copy->size = obj->size;
    if(copy->size == 0) {
        copy->array = nullptr;
    }
    else {
         copy->array = new int[obj->size]; // если массив у копируемого
объекта не пустой
             for (int i = 0; i < copy->size; i++) { // поэлементное
копирование
            copy->array[i] = obj->array[i];
```

```
        }
    }
 return copy; // возврат копии
}

void destroy(Array * obj){
    if(obj ==  nullptr)
        return;
    delete [] obj->array;
    delete obj;
}
```