

# **Лабораторная работа №7**

по дисциплине «Программирование на Си»

## **Обработка одномерных динамических массивов**

Кострицкий А. С., Ломовской И. В.

Москва — 2020 — TS2009212126

### **Содержание**

Цель работы . . . . .	1
Общее задание . . . . .	1
Варианты сортировки . . . . .	2
Варианты фильтрации . . . . .	3
Примечания . . . . .	3
Взаимодействие с системой тестирования . . . . .	4
Памятка преподавателя . . . . .	6

### **Цель работы**

Целью работы является знакомство студентов с использованием динамической памяти. Подразумевается закрепление на практике следующих навыков:

1. Выделение и освобождение динамической памяти.
2. Использование адресной арифметики для обработки одномерных массивов.
3. Использование указателей на функцию.
4. Использование указателей типа `void`.
5. Обработка текстовых файлов.
6. Организация корректной работы с ресурсами.
7. Использование в программе аргументов командной строки.

### **Общее задание**

Написать программу, которая упорядочивает данные в целочисленном массиве по возрастанию. Данные в массив считываются из текстового файла. Память под массив выделяется динамически. Количество элементов в файле не указано и определяется при первом проходе по текстовому файлу, во время второго прохода числа считываются в массив.

Требуется написать программу, которая, будучи вызванной из командной строки:

`app.exe IN_FILE OUT_FILE [f]`

где `IN_FILE` — имя файла с входными данными, `OUT_FILE` — имя файла с выходными данными, `f` — запрос фильтрации перед сортировкой;

1. Считает количество элементов, записанных в файле.
2. Выделяет память под соответствующее количество элементов.
3. Читает файл ещё раз, заполняя динамический массив.
4. Если указан параметр `f`, вызывает функцию-фильтр.
5. Сортирует массив с новыми элементами с помощью назначенного преподавателем алгоритма сортировки.
6. Записывает элементы отсортированного массива в текстовый файл.

Функция-фильтр работает следующим образом:

1. определяет количество элементов массива, которые удовлетворяют заданному условию;
2. выделяет память под соответствующее количество элементов;
3. копирует элементы, удовлетворяющие условию, из старого массива в новый.

Функция-фильтр имеет следующие название и прототип:

```
int key(const int *pb_src, const int *pe_src, int **pb_dst, int **pe_dst);
```

На вход функции-фильтру могут поступать некорректные данные.

Функция сортировки реализуется универсальной, т.е. имеет одинаковый прототип с функцией `qsort` из стандартной библиотеки и называется `mysort`.

Кроме того, необходимо сравнить время работы реализованного алгоритма сортировки и `qsort`. Постарайтесь найти примеры, когда ваша реализация лучше. Результаты представьте в виде графиков (рисунки в формате PNG или любом аналогичном векторном, оси подписаны). Графики разместить в Wiki на отдельной странице. Графики можно строить любыми средствами (например, в Excel, gnuplot).

## Варианты сортировки

1. *Сортировка выбором*: находится максимальный элемент массива и переносится в его конец; затем этот метод применяется ко всем элементам массива, кроме последнего (т.к. он уже находится на своем месте), и т.д.
2. *Сортировка обменом (метод пузырька)*: последовательно сравниваются пары соседних элементов  $x[k]$  и  $x[k+1]$  ( $k = 0, 1, \dots, n-2$ ) и, если  $x[k] > x[k+1]$ , то они переставляются; в результате наибольший элемент окажется на своём месте в конце массива; затем этот метод применяется ко всем элементам, кроме последнего, и т.д.
3. *Сортировка вставками*: пусть первые  $k$  элементов массива (от 0 до  $k-1$ ) уже упорядочены по неубыванию; тогда берётся  $x[k]$  и размещается среди первых  $k$  элементов так, чтобы упорядоченными оказались уже  $k+1$  первых элементов; этот метод повторяется при  $k$  от 1 до  $n-1$ .

4. *Модифицированная сортировка вставками:* Для поиска места вставки нового элемента используется двоичный поиск.
5. *Модифицированная сортировка пузырьком I:* Запоминайте, где произошёл последний обмен элементов, и при следующем проходе алгоритм не заходит за это место. Если последними поменялись  $i$ -ый и  $i+1$ -ый элементы, то при следующем проходе алгоритм не сравнивает элементы за  $i$ -м.
6. *Модифицированная сортировка пузырьком II:* нечётные и чётные проходы выполняются в противоположных направлениях: нечётные проходы от начала к концу, чётные – от конца массива к его началу. При нечётных проходах большие элементы сдвигаются к концу массива, а при чётных проходах – меньшие элементы двигаются к его началу.
7. *Модифицированная сортировка пузырьком III:* идеи первой и второй модифицированной сортировки пузырьком объединяются.

## Варианты фильтрации

1. В массиве остаются элементы расположенные между минимальным и максимальным элементами массива. Если минимальных и/или максимальных элементов несколько, из берется первый минимальный и первый максимальный элементы. Минимальный и максимальный элементы в отфильтрованный массив не попадают.
2. В массиве остаются элементы от нулевого до  $m$ -го, где  $m$  - индекс первого отрицательного элемента этого массива либо число  $n$  (размер исходного массива), если такого элемента в массиве нет. Т.е. отфильтрованный массив содержит элементы, расположенные перед первым отрицательным элементом, или весь исходный массив, если отрицательные элементы отсутствуют.
3. В массиве остаются элементы от нулевого до  $p$ -го, где  $p$  - индекс последнего отрицательного элемента этого массива либо число  $n$ , если такого элемента в массиве нет. (См. пояснения в пункте 2.)
4. В массиве остаются элементы, которые больше среднего арифметического всех элементов массива.
5. В массиве остаются элементы, которые больше суммы элементов расположенных за ним. Последний элемент в отфильтрованный массив не попадает никогда, потому что его не с чем сравнивать.

## Примечания

1. В методических целях запрещено использовать обращения к элементам массива с помощью квадратных скобок. Вместо указанного выражения используется выражение `*ra`, где `ra` — указатель на  $i$ -ый элемент массива (именно на  $i$ -ый элемент, а не выражение вида `*(ra + i)`). Также нельзя передавать как аргумент размер массива в элементах, если массив уже создан. Вместо этого предлагается использовать пару указателей: на первый элемент массива и на элемент массива, расположенный за последним. Ситуация когда эти указатели совпадают, означает пустоту обрабатываемого массива.
2. В методических целях запрещено использовать `realloc`.

3. Обработка пустого файла или получение пустого массива после фильтрации рассматриваются как ошибки.
4. Номером варианта студента считается назначенный преподавателем номер функции-фильтра, он же попадает в название папки с лабораторной. Номер варианта сортировки не указывается в названии папки с лабораторной.

## Взаимодействие с системой тестирования

1. Решение задачи оформляется студентом в виде многофайлового проекта. Для сборки проекта используется программа `make`, сценарий сборки `makefile` помещается под версионный контроль. В сценарии должны присутствовать цель `app.exe` — для сборки основной программы, и цель `unit_tests.exe` — для сборки модульных тестов.
2. В сценарии сборки рекомендуется обозначить, помимо прочих, следующие цели:
  - (a) `unit` — сборка и прогон модульных тестов.
  - (b) `func` — прогон функциональных тестов.
  - (c) `clean` — очистка генерируемых файлов.

3. Исходный код лабораторной работы размещается студентом в ветви `lab_LL`, а решение каждой из задач — в отдельной папке с названием вида `lab_LL_CC_PP`, где `LL` — номер лабораторной, `CC` — вариант студента, `PP` — номер задачи.

Пример: решения восьми задач седьмого варианта пятой лабораторной размещаются в папках `lab_05_07_01`, `lab_05_07_02`, `lab_05_07_03`, ..., `lab_05_07_08`.

4. Исходный код должен соответствовать оглашённым в начале семестра правилам оформления.
5. Если для решения задачи студентом создаётся отдельный проект в IDE, разрешается поместить под версионный контроль файлы проекта, добавив перед этим необходимые маски в список игнорирования. Старайтесь добавлять маски общего вида. Для каждого проекта должны быть созданы, как минимум, два варианта сборки: `Debug` — с отладочной информацией, и `Release` — без отладочной информации.
6. Для каждой программы ещё до реализации студентом заготавливаются и помещаются под версионный контроль в подпапку `func_tests` функциональные тесты, демонстрирующие её работоспособность.

Позитивные входные данные следует располагать в файлах вида `pos_TT_in.txt`, выходные — в файлах вида `pos_TT_out.txt`, аргументы командной строки при наличии — в файлах вида `pos_TT_args.txt`, где `TT` — номер тестового случая.

Негативные входные данные следует располагать в файлах вида `neg_TT_in.txt`, выходные — в файлах вида `neg_TT_out.txt`, аргументы командной строки при наличии — в файлах вида `neg_TT_args.txt`, где `TT` — номер тестового случая.

Разрешается помещать под версионный контроль в подпапку `func_tests` сценарии автоматического прогона функциональных тестов. Если Вы используете при автоматическом прогоне функциональных тестов сравнение строк, не забудьте проверить используемые кодировки. Помните, что `UTF-8` и `UTF-8(BOM)` — две разные кодировки.

Под версионный контроль в подпапку `func_tests` также помещается файл `readme.md` с описанием в свободной форме содержимого каждого из тестов. Вёрстка файла на языке Markdown обязательной не является, достаточно обычного текста.

Пример: восемь позитивных и шесть негативных функциональных тестов без дополнительных ключей командной строки должны размещаться в файлах `pos_01_in.txt`, `pos_01_out.txt`, ..., `neg_06_out.txt`. В файле `readme.md` при этом может содержаться следующая информация:

```
# Тесты для лабораторной работы №LL

## Входные данные
Целые a, b, c

## Выходные данные
Целые d, e

## Позитивные тесты:
- 01 - обычный тест;
- 02 - в качестве первого числа нуль;
...
- 08 - все три числа равны.

## Негативные тесты:
- 01 - вместо первого числа идёт буква;
- 02 - вместо второго числа идёт буква;
...
- 06 - вводятся слишком большие числа.
```

7. Рекомендуется задавать следующую структуру проекта:

- (a) Все файлы исходных кодов хранятся в подпапке `src`.
- (b) Все файлы заголовков хранятся в подпапке `inc`.
- (c) Для каждого модуля создаётся и помещается в подпапку `unit_tests` один файл с модульными тестами, имя которого повторяет имя модуля с префиксом «`check_`». Основная программа модульного тестирования носит название «`check_main.c`».
- (d) Функциональные тесты оформляются в соответствии с предыдущими пунктами.
- (e) Сценарий сборки и конечные приложения генерируются в корне проекта.
- (f) Все остальные генерируемые файлы, в том числе объектные файлы и файлы статистики `gsov`, создаются в подпапке `out`.

Пример: папка с проектом для лабораторной работы, состоящего из текста программы и двух модулей, будет иметь следующий вид:

```
/lab_LL_CC_PP/  
  app.exe  
  makefile  
  unit_tests.exe  
  /inc/  
    unit_a.h  
    unit_b.h  
  /out/  
    main.o  
    unit_a.o  
    unit_b.o  
  /src/  
    main.c  
    unit_a.c  
    unit_b.c  
  /func_tests/  
    ...  
  /unit_tests/  
    check_main.c  
    check_unit_a.c  
    check_unit_b.c
```

8. Для каждой подпрограммы должны быть подготовлены модульные тесты с помощью фреймворка check, которые демонстрируют её работоспособность.
9. Все динамические ресурсы, которые уже были Вами успешно запрошены, должны быть высвобождены к моменту выхода из программы. Для контроля можно использовать, например, программы Dr. Memory или valgrind.
10. Успешность ввода должна контролироваться. При первом неверном вводе программа должна прекращать работу с ненулевым кодом возврата.  
Обратите внимание, что даже в этом случае все динамические ресурсы, которые уже были Вами успешно запрошены, должны быть высвобождены.
11. Вывод Вашей программы может содержать текстовые сообщения и числа. Тестовая система анализирует только числа в потоке вывода, поэтому они могут быть использованы только для вывода результатов — использовать числа в информационных сообщениях запрещено.  
Пример: сообщение «**Input point 1:**» будет неверно воспринято тестовой системой, а сообщения «**Input point A:**» или «**Input first point:**» — правильно.
12. Если не указано обратное, числа двойной точности следует выводить, округляя до шестого знака после запятой.

## Памятка преподавателя

1. *Только для ЛРН<sup>7</sup>.* Соответствие реализованного студентом алгоритма сортировки выданному преподавателю не проверяется тестовой системой.

2. *Только для ЛР№7.* Неиспользование функции `realloc` студентом не проверяется тестовой системой.
3. *Только для ЛР№7.* Неиспользование студентом квадратных скобок не проверяется тестовой системой.
4. *Только для ЛР№7.* То, что студент работает с адресной арифметикой так, как требуется, не проверяется тестовой системой.
5. *Только для ЛР№7.* Совпадение структур и типов данных у студента и в задании не проверяется тестовой системой.