

Лабораторная работа №6

по дисциплине «Программирование на Си»

Обработка массива структур со статическими полями

TS2009071831

Москва — 2020

Содержание

Цель работы	1
Вариант №1	1
Вариант №2	2
Вариант №3	3
Примечания	3
Взаимодействие с системой тестирования	3
Памятка преподавателя	5

Цель работы

Закрепить на практике навыки обработки массива структур со статическими полями и текстовых файлов, в том числе:

1. чтение из текстового файла;
2. поиск в массиве структуры по признаку;
3. использование в программе аргументов командной строки.

Вариант №1

В текстовом файле хранится информация о кинофильмах, которая включает в себя название кинофильма, фамилию режиссёра и год выхода на экран. В названии кинофильма не более двадцати пяти символов. В фамилии кинорежиссёра не более двадцати пяти символов. Год выхода задаётся целым числом. Количество кинофильмов в самом файле не указано. Требуется написать программу, которая, будучи вызванной из командной строки:

`app.exe FILE FIELD [KEY]`

где `FILE` — имя файла, `FIELD` — анализируемое поле, `KEY` — значение ключа;

1. считывает информацию о кинофильмах в массив. Информация после чтения полей каждой структуры помещается в массив таким образом, чтобы он сразу же был упорядочен по указанному полю.

2. Если значение ключа не указано, выводит упорядоченный массив.
3. Если значение ключа указано, выполняет двоичный поиск¹ по полю и значению ключа. Если кинофильм с искомым значением ключа найден, программа выводит информацию о нём на экран, иначе программа выводит сообщение «Not found».

Возможные значения FIELD:

1. «title» — название кинофильма;
2. «name» — фамилия режиссёра;
3. «year» — год выхода.

Примеры валидного вызова программы:

1. `app.exe films.txt title`
2. `app.exe films.txt title CasinoRoyale`
3. `app.exe films title "Tinker Tailor Soldier Spy"`
4. `app.exe films.txt name "Marceau (Мапу)"`
5. `app.exe movies.txt name "John Howard Carpenter"`

Вариант №2

В текстовом файле хранится информация о предметах, которая включает в себя название предмета, его массу и объём. В названии предмета не более двадцати пяти символов. Масса и объём задаются вещественными числами. Количество предметов в файле не указано. Требуется написать программу, которая, будучи вызванной из командной строки:

```
app.exe FILE [PREFIX]
```

где **FILE** — имя файла, **PREFIX** — искомая начальная подстрока;

1. считывает информацию о предметах в массив.
2. Если подстрока не указана, сортирует информацию о предметах по возрастанию значения плотности, после чего выводит упорядоченный массив на экран.
3. Если значение подстроки равно «ALL», выводит информацию обо всех предметах.
4. Если подстрока указана и не равна «ALL», выводит на экран информацию о предметах, названия которых начинаются с указанной подстроки.

Примеры валидного вызова программы:

1. `app.exe items.txt`
2. `app.exe items.txt dog`

¹Со стороны тестовой системы гарантируется единственность искомой структуры при её существовании в массиве. На стороне студента при использовании устойчивой сортировки результат поиска даже при наличии нескольких подходящих структур всё ещё детерминирован — в зависимости от реализации разные студенты могут получать разные результаты на одинаковых входных данных, но в рамках одной программы будет выбираться всегда одна определённая структура. Поэтому в функциональных тестах можно использовать файлы с несколькими подходящими записями.

3. `app.exe stuff houn`
4. `app.exe stuff.txt sopra`
5. `app.exe items.txt aLL`
6. `app.exe items.txt ALL`

Вариант №3

В текстовом файле хранится информация о продуктах, которая включает в себя название продукта и цену. В названии продукта не более двадцати пяти символов. Цена задается целым числом. Количество продуктов указано в первой строке файла. Требуется написать программу, которая, будучи вызванной из командной строки:

`app.exe FILE P`

где `FILE` — имя файла, `P` — значение цены;

1. считывает информацию о продуктах в массив.
2. Выводит на экран информацию о продуктах, цена которых ниже значения `P`.

Примеры валидного вызова программы:

1. `app.exe goods.txt 1.3`
2. `app.exe products.txt 700`
3. `app.exe products 1e4`

Примечания

1. Каждое поле структуры в файле записано в отдельной строке.
2. Каждое поле выводимой на экран структуры выводится в отдельной строке.
3. После последнего поля последней структуры на экран печатается символ новой строки ради единообразия вывода.
4. Количество структур в статическом массиве не превышает пятнадцати.
5. Все алгоритмы сортировки должны обладать устойчивостью.
6. Регистр в строках учитывается.

Взаимодействие с системой тестирования

1. 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`.

2. Исходный код должен соответствовать оглашённым в начале семестра правилам оформления.

3. 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.
4. 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 ноль;
- ...
- 08 - все три числа равны.

```
## Негативные тесты:
```

- 01 - вместо первого числа идёт буква;
- 02 - вместо второго числа идёт буква;
- ...
- 06 - вводятся слишком большие числа.

5. Успешность ввода должна контролироваться. При первом неверном вводе программа должна прекращать работу с ненулевым кодом возврата.

6. Вывод программы может содержать текстовые сообщения и числа. Если не указано обратное, тестовая система анализирует 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.

7. Если не указано обратное, числа двойной точности следует выводить, округляя до шестого знака после запятой.

Памятка преподавателя

1. *Только для ЛР№6.* Устойчивость сортировки не может быть проверена тестовой системой, преподаватель должен сам проверить выбранный студентом алгоритм.
2. *Только для ЛР№6.* Алгоритм бинарного поиска не проверяется тестовой системой, преподаватель должен сам проверить, что выбранный студентом алгоритм поиска структуры является алгоритмом бинарного поиска.
3. *Только для ЛР№6.* Совпадение структур и типов данных у студента и в задании не проверяется тестовой системой.