Лабораторная работа №5

по дисциплине «Программирование на Си»

Обработка структур и файлов

Кострицкий А. С., Ломовской И. В.

Москва — 2022 — TS2205071209

Содержание

1 Цель работы

Целью лабораторной работы является знакомство студентов с обработкой файлов: текстовых, типизированных и двоичных общего вида.

Студенты должны получить и закрепить на практике следующие знания и умения:

1. Обрабатывать текстовые и двоичные файлы (открывать в различных режимах, читать и записывать информацию).
2. Организовывать корректную работу с ресурсами (в данном случае – файловыми описателями).
3. Анализировать информацию об ошибках с помощью функций стандартной библиотеки.
4. Использовать в программе аргументы командной строки
5. Описание структурного типа.
6. Обработка текстовых и двоичных файлов.
7. Организация корректной работы с ресурсами (файловые описатели).
8. Использование в программе аргументов командной строки.

1.1 Задача №1

Пользователь вводит целые числа. Признаком окончания ввода считать любой инвалидный ввод.

Прототип функции, которая реализует решение задачи, должен выглядеть следующим образом:

```
int process(FILE *f [, прочие выходные параметры]);
```

Функция process возвращает 0 в случае успешного решения задачи и отрицательный код ошибки в противном случае (например, -1 – входных данных нет и т.д.). Для каждого кода ошибки задаётся мнемоническое имя с помощью директивы define.

При решении любого варианта задачи 1 два цикла ввода и массивы не использовать.

Варианты

0. Найти наибольшее положительное из чисел, которые следуют за отрицательным числом (если пользователь вводит «1 2 -3 4 -6 5 -7 -8», максимум выбирается среди 4 и 5).

1. Найти первый и второй максимумы последовательности (возможно совпадающие).

2. Найти порядковый номер (позиция начинается с 1) максимального из чисел (если чисел с максимальным значением несколько, то должен быть найден номер первого из них).

3. Определить, сколько раз в последовательности чисел меняется знак (нуль считается положительным числом).

4. Найти количество чисел, которые больше своих «соседей», т.е. предшествующего и последующего.

5. Найти наибольшее число подряд идущих элементов последовательности, которые равны друг другу;

6. Найти наибольшую длину монотонного фрагмента последовательности (то есть такого фрагмента, где все элементы либо больше предыдущего, либо меньше);

7. Определить количество локальных максимумов в последовательности (Элемент последовательности называется локальным максимумом, если он строго больше предыдущего и последующего элемента последовательности. Первый и последний элемент последовательности не являются локальными максимумами.);

8. Определить наименьшее расстояние между двумя локальными максимумами последовательности (понятие локального максимума описано в пункте 7).

1.2 Задача №2

Написать программу, которая считывает из текстового файла вещественные числа и выполняет над ними некоторые вычисления согласно варианту. При решении задачи массивы не использовать. Имя файла берётся из аргументов командной строки. Решение любой из этих задач выполняется минимум за два просмотра файла.

Варианты

0. найти число, наиболее близкое по значению к среднему значению всех чисел;
1. найти количество чисел, значение которых больше среднего арифметического минимального и максимального чисел;
2. рассчитать дисперсию чисел (математическое ожидание и дисперсия рассчитываются отдельно);
3. проверить выполняется ли правило «трёх сигм» для чисел (если правило «трёх сигм» выполняется, выводится 1, если нет – 0);
4. найти среднее значение чисел, расположенных между минимальным и максимальным числами («между» – не по значению, а по расположению); предполагается, что минимум и максимум единственны.

1.3 Задача №3

В этой задаче нет вариантов, номера варианта в названии папки нет. Требуется написать программу, совершающую действие над двоичным файлом целых чисел int в ответ на вызов с ключом:

1. создавать файл и заполнять его случайными числами (аргументы в произвольной форме, например, app.exe c number file; создание не проверяется :));
2. выводить числа из файла на экран (app.exe p file);
3. упорядочивать числа в файле (app.exe s file).

Прежде чем приступать к реализации сортировки файла, необходимо реализовать функцию `get_number_by_pos`, которая по заданной позиции позволяет прочитать число в указанной позиции, и функцию `put_number_by_pos`, которая позволяет записать число на указанную позицию с затиранием. Функцию упорядочивания необходимо реализовать с использованием этих функций.

В начале файла, содержащего исходный код программы, должен располагаться многострочный комментарий, в котором необходимо указать детали реализации этой задачи: как минимум, выбранные целочисленный тип, алгоритм сортировки, «направление» упорядочивания.

1.4 Задача №4

Вариант, полнота его реализации и тип обрабатываемых файлов, *текстовый* или *двоичный*, определяются преподавателем.

При работе с текстовым файлом данные сначала загружаются в массив, затем обрабатываются, выводятся на экран и, при необходимости, сохраняются в файл. При работе с двоичным файлом данные обрабатываются без использования массива. В текстовом файле на каждой строке располагается значение одного поля структурного типа. Поля структурного типа описываются в том порядке, в котором перечислены в задании. Структурный тип используется без упаковки. При вводе информации для формирования переменной структурного типа значения полей вводится в том порядке, в котором указана в задании. При выводе информации из переменной структурного типа на экран или в текстовый файл

поля выводятся в том порядке, в котором указано в задании. Каждое поле располагается на отдельной строке.

В случае если ваше приложение запущено с неправильными или незнакомыми параметрами командной строки, приложение должно возвращать код ошибки 53.

Варианты

1. О каждом *студенте* известны:
   (a) фамилия, не более 25 символов;
   (b) имя, не более 10 символов;
   (c) оценки по четырём предметам, сохраняемые в виде массива целых беззнаковых в 32 бита.

   Требуется написать программу, совершающую действие в ответ на вызов с ключом:

   (a) Сортировка студентов по фамилии, студенты с одинаковыми фамилиями должны быть упорядочены по имени. Результат выводится на экран.

   Запуск программы для обработки текстового файла:

   ```
   ./app.exe st FILE
   ```

   Запуск программы для обработки двоичного файла:

   ```
   ./app.exe sb FILE
   ```

   (b) Вывод информации о студентах, фамилии которых начинаются с заданной подстроки, в другой файл.

   Запуск программы для обработки текстового файла:

   ```
   ./app.exe ft FILE_SRC FILE_DST SUBSTR
   ```

   Запуск программы для обработки двоичного файла:

   ```
   ./app.exe fb FILE_SRC FILE_DST SUBSTR
   ```

   (c) Удаление из файла студентов, чей средний балл меньше среднего балла по всему файлу. Результат записывается в тот же файл.

   Запуск программы для обработки текстового файла:

   ```
   ./app.exe dt FILE
   ```

   Запуск программы для обработки двоичного файла:

   ```
   ./app.exe db FILE
   ```

2. О каждом *товаре* известны:
   (a) наименование, не более 30 символов;
   (b) изготовитель, не более 15 символов;
   (c) цена единицы товара, сохраняемая в виде целого беззнакового в 32 бита;
   (d) количество единиц товара, сохраняемое в виде целого беззнакового в 32 бита.

   Требуется написать программу, совершающую действие в ответ на вызов с ключом:

(a) Сортировка товаров по убыванию цены за единицу товара, товары с одинаковой ценой должны быть упорядочены по убыванию по количеству единиц. Результат сохраняется в другой файл.

Запуск программы для обработки текстового файла:

```
./app.exe st FILE_SRC FILE_DST
```

Запуск программы для обработки двоичного файла:

```
./app.exe sb FILE_SRC FILE_DST
```

(b) Вывод на экран информации о товарах, наименование которых заканчивается на заданную подстроку.

Запуск программы для обработки текстового файла:

```
./app.exe ft FILE SUBSTR
```

Запуск программы для обработки двоичного файла:

```
./app.exe fb FILE SUBSTR
```

(c) Добавление информации о новом товаре в уже упорядоченную последовательность товаров так, чтобы упорядоченность не нарушилась. Информация о новом товаре запрашивается у пользователя. Результат записывается в тот же файл. Повторное использование сортировки не считается правильным решением задачи.

Запуск программы для обработки текстового файла:

```
./app.exe at FILE
```

Запуск программы для обработки двоичного файла:

```
./app.exe ab FILE
```

1.5 Примечания

1. В каждой программе следует выделить, по крайней мере, один модуль — проект обязан быть многофайловым.

2. *Статические массивы* следует отличать от *массивов переменной длины (англ. Variable Length Array, VLA)*. Во избежание случайного использования последних при компиляции программы необходимо указывать ключ `-Wvla`.

3. Для реализации каждой из задач этой лабораторной работы необходимо выделить несколько осмысленных функций. Необходимо предусмотреть обработку ошибочных ситуаций.

4. При вводе строк следует контролировать переполнение буфера.

2 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_PP_CC`, где LL — номер лабораторной, PP — номер задачи, CC — вариант студента. Если дана общая задача без вариантов, решение следует сохранять в папке с названием вида `lab_LL_PP`.

Пример: решения восьми задач седьмого варианта пятой лабораторной размещаются в папках `lab_05_01_07`, `lab_05_02_07`, `lab_05_03_07`, ..., `lab_05_08_07`.

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. Сборка проекта на сервере происходит с помощью компилятора gcc с ключами `std=c99`, `Wall`, `Werror`, `Wpedantic`, `Wextra`.

   При сборке проектов, в которых используются типы с плавающей точкой, дополнительно указываются флаги `Wfloat-equal` и `Wfloat-conversion`.

   При сборке проектов лабораторных работ, в которых запрещено использовать массивы переменной длины (VLA), дополнительно указывается флаг `Wvla`.

   Если в Вашей программе используются математические функции из стандартной библиотеки, в Linux команда компиляции Вашей программы должна включать ключ `lm`, указывающий компилятору на явную компоновку математической библиотеки, которая в Linux не добавляется по умолчанию, в отличие от оставшейся части стандартной библиотеки.

   Пример:

   ```
   gcc -std=c99 -Wall -Werror -o app.exe main.c -lm
   ```

5. Крайне рекомендуется для проверки с некоторой периодичностью дополнительно собирать проект с помощью компилятора clang с тем же набором флагов.

6. Советуем проводить анализ проекта с помощью одного или нескольких статических анализаторов, которые рассматриваются в рамках практикума. Помните, что рекомендации статанализатора нужно принимать или отвергать обоснованно.

7. Для каждой программы ещё до реализации студентом заготавливаются и помещаются под версионный контроль в подпапку `func_tests/data/` 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/scripts/` 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.
```

8. Если не указано обратное, успешность ввода должна контролироваться. При первом неверном вводе программа должна прекращать работу с ненулевым кодом возврата.

9. 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.

   Тестовая система вычленяет из потока вывода числа, обособленные пробельными символами.

   Пример: сообщения «a=1.043» и «a = 1.043.» будут неверно восприняты тестовой системой, а сообщения «a: 1.043» или «a = 1.043» — правильно.

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