Московский авиационный институт
(Национальный Исследовательский Институт)

Кафедра вычислительной математики и программирования.

Курсовая работа

по курсу «практикум на ЭВМ»

Задание №6:

Обработка последовательной файловой структуры на языке Си

Студент:	Суханов Е. А.
Группа:	М80-106Б
Преподаватель:	Дубинин А. В.
Оценка:	
Дата:	

Москва
2020

Оглавление

Введение

Цель. Изучить набор стандартных функций языка Си для работы с файлами. Работа с структурами данных.

Задание. Составить две программы на языке программирования Си.
Первая программа должна уметь:

- Создавать бинарный файл на основе текстовой таблицы;

Вторая программа должна уметь:

- Открывать бинарный файл и выполнять запрос: поиск авокадо с ценой ниже среднего;
- Принимать параметры:
  - -f [FILE] – указывает какой файл открыть, если этот параметр не указан, то использовать стандартный поток ввода;
  - -p [YEAR] – указывает по какому году производить запрос, если этот параметр не указан, то производить поиск по всем годам;
  - -h – выводит справку о программе;
- Выводить результат в выходной поток;

Программа будет работать с таблицей продаж авокадо. Таблица хранится в .csv файле (но сама программа не будет выполнять стандарт RFC 4180).

Теория

Файловая система

Файловая система задает способ организации, хранения, именования данных на каких-то носителях информации. Она позволяет абстрагироваться от низкого уровня работы с носителями информации и представляет интерфейс для работы с данными. Обычно данные группируются в файлы, а файлы находятся в каталогах, которые в свою очередь тоже где-то находятся. При этом образуется иерархическая система. Файловая система управляет доступом к файлам, определяет атрибуты файлов, максимальный размер и т. д.. Некоторые файловые системы не связаны с носителями информации напрямую, например существуют виртуальные и сетевые файловые системы.

Файл

Как уже было сказано выше: файлом называется именованный блок данных. В контексте файловой системы, файл имеет определенные атрибуты, например права на запись и чтение. Во многих операционных системах, например в UNIX, понятие файла более обширно: интерфейсы общения с устройствами, как физических, так и виртуальных; именованные каналы для общения между процессами; сокеты; и прочее, – представляются как файл.

Директория

Директории используются для группирования файлов по смыслу, для более быстрого поиска и упрощении работы с ними.

Файловый дескриптор

Чтобы работать с файлом, нужно уметь обращаться к нему. Для этого придуманы файловые дескрипторы. Файловым дескриптором (ФД) называется целое неотрицательное число, которое ассоциируется с каким-то файлом. По сравнению с «классическим» путем файла он имеет несколько достоинств:

- Быстрее работает, так как требуется обработать число, а не строчку;
- Если работу с файлом уже начали (получили ФД), то даже после переименования, удаления, изменения прав на файл, с ним все еще можно работать с помощью ФД. (Использовать путь уже не получится);
- Независимость от файловой системы;

Для каждого процесса (запущенной программы) по умолчанию создается три файловых дескриптора: 0, 1 и 2. С ними связаны файлы стандартного ввода, стандартного вывода и стандартного вывода ошибок соответственно.

Для работы с файлами ОС UNIX представляет набор системных вызовов. Список основных системных вызовов для работы с файлами:

- `int open(const char* path, int oflag, mode_t mode)`
  Получает доступ на чтение и/или запись к указанному файлу. Если файл не существует, то он может быть создан. Возвращает связанный с этим файлом ФД либо -1, если произошла ошибка;

  path задает путь к файлу.
  oflag задает режим открытия файла, представляет собой побитовое объединение флагов. Например O_RDONLY, O_WRONLY, O_RDWR – открывают существующий файл с правами на чтение, запись, чтение и запись соответственно. А если добавить к ним флаг O_CREAT, то в случае, если файл не существует, он будет создан.
  mode задает права доступа, в случае, если файл был только что создан (похоже на задание прав с помощью chmod);
- `int creat(const char* path, mode_t mode)`
  Создает новый файл, и возвращает файловый дескриптор для работы с новым файлом. Если файл уже существует, то его длина сокращается до 0, а права доступа и владельцы сохраняются прежними.
  path – задает путь до файла
  mode – права доступа к файлу;
- `int close(int fildes)`
  Закрывает файловый дескриптор fildes, разрывая связь с файлом. В случае успеха close возвращет 0, в случае неудачи – -1. Следует заметить, что файловые дескрипторы программы автоматически закрываются при выходе программы (при вызове функции eixt());
- `off_t lseek(int fildes, off_t offset, int whence)`
  Изменяет файловый указатель (смещение в файле). Например его можно установить в начало, в конец или в любое другое место файла.
  В случае успеха вызов возвращает положительное целое число, равное текущему значению файлового указателя;

fildes – файловый дескриптор
offset – новое смещение относительно whence
whence – указывает опорную точку, относительно которой будет считаться смещение. Доступные варианты:
SEEK_CUR – относительно текущего положения
SEEK_END – относительно конца
SEEK_SET – относительно начала

- `ssize_t read(int fildes, void* buf, size_t nbyte)`
  Считывает из файла, связанный с ФД fildes, nbyte байт в буфер buf. Вызов возвращает количество считанных байтов. Это число может быть меньше nbyte, например из-за конца файла или считывания из потока ввода. Поэтому это тоже нужно учитывать. Вызов смещает файловый указатель на кол-во успешно считанных байт;
- `ssize_t wrtie(int fildes, void* buf, size_t nbyte)`
  Записывает nbyte из буфера buf в файл, связанный с ФД fildes.
  Вызов возвращает количество успешно записанных байт, а так же смещает файловый указатель на кол-во успешно записанных байт;

Более подробно можно узнать в [2], либо вызвав man syscalls.

Данные вызовы представляют низкоуровневый интерфейс, представляемый ОС для работы с файлами. Он удобен, если требуется держать все под контролем. Однако, зачастую этого не требуется, поэтому стандартная библиотека языка Си имеет более высокоуровневый интерфейс для работы с файлами.

Файл stdio.h
Данный файл входит в стандартную библиотеку языка Си и предоставляет оболочку над низкоуровневыми функциями ввода вывода.
Достоинства:

- Независимость от платформы. Так как входит в стандартную библиотеку языка Си.
- Простота работы.

Данный файл определяет структуру FILE (файловый указатель или просто поток), которая хранит следующие параметры:

- файловый дескриптор

- текущую позицию в потоке
- индикатор конца файла
- индикатор ошибок
- указатель на буфер потока, если возможность

данная структура является аналогом ФД.

Определяются некоторые константы:

- EOF – Отрицательное число типа int, используемое для обозначения конца файла
- FILENAME_MAX – Размер массива char, достаточно большого, для помещения любого пути
- FOPEN_MAX – Количество файлов, которые могут быть открыты одновременно
- и др.

Перечислим популярные функции для работы с файлами:

- fopen – открывает файл и связывает с ним FILE
- fclose – закрывает файл
- fflush – освобождает буфер файла
- fgetc – считывает символ, на который указывает файловый указатель указанного потока
- fputc – записывает символ в указанный поток
- fgets – считывает строку из потока ввода
- fputs – записывает строку в поток ввода
- fread – считывает блок данных из файла
- fwrite – записывает блок данных в файл
- fseek – изменяет файловый указатель
- fprintf – форматированный ввод
- fscanf – форматированное считывание

- fileno – возвращает файловый дескриптор открытого файла
- feof – проверяет, был ли достигнут конец файла. (Не производит обращений к файлу, по этому нужно выполнить какие-то операции с ним, перед проверкой)

Как видно, stdio.h представляет больший функционал, а так же упрощает считывание значений и вывод (функции printf и scanf). Так же, для стандартных потоков ввода, вывода и вывода ошибок есть специальные переменные: stdin, stdout, stderr соответственно.

Описание программы

Разобьём задание на задачи

Реализация первой программы включает в себя

- Создание структуры, которая будет содержать в себе одну запись таблицы
- Считывание записи в структуру
- Записать записи в структуру

Реализация второй программы включает в себя

- Обработка флагов
- Выполнение запроса и вывод информации

Таблица

Таблица имеет следующие столбцы:

1. Номер записи; тип int
2. Date – дата наблюдения; тип массив char длиной 12
3. AveragePrice – Средняя цена одного авокадо; тип double
4. Total Volume – Общее количество продаж авокадо; тип double
5. 4046 – PLU код, для маленького авокадо; тип double
6. 4225 – PLU код, для среднего авокадо; тип double
7. 4770 – PLU код, для большого авокадо; тип double
8. Total Bags; тип double
9. Small Bags; тип double
10. Large Bags; тип double
11. Xlarge Bags; тип double
12. Type – обычный авокадо или органический; тип массив char длинной 32
13. Year – год; тип int
14. Region – город или район наблюдения; тип массив char длинной 64

Dataset для этой программы можно найти на kaggle [6].

Выделение модулей
Так как работать с таблицей надо как в первой, так и во второй программе, то реализацию структуры record (хранит в себе одну строчку таблицы), нужно вынести в отдельный модуль. Модуль содержит функции для считывания/записи одной записи из текстового файла, а так же функции для считывания/записи одной записи из нетекстового файла.

Первая программа называется avocado-converter. Ее функционал заключается в считывании потока ввода (текстовое представление таблицы) и записи в поток вывода (бинарное представление таблицы).

Вторая программа называется avocado-finder. Ее функционал заключается в выполнении запроса. Файл можно указать с помощью перенаправления входного потока или с помощью ключа -f. Программа выведет результат в поток stdout.

Тестирование программ
Тестирование производится на пустом файле, на фале, состоящим из одной записи, а так же на файле состоящим из трех записей. Файлы для проверки этого модуля находятся в data

Для проверки avocado-finder используются отдельные файлы, созданные с помощью первой программы.

Сборка проекта
Для упрощения сборки и тестирования будет использоваться cmake.

Алгоритм создания бинарного файла

1. Пока поток ввода не пустой:
    1. Считываем запись из потока ввода
    2. Записываем запись в поток вывода

Временная сложность составляет **O(*n*)**, где *n* – кол-во записей. Так как алгоритм использует цикл, и за одну итерацию считывает по одной записи.
Объемная сложность составляет **O(1)**, так как программа использует одну промежуточную переменную, хранящую запись.

Алгоритм выполнения запроса

- Находим среднюю цену
    1. Пока не конец файла
        1. Считываем запись из файла
        2. Если год записи подходит, то:
            1. Суммируем стоимость авокадо
            2. Увеличиваем на один счетчик количества записей
- Выводим записи с ценой ниже среднего
    1. Возвращаемся в начало файла
    2. Пока не конец файла
        1. Считываем запись из файла
        2. Если год записи подходит, то:
            1. Выводим в поток вывода запись в текстовом формате

Временная сложность алгоритма составляет $\mathbf{O(n) + O(n) = O(n)}$**.** Так как поиск средней цены работает за n итераций цикла + некоторая константа, а вывод записей требует прохода по файлу.

Объемная сложность алгоритма составляет **O(1).** Так как нужна временная переменная для хранения записи.

Заключение

Потоки обобщают понятие файла и позволяют использовать одну реализацию для общения как и с стандартным потоком ввода, вывода, так и с отдельным файлом, который может быть как настоящим файлом, так и файлом устройства. Например, можно подменить поток стандартного ввода/вывода на определенный файл, а программа не будет знать о том, что работает с файлами.

Что касается самого задания. Бинарный файл, создаваемый первым приложением не является переносимым, так как данные из оперативной памяти записываются в него «как есть», то есть, либо в LE, либо в BE прядке байт. Поэтому, если бинарный файл бы создан на компьютере с LE порядком, то на компьютере с BE порядком этот файл будет неверно интерпретирован. Эту проблему можно решить с помощью реализации своего сериализатора и десериализатора, который будет независим от прядка байт.

Список источников

1. Гайсарян С. С., Зайцев В. Е. Курс информатики: Учеб. Пособие. – М.: Изд-во Вузовская книга, 2012. – 424с.: ил.
2. Робачевский А. М. Операционная система UNIX. - СПб.: БХВ-Петербург, 2002. - 528 с.: ил.
3. Практикум по циклу дисциплин “Информатика”. Ч. II. 2012/13 уч. Года
4. Заголовочный файл: stdio.h http://cppstudio.com/cat/309/323/
5. Файловая система: https://en.wikipedia.org/wiki/File_system
6. avocado prices dataset: https://www.kaggle.com/neuromusic/avocado-prices?select=avocado.csv