Лабораторная работа № 7

Организация хранения данных в файлах (массивы)

Цель работы: Изучение организации хранения однородных структурированных данных в файлах и работы с ними при использовании языка программирования Си.

Теоретические сведения

Понятие файлв в вычислительной системе

В информатике используется следующее определение: файл – это упорядоченная совокупность данных, хранимая на диске и занимающая именованную область внешней памяти.

С точки срения вычислительной системы файл – это сущность, позволяющая получить доступ к какому-либо ресурсу вычислительной системы и обладающая рядом признаков, основные из которых:

- фиксированное имя (последовательность символов, число или что-то иное, однозначно характеризующее файл);
- определённое представление (формат) и соответствующие ему операции чтения/записи.

Формат файла может быть любым – от последовательности байт до базы данных со сложной структурной организацией.

Работа с файлами реализуется средствами операционной системы.

Ресурсами, доступными через файлы, в общем случае, может быть что угодно, представимое в цифровом виде. Это зависит от конкретной операционной системы, и чаще всего в перечень входят:

- области данных (в памяти или на диске);
- устройства (как физические, так и виртуальные);
- потоки данных (в частности, вход или выход процесса);
- сетевые ресурсы;
- объекты операционной системы.

Для того чтобы систематизировать порядок хранения файлов их объединяют в каталоги с древовидной структурой. В связи с этим файл определяет не только имя, но и путь к нему, то есть перечисление узлов дерева файловой системы, которые нужно пройти, чтобы получить доступ к файлу. Существует два вида пути: абсолютный путь – полный путь к файлу от корневого каталога файловой системы и относительный – путь от текущего узла дерева. Текущей узел дерева определяется для каждой программы и может быть изменен самой программой в процессе ее работы.

Операции с файлом

Можно выделить два типа операций с файлом – связанные с его открытием, и выполняющиеся без его открытия. Операции первого типа обычно служат для чтения/записи информации или подготовки к записи/чтению. Операции второго типа выполняются с файлом как с объектом файловой системы, в котором файл является неделимой единицей.

Обычно выделяют дополнительные сущности, связанные с работой с файлом:

- идентификатор файла (handler - хэндлер, descriptor – дескриптор). При открытии файла, операционная система возвращает число (или указатель на структуру), с помощью которого выполняются все остальные файловые операции. По их завершению файл закрывается, а хэндлер теряет смысл.

- файловый указатель (файловая позиция) - число, являющееся смещением относительно нулевого байта в файле. По этому адресу осуществляется чтение/запись, в случае, если вызов операции чтения/записи не предусматривает указание адреса. При выполнении

операций чтения/записи файловый указатель смещается на число прочитанных/записанных байт. Последовательный вызов операций чтения, таким образом, позволяет прочитать весь файл, не зная его размер. Если указатель достигает конца файла, то выставляется специальный признак достижения конца файла, при этом выполнение операции чтения возвращает ошибку.

- файловый буфер – операционная система осуществляет кэширование файловых операций в специальном буфере памяти. При закрытии файла буфер сбрасывается.

- режим доступа, в зависимости от потребностей программы, файл может быть открыт на чтение и/или запись. Кроме того, некоторые операционные системы предусматривают режим работы с текстовыми файлами. Режим обычно указывается при открытии файла.

- режим общего доступа, Возможна ситуация, когда несколько программ одновременно могут работать с файлом на запись и/или чтение. Для управление этим существуют режимы общего доступа или эксклюзивного доступа, указывающие на разрешение или запрета совместного доступа к файлу.

Операции, связанные с открытием файла

- открытие файла, обычно в качестве параметров передается имя файла, режим доступа и режим совместного доступа, а в качестве возвращаемого значения выступает специальный индентификатор - файловый хэндлер. Кроме того, обычно имеется возможность в случае открытия на запись указать на то, должен ли размер файла изменяться на нулевой.

- закрытие файла, в качестве аргумента выступает значение, полученное при открытии файла. При закрытии все файловые буферы сбрасываются.

- запись, в файл помещаются данные.

- чтение, данные из файла помещаются в область памяти.

- перемещение указателя - указатель перемещается на указанное число байт вперёд/назад или перемещается по указанному смещению относительно начала/конца. Не все файлы позволяют выполнение этой операции, например, указатель в файле ассоциированным с устройством потокового ввода-вывода не может перемещатся.

- сброс буферов - содержимое файловых буферов с не записанной в файл информацией записывается. Используется обычно для указания на завершение записи логического блока для надежного сохранения данных в файле на случай сбоя.

- получение текущего значения файлового указателя.

Операции, не связанные с открытием файла

Операции, не требующие открытия файла оперируют с его «внешними» признаками – размером, именем, положением в дереве каталогов. При таких операциях невозможно получить доступ к содержимому файла, файл является минимальной единицей деления информации. В зависимости от файловой системы, носителя информации, операционной системой часть операций может быть недоступна:

- удаление файла;

- переименование файла;

- копирование файла;

- перенос файла в другое место (каталог, носитель и т.п.);

- получение или изменение атрибутов файла.

Работа с файлами в языке Си

Так как средства файлового ввода-вывода зависят от операционной системы, в языке Си все операции ввода-вывода реализуются с помощью функций, находящихся в стандартных библиотеках для конткретной опреационной системы.

Основной из таких библиотек является stdio.h. В ней структура данных языка Си, описывающая поток данных, называется FILE. Она содержит внутреннюю информацию о состоянии соединения с ассоциированным файлом, включая информацию о позиции указателя, информацию о буферизации, индикатор конца файла и ошибки.

Открытие файла выполняется с помощью функции fopen, которая создает новый поток, ассоциированный с открываемым файлом. Данное действие может привести к созданию нового файла.

FILE * fopen (const char *filename, const char *opentype)

Функция fopen открывает поток для ввода/вывода в/из файла filename и возвращает указатель на поток. Filename, кроме имени файла, должна содержать абсолютный или относительный путь к файлу.

Агрумент opentype – это строка, управляющая режимом открытия файла, сосятощая из двух частей. Первая часть:

- "r" – файл открывается только для чтения (файл должен существовать);

- "w" – файл открывается только для записи (если файла нет, он создается, если - уже существует, то вся информация в нем стирается);

- "a" – файл открывается для добавления информации в его конец (файл должен существовать);

- "r+" – файл открывается для чтения и записи (файл должен существовать);

- "w+" – открывается пустой файл для чтения и записи (если файла нет, он создается, если уже существует, то вся информация в нем стирается);

- "a+" – файл открывается для чтения и добавления информации в его конец.

Вторая часть символ "t" или "b":

- "t" - текстовый режим, информация при вводе/выводе будет восприниматься последовательность символьных строк с кодировкой, которая зависит от конкретной операционной системы,

- "b" – двоичный режим, информация при вводе/выводе воспринимается как последовательность байт, интерпритация данных должна выполняется самой программой.

В случае ошибки (файл не найден, не может быть создан) fopen возвращает пустой указатель NULL.

Поток закрывается с помощью функции fclose, которая разрывает соединение между потоком и файлом. Буферизованный вывод записывается в файл, а любой буферизованный ввод отбрасывается. fclose возвращает 0 в случае успеха и EOF в случае ошибки, например ситуация когда нет места для записи данных.

```
int fclose (FILE *stream);
```

Очень важно зарывать файл в программе при завершении работы с ним, так в случае корректного завершения функции main – все открытые потоки автоматически и корректно закрываются, а в случае же аварийного завершения возможна потеря данных, которые были записаны в файл программой, но еще находились в буфере потока.

Для работы с бинарными файлами могут быть несколько различных функции. Наиболее применяемые: fread и fwrite. Эти функции позволяют

читать и записывать блоки данных любого типа. Прототипы этих функций следующие:

size_t fread (void * buffer, size_t size, size_t count, FILE * fp);

size_t fwrite (const void * buffer, size_t size, size_t count, FILE * fp);

Параметры: buffer – для fread – указатель на область памяти, в которую будут прочитаны данные из файла; для fwrite – из которой будут записываться днные в файл.

Счетчик count – определяет, сколько считывается и записывается элементов данных, причем длина каждого элемента в байтах равна size.

Функция fread возвращает количество прочитанных элементов. Если достигнут конец файла или произошла ошибка, то возвращаемое значение может быть меньше, чем счетчик.

Функция fwrite возвращает количество записанных элементов. Если ошибка не произошла, то возвращаемый результат будет равен значению счетчика.

Пример записи в файл массива вещественных чисел:

```
FILE * fp;
float *buff;

buff = new float[64];
for (int i=0;i<64;i++) buff[i] = (rand()%100) / 100;

fp = fopen("some_float.dat", "wb");
if (fp == NULL) {
   printf("Ошибка создания файла \n");
} else {
   fwrite(buff, sizeof(float), 64, fp);
   fclose(fp);
}

free(buff);
```

Остальные основные операции, связанные с файловым вводом-выводом, описаны в таблице ниже.

Функция	**Описание**
`int feof(FILE *`*`stream`*`)`	Получение признака конца файла. Если указатель текущей позиции файла

	установлен на конец файла, возвращается ненулевое значение; в противном случае возвращается нуль.
`long int ftell(FILE *stream)`	Получение текущего значения файлового указателя от начала файла. В случае двоичных потоков это значение равно количеству байтов от начала файла начиная с 0.
`int fseek(FILE *stream,` `  long int offset, int whence)`	Перемещение указателя - указатель перемещается на offset - указанное число байт вперёд/назад или перемещается по указанному числу относительно начала/конца. Whence указывает откуда осуществляется перемещение: SEEK_SET – от начала SEEK_CUR – от текущей позиции SEEK_END – от конца Не все файлы позволяют выполнение этой операции (например, файл, связанный с устройством потокового ввода/вывода, не может перематываться назад).
`int fflush(FILE *stream);`	Запись буфера - содержимое файлового буфера с еще незаписанной на носитель информацией записывается на носитель

Задание на лабораторную работу

1. Разработать программу создания динамического массива, заполнения его исходных данными для варианта задания, вывода данных на экран и записи данных в файл (двоичный режим).

2. Разработать программу чтения данных из файла, обработки данных, согласно варианту задания, и сохранения результата в новый файл (в двоичном режиме), с выводом исходных и результирующих данных.

Содержание отчета по лабораторной работе

– титульный лист;

– тема и цель работы; задание на лабораторную работу согласно варианту.

– ход выполнения работы:

- исходные тексты полученных программ на языке Си;
- снимки экранов с результатами выполнения программ;
- снимки экранов Hex-редактора, демонстрирующие исходный и результирующий файлы;
- выводы о проделанной работе.

Контрольные вопросы

1. Что такое файлы в вычислительной системе?
2. Какие бывают виды файлов?
3. Какие существуют системные операции с файлами?
4. Какие системные программные сущности связанны с файлами?
5. Как организуется хранение данных в файлах на двоичном уровне?
6. Какие есть операции по управлению вводом/вывода для файлов в языке программирования С?
7. Какие есть операции вводом/вывода для файлов в языке С?

Варианты для задания

1. Преобразовать файл данных с вещественными числами в новый файл с целыми числами, при помощи округления до ближайшего целого.
2. Преобразовать файл данных с вещественными числами в новый файл с целыми числами, при помощи отбрасывания дробной части.
3. Преобразовать файл данных с вещественными числами в новый файл с целыми числами, при помощи округления к большему числу.
4. Дан файл с целыми однобайтовыми числами. Получить новый файл содержаций разницу каждого исходного числа со средним арифметическим всех чисел.
5. Дан файл с целыми однобайтовыми числами. Получить новый файл содержаций числа, являющиеся разницой между соседними числами в исходном файле.

6. Дан файл с целыми двухбайтовыми числами, хранящимися в порядке LittleEndian. Получить новый файл содержаций числами в порядке BigEndian.

7. Дан файл с вещественными числами, хранящимися в порядке LittleEndian. Получить новый файл содержаций числами в порядке BigEndian.

8. Дан файл с целыми двухбайтовыми числами. Получить новый файл - вставить число K между всеми соседними исходными числами, которые имеют разные знаки.

9. Дан файл с целыми четырехбайтовыми числами. Получить новый файл - вставить число K перед всеми исходными числами, большими числа K.

10. Дан файл с вещественными числами. Получить новый файл из исходного - удалить числа, у которых целая часть четная.

11. Дан файл с целыми четырехбайтовыми числами. Получить новый файл с двухбайтовыми числами, содержащий только те исходные числа, которые помещаются в два байта без потери данных.

12. Дан файл с целыми однобайтовыми числами. Получить новый файл – переставить в каждом исходном числе биты в обратном порядке.

13. Дан файл с целыми двухбайтовыми числами. Получить новый файл – переставить в каждом исходном числе десятичные цифры числа в обратном порядке.

14. Дан файл с однобайтовыми целыми числами. Получить новый файл, содержащий исходные числа и после каждых 8 чисел их простейшую контрольную сумму.

15. Дан файл с целыми однобайтовыми числами. Получить новый файл с зашифрованными данными, содержаций числа равные результату функции «исключающее или» (xor) каждого исходного числа и числа K.