Лабораторная работа 8

Цель работы

Изучение принципов работы с текстовыми файлами.

Задание

Разработать программу и подпрограмму (подпрограммы), работающую с текстовым файлом и выполняющую действия согласно варианту задания

№	Задание
1	Подсчитать количество слов и определить и вывести на экран максимальное и минимальное слова и их длину.
2	Определить количество букв и цифр в файле (сколько раз в файле встречается каждая буква и каждая цифра) и среднее количество букв и цифр в строке.
3	Подсчитать количество слов разной длины.
4	Подсчитать количество строк и определить строку максимальной длины и вывести на экран количество строк в файле, самую длинную строку и ее длину.
5	Создать выходной файл на основе входного, удалив в каждой строке входного файла текст после первой точки.
6	Для входного файла подсчитать контрольные суммы строк – разности суммы кодов символов, стоящих на четных и на нечетных позициях строки.

Теоретическая часть

Ввод-вывод в языке Си

Файл – это именованная область внешней памяти. Файл имеет следующие характерные особенности:

1. имеет имя на диске, что дает возможность программам работать с несколькими файлами;
2. длина файла ограничивается только емкостью диска.

Особенностью языка Си является отсутствие в нем структурированных файлов. Все файлы рассматриваются как неструктурированная последовательность байтов. При таком подходе понятие файла распространяется и на различные устройства. Одни и те же функции используются как для обмена данными с файлами, так и для обмена с устройствами.

Потоковый ввод-вывод

На уровне потокового ввода-вывода обмен данными производится побайтно, т. е. за одно обращение к устройству (файлу) производится считывание или запись фиксированной порции данных (512 или 1024 байта). При вводе с диска или при считывании из файла данные помещаются в буфер, а затем побайтно или порциями передаются программе пользователя. При выводе в файл данные также накапливаются в буфере, а при заполнении буфера записываются в виде единого блока на диск. Буферы реализуются в виде участков оперативной памяти. Таким образом, поток – это файл вместе с предоставленными средствами буферизации. Функции библиотеки Си, поддерживающие обмен данными на уровне потока позволяют обрабатывать данные различных размеров и форматов. При работе с потоком можно:

1. Открывать и закрывать потоки (при этом указатели на поток связываются с конкретными файлами);
2. Вводить и выводить строки, символы, форматированные данные, порции данных произвольной длины;
3. Управлять буферизацией потока и размером буфера;
4. Получать и устанавливать указатель текущей позиции в файле.

Прототипы функций ввода-вывода находятся в заголовочном файле stdio.h, который также содержит определения констант, типов и структур, необходимых для обмена с потоком.

Открытие и закрытие потока

Прежде, чем начать работать с потоком, его надо инициировать, т. е. открыть. При этом поток связывается со структурой предопределенного типа FILE, определение которой находится в файле stdio.h. При открытии потока возвращается указатель на поток, т. е. на объект типа FILE. Указатель на поток должен быть объявлен следующим образом:

```
#include <stdio.h>
. . . . . .
FILE*f;    /* указатель на поток */
```

Указатель на поток приобретает значение в результате выполнения функции открытия потока:

```
FILE* fopen(const char*filename,const char*mode);
```

где

const char*filename – строка, которая содержит имя файла, связанного с потоком;
const char*mode – строка режимов открытия файла.

Например, `f=fopen("t.txt","r"); -` здесь t.txt – имя файла, r – режим открытия файла.

Файл, связанный с потоком, можно открыть в одном из 6 режимов:

Режим	Описание режима открытия файла
r	Файл открывается для чтения, если файл не существует, то возвращается ошибка.
w	Файл открывается для записи, если файл не существует, то он будет создан, если файл уже существует, то вся информация из него стирается.
a	Файл открывается для добавления, если фай не существует, то он будет создан, если существует, то информация из него не стирается, можно выполнять запись в конец файла
r+	Файл открывается для чтения и записи, изменить размер файла нельзя, если файл не существует, то возвращается ошибка.
w+	Файл открывается для чтения и записи, если файл не существует, то он будет создан, если файл уже существует, то вся информация из него стирается.
a+	Файл открывается для чтения и записи, если фай не существует, то он будет создан, если существует, то информация из него не стирается, можно выполнять запись в конец файла

Поток можно открывать в текстовом (t) или двоичном режиме(b). В текстовом режиме поток рассматривается как совокупность строк, в конце каждой строки находится управляющий символ '\n'. В двоичном режиме поток рассматривается как набор двоичной информации. Текстовый режим устанавливается по умолчанию. В файле stdio.h определена константа EOF, которая сообщает о конце файла.

При открытии потока могут возникать следующие ошибки:
- файл, связанный с потоком не найден (при чтении из файла);
- диск заполнен (при записи);
- диск защищен от записи (при записи) и т. п.

В этих случаях указатель на поток приобретет значение NULL.

Для вывода об ошибке при открытии потока используется стандартная библиотечная функция

```
void perror (const char*s);
```

Эта функция выводит на экран строку символов, на которую указывает указатель s, за этой строкой размещается двоеточие пробел и сообщение об ошибке. Текст сообщения выбирается на основании номера ошибки. Номер ошибки заносится в переменную int errno (определена в заголовочном файле errno.h).

После того как файл открыт, в него можно записывать информацию или считывать ее, в зависимости от режима.

Открытые файлы после окончания работы рекомендуется закрыть явно. Для этого используется функция:

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

Изменить режим работы с файлом можно только после закрытия файла.

Стандартные файлы и функции для работы с ними

Когда программа начинает выполняться, автоматически открываются несколько потоков, из которых основными являются:
- стандартный поток ввода (stdin);
- стандартный поток вывода (stdout);
- стандартный поток вывода об ошибках (stderr).

По умолчанию потоку stdin ставится в соответствие клавиатура, а потокам stdout и stderr - монитор. Для ввода-вывода с помощью стандартных потоков используются функции:

- getchar()/putchar() – ввод-вывод отдельного символа;
- gets()/puts() – ввод-вывод строки;
- scanf()/printf() – форматированный ввод/вывод.

Символьный ввод-вывод

Для символьного ввода-вывода используются функции:

- **int fgetc(FILE*fp)**, где fp – указатель на поток, из которого выполняется считывание. Функция возвращает очередной символ в форме int из потока fp. Если символ не может быть прочитан, то возвращается значение EOF.
- **int fputc(int c, FILE*fp)**, где fp – указатель на поток, в который выполняется запись, c – переменная типа int, в которой содержится записываемый в поток символ. Функция возвращает записанный в поток fp символ в форме int . Если символ не может быть записан, то возвращается значение EOF.

Строковый ввод-вывод

Для построчного ввода-вывода используются следующие функции:

- **char* fgets(char* s,int n,FILE* f)**, где char*s – адрес, по которому размещаются считанные байты, int n – количество считанных байтов, FILE* f – указатель на файл, из которого производится считывание. Считывание байт заканчивается после передачи n-1 байтов или при считывании символа "новая строка" (\n), который так же передается в буфер.
  Строка в любом случае заканчивается '\0'. При успешном завершении считывания функция возвращает указатель на прочитанную строку, при неуспешном – NULL.
- **int puts(char* s, FILE* f),** где char*s – адрес, из которого берутся записываемые в файл байты, FILE* f – указатель на файл, в который производится запись.
  Символ конца строки ('\0') в файл не записывается. Функция возвращает EOF, если при записи в файл произошла ошибка, при успешной записи возвращает неотрицательное число.

Форматированный ввод-вывод

В некоторых случаях информацию удобно записывать в файл без преобразования, т. е. в символьном виде пригодном для непосредственного отображения на экран. Для этого можно использовать функции форматированного ввода-вывода:

- **int fprintf(FILE *f, const char*fmt, ...)** , где FILE*f – указатель на файл, в который производится запись, const char*fmt – форматная строка, … – список переменных, которые записываются в файл. Функция возвращает число записанных символов.
- 2) **int fscanf(FILE *f, const char*fmt, par1,par2, ...)** , где FILE*f – указатель на файл, из которого производится чтение, const char*fmt – форматная строка, par1,par2, … – список переменных, в которые заносится информация из файла. Функция возвращает число переменных, которым присвоено значение.

Рекомендации по выполнению лабораторной работы

При выполнении операций с текстовыми файлами "плохим стилем" программирования считается использование функций считывания символов в случае, если задание подразумевает работу со строками.

При считывании строк из текстового файла с помощью функции fgets() следует задавать входной буфер такого размера, чтобы в него гарантированно поместилась строка целиком, считая завершающий символ '\n' (который может отсутствовать только в последней строке файла) и нулевой байт в конце строки. Если учесть, что тестовые файлы создаются с помощью тестовых редакторов, то размер буфера 256 байт вполне достаточен для считывания строк.

Пример программы, считывающий строки из файла.

```
#include <stdio.h>

void main()
{
char s[256];
FILE *in;
     in = fopen("myfile.txt","rt");
     if(in != NULL)
     {
```

```
            while(fgets(in,256,s) != NULL)      /* читаем до конца файла */
            {
            /*
            здесь производится обработка считанной строки
            */
            }
            fclose(in);             /* закроем файл */
      }
}
```

При определении длин считанных строк нужно учитывать, что в конце строки находится символ '\n' (за исключением, может быть, последней строки файла), поэтому этот символ НЕ ДОЛЖЕН учитываться.

Содержание отчета

Отчет по лабораторной работе должен содержать:

- задание лабораторной работы, соответствующее варианту
- структурную схему алгоритма программы и подпрограммы (подпрограмм)
- текст программы
- результаты работы программы