**Files**

Variables of the programs die at the end of the program. They do not remain after the program termination. We use files to persist data even after the program exits. The file size is not limited by the maximum size of a process - a program in execution. The data in the files can be shared between processes. The data in a file can be structured – stored in the form of rows and columns, filed name and values. The data can also be unstructured like a story. A program in C is itself data for the ‘C’ compiler. The keyboard and the screen are also considered as files.

A file is maintained by the operating system. The operating system dictates the naming convention of a file. Ex: /*home/*kumar/C/ex1.c in unix, D:[\\home\kumar\ex1.c](smb://home/kumar/ex1.c) in Microsoft Windows. This is referred to as the physical filename.

In a program in a programming language like ‘C’, we use an identifier to refer to a file. This is called the logical name or the file handle.

There is an opaque type called FILE – which is a typedef. The type FILE could vary from one implementation to another. We use FILE\* in our program and never FILE so that our program will not depend on the layout of the type FILE.

In a program, we may want to inspect what a file contains – that is called reading. We may want to create a new file – that is called writing. We may want to add at the end of a file – that is called appending. These are called modes of file opening. The modes are “r”, “w” and “a” respectively.

We connect the physical filename, the logical filename(FILE\* ) and the mode by using a function called fopen. The function fopen returns a pointer of type FILE\*. We shall use this pointer to access the file.

The file opening could fail for a number of reasons. If we try to open a file for reading and the file is not found or we may have no permission to read the file or any other error, fopen returns NULL. We must check this return value before proceeding.

Initializing a file pointer does not mean that the whole file is made available in memory. We may use functions like fscanf, getc to read some bytes from the file.

Opening of a file causes creation of data structures in the memory. We should remove all these when the file is no more required. We do that by closing the file.

This first program 1\_intro.c shows how to open a file and check whether the file opening succeeds. It also shows how to release the resources by closing the file.

//

// - size not limited by the virtual address space of the process

// - data created by one program may be used by the other

// - data could be structured or unstructured

// - a program in C is data for a compiler

// - stdin and stdout are also logical files

#include <stdio.h>

int main()

{

// fopen : connects physical filename with a logical filename

// physical filename : name as in the OS

// logical filename in C is a pointer to a opaque type : FILE; is an identifier

// mode : we open the file for reading or writing or appending ...

// check man pages for fopen, fclose

FILE \*fp;

#if 0

//fp = fopen("junk.dat", "r"); // no default mode ;

printf("fp ? %d\n", fp == NULL); // 1

fclose(fp); // crashing; closing an unopened file

#endif

fp = fopen("1\_intro.c", "r"); // acquire resources

printf("fp %p\n", fp);

fclose(fp); // release resources

}

In the second example 2\_read.c, we show how to read a file. This file contains a single number followed by so many numbers. We want to add all these numbers programmatically.

This is the content of numbers2.dat

5

10 30 20 50 40

#include <stdio.h>

// read a file which has

// line 1: # of numbers

// line 2: so many numbers

void disp(int\* x, int n)

{

for(int i = 0; i < n; ++i)

{

printf("%d\t", x[i]);

}

printf("\n");

}

int main()

{

FILE \*fp;

fp = fopen("numbers2.dat", "r");

int n;

fscanf(fp, "%d", &n);

int a[n];

for(int i = 0; i < n; ++i)

{

fscanf(fp, "%d", &a[i]);

}

disp(a, n);

fclose(fp);

}

The above program opens the file numbers2.dat for reading. We do not check for errors for brevity. We read in a formatted way from the file using fscanf. fscanf is similar to scanf but for the first argument which is a file handle. We read the number of numbers into a variable(n), use this to create a variable length array(VLA) and read n numbers from the file, display them, then close the file.

Sometimes while processing a file, we do not know how many elements it contains. If we keep reading the file, we will definitely reach the end of file at some point. The input functions like fscanf or getc will return a value to indicate that the end of file has been reached. This value could differ from implementation to implementation. The compiler will provide that as a hash defined constant EOF. So, while reading in a conditional loop, we check for the return value being EOF to terminate the loop.

As we do not know the number of elements to be processed, we are constrained to make a fairly big array.

The data file numbers3.dat contains the following.

10 30 20 50 40

#include <stdio.h>

// read a file which has a line of numbers - count not specified

// max say 100

void disp(int\* x, int n)

{

for(int i = 0; i < n; ++i)

{

printf("%d\t", x[i]);

}

printf("\n");

}

int main()

{

FILE \*fp;

fp = fopen("numbers3.dat", "r");

int n = 0;

const int MAX = 100;

int a[MAX];

// EOF : is an indication that the end of file has been reached; is a number

**while(fscanf(fp, "%d", &a[n]) != EOF)**

{

++n;

}

disp(a, n);

fclose(fp);

}

This program 4\_read.c is similar to the earlier example, but for the fact that there is reading before the condition of the loop and reading at the end of the block. This is characteristic of sentinel loops. EOF acts like a sentinel.

#include <stdio.h>

// read a file which has a line of numbers - count not specified

// max say 100

void disp(int\* x, int n)

{

for(int i = 0; i < n; ++i)

{

printf("%d\t", x[i]);

}

printf("\n");

}

int main()

{

FILE \*fp;

fp = fopen("numbers3.dat", "r");

int n = 0;

const int MAX = 100;

int a[MAX];

**fscanf(fp, "%d", &a[n]);**

while( ! feof(fp) )

{

++n;

**fscanf(fp, "%d", &a[n]);**

}

disp(a, n);

fclose(fp);

}

This example 5\_create.c creates a newfile. It sorts the given input and writes to the file.

Let us examine the lines associated with the file creation and writing.

FILE \*fpout;

The above statement declares a variable of FILE\* type.

fpout = fopen("sorted3.dat", "w");

The above line opens the file for writing using the mode “w”.

fprintf(fpout, "%d\t", a[i]);

The above statement allows formatted output. This is similar to printf but for the first argument which is a file pointer.

#include <stdio.h>

// read a file which has a line of numbers - count not specified

// max say 100

void disp(int\* x, int n)

{

for(int i = 0; i < n; ++i)

{

printf("%d\t", x[i]);

}

printf("\n");

}

void sort(int a[], int n)

{

int i; int j;

for(i = 0; i < n - 1; ++i)

{

for(j = i + 1; j < n; ++j)

{

if(a[j] < a[i])

{

int temp = a[i]; a[i] = a[j]; a[j] = temp;

}

}

}

}

int main()

{

FILE \*fp;

fp = fopen("numbers3.dat", "r");

int n = 0;

const int MAX = 100;

int a[MAX];

fscanf(fp, "%d", &a[n]);

while( ! feof(fp) )

{

++n;

fscanf(fp, "%d", &a[n]);

}

fclose(fp);

sort(a, n);

FILE \*fpout;

fpout = fopen("sorted3.dat", "w");

for(int i = 0; i < n; ++i)

{

fprintf(fpout, "%d\t", a[i]);

}

fprintf(fpout, "\n");

fclose(fpout);

}

The data file votes.csv has names(very unusual ones though!) and votes secured. The two fields are separated by comma. Such a file is normally referred to as CSV files – Comma Separated Values file. How do we find who bagged the highest score?

The challenge is to read lines which have comma separated values.

There are two ways of doing this.

1. read character by character, populate the required fields in a structure, process the separator comma as required

2. read the whole line, split the line based on comma using a function called strtok.

Strategy 1:

we use getc to read a character at a time. This is similar to getchar. In fact getchar calls getc with stdin as its argument. We use a status variable in\_name to indicate whether we are still reading characters from the name field. When we encounter comma, we terminate the string in the array and we make in\_name false. We then process the integer value digit by digit until a newline is encountered. We repeat this operation for every line of the file.

#include <stdio.h>

// file structure:

// string, int

// find the highest

// strategy 1: read character by character

int main()

{

FILE \*fp;

char ch;

char name[100];

int votes = 0;

int i = 0;

int in\_name = 1;

fp = fopen("votes.csv", "r");

while((ch = getc(fp)) != EOF)

{

if(ch == ',')

{

in\_name = 0;

name[i] = '\0';

}

else if(ch == '\n')

{

printf("name: %s votes: %d\n", name, votes);

in\_name = 1;

votes = 0;

i = 0;

}

else if(in\_name)

{

name[i++] = ch;

}

else if(ch >= '0' && ch <= '9')

{

votes = votes \* 10 + (ch - '0');

}

}

fclose(fp);

}

Strategy 2:

We read the whole line using the function fgets. Then we use ftok to break into parts based on the separator or delimter.

strtok – check the man page – takes two arguments – a source string or NULL and the delimiter string.

The first time strtok is called, the string to be split is passed as the first argument. In subsequent calls, NULL is passed to indicate that strtok should keep splitting the same string for the next component.

strtok returns a pointer to a NULL terminated string which the caller should copy as each call will return a new string. Storing just the pointer will not work.

strtok returns NULL when the string has been completely processed.

You may want to think how such a function could be implemented.

#include <stdio.h>

#include <string.h>

// file structure:

// string, int

// find the highest

// strategy 2: read characters together; use strtok to break; may use atoi to convert str to int

int main()

{

FILE \*fp;

char ch;

char name[100];

int votes;

const int SIZE = 200;

char buffer[SIZE];

fp = fopen("votes.csv", "r");

while(! feof(fp) )

{

fgets(buffer, SIZE - 1, fp);

printf("buffer : %s\n", buffer);

char \*p = strtok(buffer, ",");

printf("name: %s\n", p);

while( (p = strtok(NULL, ",")) != NULL)

{

printf("votes: %s\n", p);

}

}

fclose(fp);

}

That is about files.