ICS 2305 : SYSTEMS PROGRAMMING

FILES HANDLING AND MANAGEMENT

Karanja Mwangi

Lesson objective

- **At the end of this class you will**
  - Appreciate the concepts of File Management
- Describe and implement **Open – Read – Write – Close file operations in C programming**

File Management-1

- A FILE IS - A named collection of data, stored in secondary storage (typically).
- How is a file stored? – Stored as sequence of bytes, logically contiguous (may not be physically contiguous on disk).
- Two kinds of files: –
  - **Text** :: contains ASCII codes only Text files are the normal **.txt** files. You can easily create text files using any simple text editors such as Notepad.
  - **Binary** :: can contain non-ASCII characters such as Image, audio, video, executable, etc.

  To check the end of file here, the file size value (also stored on disk) needs to be checked. The last byte of a file contains the end-of-file character (EOF), with ASCII code 1A (hex). – While reading a text file, the EOF character can be checked to know the end.

File Management -2

File Operations

- In C, you can perform 5 major operations on files, either text or binary:
  - Creation of a file
  - Opening a file
  - Reading a file
  - Writing to a file
  - Closing a file

File Management -3

File Operations

- In C, When working with files, you need to declare a pointer of type file. This declaration is needed for communication between the file and the program.

  FILE *fpointer;

File Management -4 Basic Functions for File operations in C

function	purpose
fopen ()	Creating a file or opening an existing file
fclose ()	Closing a file
fprintf ()	Writing a block of data to a file
fscanf ()	Reading a block data from a file
getc ()	Reads a single character from a file
putc ()	Writes a single character to a file
getw ()	Reads an integer from a file
putw ()	Writing an integer to a file
fseek ()	Sets the position of a file pointer to a specified location
ftell ()	Returns the current position of a file pointer
rewind ()	Sets the file pointer at the beginning of a file

Opening a file - for creation and edit

- Opening a file is performed using the fopen() function defined in the stdio.h header file in C.
- The syntax for opening a file in standard I/O is:

FILE *fp;

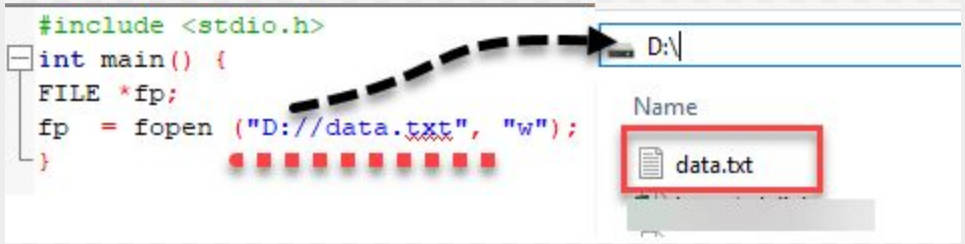
fp = fopen ("file_name", "mode");

Example

#include <stdio.h>

int main()

{ FILE *fp; fp = fopen ("D://data.txt", "**w**"); }



Note

w is the mode)

Opening a file – Modes

MODE	Meaning
r	We use it to open a text file in reading mode
w	We use it to open or create a text file in writing mode
a	We use it to open a text file in append mode
r+	We use it to open a text file in both reading and writing mode
w+	We use it to open a text file in both reading and writing mode
a+	We use it to open a text file in both reading and writing mode
rb	We use it to open a binary file in reading mode
wb	We use it to open or create a binary file in writing mode
ab	We use it to open a binary file in append mode
rb+	We use it to open a binary file in both reading and writing mode
wb+	We use it to open a binary file in both reading and writing mode
ab+	We use it to open a binary file in both reading and writing mode

Closing a file

- Closing a File
  - The **fclose()** function is used to close an already opened file.

    fclose (file_pointer);
  - The fclose function takes a file pointer as an argument.
  - Here fclose() function closes the file and returns zero on success, or EOF (End of File) if there is an error in closing the file. This EOF is a constant defined in the header file stdio.h.
  - **Example**

```
FILE *fp;
fp = fopen ("data.txt", "r");
fclose (fp);
```

Input/Output operation on File – Writing to a file

- In C, when you write to a file, newline characters '\n' must be explicitly added.
- The stdio.h library offers the necessary functions to write to a file:
  - **fputc(char, file_pointer)**: It writes a character to the file pointed to by file_pointer.
  - **fputs(str, file_pointer)**: It writes a string to the file pointed to by file_pointer.
  - **fprintf(file_pointer, str, variable_lists)**: It prints a string to the file pointed to by file_pointer. The string can optionally include format specifiers and a list of variables variable_lists
- Next Examples …….
- You may also need to **#include <stdlib.h for the codes to work**

Writing to a file - using fputc() Function:

- getc() and putc() are simplest functions used to read and write individual characters to a file.

```
#include <stdio.h>
int main() {
int i; FILE * fptr;
char fn[50];
char str[] = “Masomo yetu\n";
fptr = fopen(“ICS2305.txt", "w"); // "w" defines "writing mode"
for (i = 0; str[i] != '\n'; i++) {
/* write to file using fputc() function */
fputc(str[i], fptr);
}
fclose(fptr);
return 0;
}
```

Writing to a file - using fputc() Function -2

```
#include <stdio.h>
int main() {
FILE * fp;
 fp = fopen(“ICS2305.txt", "w+");
 fputs(“How was the CAT last week?,", fp);
fputs(“Easier than we expected\n", fp);
fputs(“We are on track!\n", fp);
 fclose(fp);
 return (0);
}
```

Writing to a file - using fprintf()Function

```
#include <stdio.h>
int main() {
FILE *fptr;
 fptr = fopen(“JKUAT.txt", "w"); // "w" defines "writing mode"
/* write to file */
fprintf(fptr, “Computer Science in JKUAT rocks \n");
fclose(fptr);
return 0;
}
```

Functions for reading from a text File

- **fgetc(file_pointer):** It returns the next character from the file pointed to by the file pointer. When the end of the file has been reached, the EOF is sent back.
- **fgets(buffer, n, file_pointer):** It reads n-1 characters from the file and stores the string in a buffer in which the NULL character '\0' is appended as the last character.
- **fscanf(file_pointer, conversion_specifiers, variable_adresses)**: It is used to parse and analyze data. It reads characters from the file and assigns the input to a list of variable pointers variable_adresses using conversion specifiers. Keep in mind that as with scanf, fscanf stops reading a string when space or newline is encountered.
-

using fgets(),fscanf() and fgetc () functions

In the next program we want to read the test from our **ICS2305.txt** file

- we use the fgets() function which reads line by line where the buffer size must be enough to handle the entire line.
- We reopen the file to reset the pointer file to point at the beginning of the file. Create various strings variables to handle each word separately. Print the variables to see their contents. The fscanf() is mainly used to extract and parse data from a file.
- Reopen the file to reset the pointer file to point at the beginning of the file. Read data and print it from the file character by character using getc() function until the EOF statement is encountered
- After performing a reading operation file using different variants, we again closed the file using the fclose function.

using fgets(),fscanf() and fgetc () functions

```
#include <stdio.h>
 int main() {
 FILE * file_pointer;
char buffer[30], c;
file_pointer = fopen(" ICS2305.txt ", "r");
printf("----read a line----\n");
fgets(buffer, 50, file_pointer);
 printf("%s\n", buffer);
printf("----read and parse data----\n");
file_pointer = fopen("fprintf_test.txt", "r"); //reset the
pointer
char str1[10], str2[2], str3[20], str4[2];
 fscanf(file_pointer, "%s %s %s %s", str1, str2, str3, str4);
```

Next page …..

using fgets(),fscanf() and fgetc () functions

```
printf("Read String1 |%s|\n", str1);
printf("Read String2 |%s|\n", str2);
printf("Read String3 |%s|\n", str3);
printf("Read String4 |%s|\n", str4);
printf("----read the entire file----\n");

file_pointer = fopen(" ICS2305.txt ", "r"); //reset the pointer
while ((c = getc(file_pointer)) != EOF) printf("%c", c);

fclose(file_pointer);
return 0;
}
```

Exercise

Write a program that takes integer input from a user and stores it in a notepad file (Kenya.txt) stored in any location in your drive

Hint: remember to open the file in write mode

Write another C program that reads the integer present in the Kenya.txt file and prints it onto the screen.

Reading and writing to a binary file

Functions fread() and fwrite() are used

To write into a binary file, you need to use the **fwrite()** function. The functions take four arguments:

- address of data to be written in the disk
- size of data to be written in the disk
- number of such type of data
- pointer to the file where you want to write.
- The general syntax is as follows

fwrite(addressData, sizeData, numbersData, pointerToFile);

writing to a binary file –f write ()

```
// C program for writing
// struct to file
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// a struct to read and write
struct person
{
        int id;
        char fname[20];
        char lname[20];
};

int main ()
{
        FILE *outfile;

        // open file for writing
        outfile = fopen ("person.dat", "w");
        if (outfile == NULL)
        {
                fprintf(stderr, "\nError opend file\n");
                exit (1);
        }
```

```
        struct person input1 = {1, "Sharon", "Ndinda"};
        struct person input2 = {2, "Opiyo", "Ndugu"};

        // write struct to file
        fwrite (&input1, sizeof(struct person), 1, outfile);
        fwrite (&input2, sizeof(struct person), 1, outfile);

        if(fwrite != 0)
                printf("contents to file written successfully
!\n");
        else
                printf("error writing file !\n");

        // close file
        fclose (outfile);

        return 0;
}
```

Reading a binary file –f read ()

```
// C program for reading
// struct from a file
#include <stdio.h>
#include <stdlib.h>

// struct person with 3 fields
struct person
{
   int id;
   char fname[20];
   char lname[20];
};

// Driver program
int main ()
{
   FILE *infile;
   struct person input;

   // Open person.dat for reading
   infile = fopen ("person.dat", "r")
```

```
      if (infile == NULL)
   {
      fprintf(stderr, "\nError opening file\n");
      exit (1);
   }

   // read file contents till end of file
   while(fread(&input, sizeof(struct person), 1,
infile))
      printf ("id = %d name = %s %s\n",
input.id,
      input.fname, input.lname);

   // close file
   fclose (infile);

   return 0;
}
```

OUTPUT

id = 1 name = Sharon Ndinda

id = 2 name = Opiyo Ndugu

Read about the following

- fseek(), ftell() and rewind() functions •
- fseek() - It is used to move the reading control to different positions using fseek function.
- ftell() - It tells the byte location of current position of cursor in file pointer.
- rewind() - It moves the control to beginning of the file.