Fundamentals of Programming CCS1063/CSE1062

Lecture 10 –File Handling in C

Professor Noel Fernando

UCSC

What is a File?

- A File is a collection of data stored in the secondary storage.
- The data was entered into the programs through the keyboard or other alternative method. .
- So Files are used for storing information that can be processed by the programs.
- A file is a collection of related data that a computers treats as a single unit.
- When a computer reads a file, it copies the file from the storage device to memory; when it writes to a file, it transfers data from memory to the storage device.
- In C language, we use a structure pointer of FILE type to declare a file.
- FILE is defined in stdio.h library.

Why file handling is important and a few features of using files:

- **Reusability:** File handling allows us to preserve the information/data generated after we run the program.
- **Saves Time:** Some programs might require a large amount of input from their users. In such cases, file handling allows you to easily access a part of a code using individual commands.
- **Commendable storage capacity:** When storing data in files, you can leave behind the worry of storing all the info in bulk in any program.
- **Portability:** The contents available in any file can be transferred to another one without any data loss in the computer system. This saves a lot of effort and minimises the risk of flawed coding.

Types of Files in a C Program

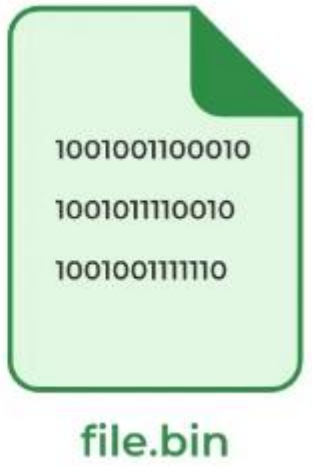
- There are two types of files in C
- Text files
- Binary files

- Text files
- Text files are the normal **.txt** files. You can easily create text files using any simple text editors such as Notepad.
- When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents.
- They take minimum effort to maintain, are easily readable, and provide the least security and takes bigger storage space.

Summary

Text File:

- Contains text in ASCII characters generally alphabets and numeric.
- Human readable
- Takes more space compared to binary files.



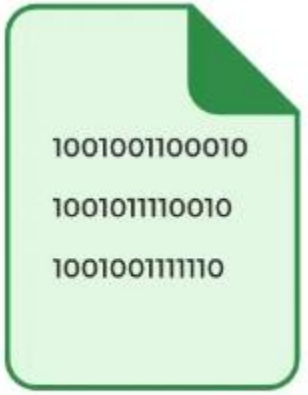
Types of Files in a C Program

- **Binary files**
- Binary files are mostly the **.bin** files in your computer.
- Instead of storing data in plain text, they store it in the binary form (0's and 1's).
- They can hold a higher amount of data, are not readable easily, and provides better security than text files.

Binary File:

- Contains text in binary form
- Not Human readable
- Faster access compared to text files.

file.txt



file.bin

File Operations

- File handling in C enables us to create, update, read, and delete the files stored on the local file system through our C program.
- The following operations can be performed on a file.
  - Creation of the new file
  - Opening an existing file
  - Reading from the file
  - Writing to the file
  - Deleting the file

C File Operations

- C file operations refer to the different possible operations that we can perform on a file in C such as:
  - Creating a new file – **fopen() with attributes as “a” or “a+” or “w” or “w+”**
  - Opening an existing file – **fopen()**
  - Reading from file – **fscanf() or fgets()**
  - Writing to a file – **fprintf() or fputs()**
  - Moving to a specific location in a file – **fseek(), rewind()**
  - Closing a file – **fclose()**

Working with files

- 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.

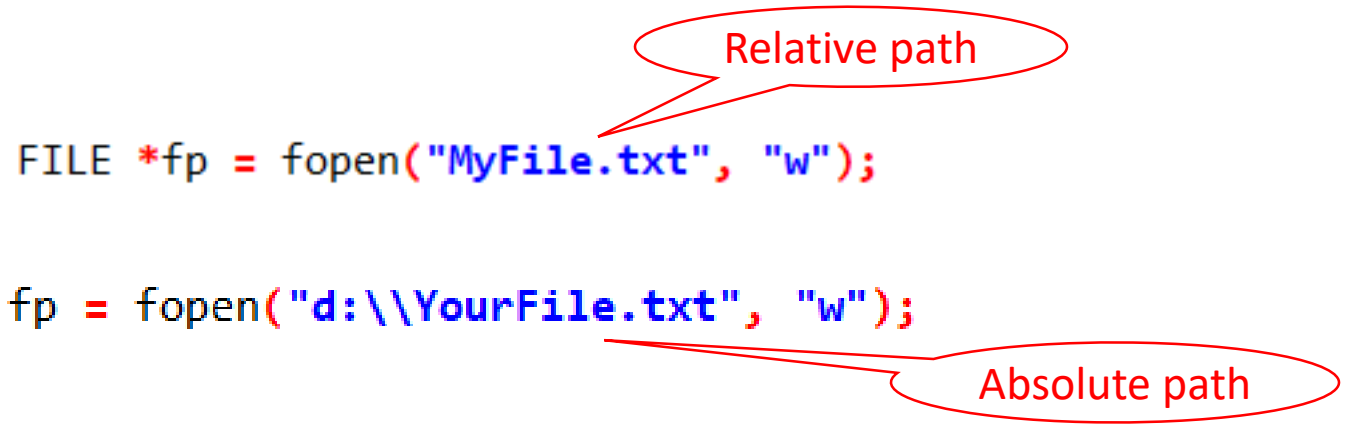
```
FILE *fptr
```

Opening a file - for creation and edit

Opening a file is performed using the fopen() function defined in the stdio.h header file.

Opening a File

- The file mode tells C how the program will use the file.
- The filename indicates the system name and location for the file.
- We assign the return value of fopen() to our pointer variable:



Closing a File

- When we finish with a mode, we need to close the file before ending the program or beginning another mode with that same file.

- The file (both text and binary) should be closed after reading/writing.
- Closing a file is performed using the fclose() function.

```
fclose(fptr);
```

Working with files

- The syntax for opening a file in standard I/O is:

```
ptr = fopen("fileopen","mode");
```

For example,

```
1) fopen("E:\\cprogram\\newprogram.txt","w");
2) fopen("E:\\cprogram\\oldprogram.bin","rb");
```

1) If newprogram.txt file is not exist in the location, The first function creates a new file named “newprogram.txt” and opens it for writing as per the mode **'w’**.

The writing mode allows you to create and edit (overwrite) the contents of the file.

Working with files

(2) fopen("E:\\cprogram\\oldprogram.bin","rb");

- The second binary file “oldprogram.bin"," exists in the location.
- The second function opens the existing file for reading in binary mode **'rb’**.
- The reading mode only allows you to read the file, you cannot write into the file.

Functions for file handling

- There are many functions in the C library to open, read, write, search and close the file.
- A list of file functions are given below:

Functions for file handling

No.	Function	Description
1	fopen()	opens new or existing file
2	fprintf()	write data into the file
3	fscanf()	reads data from the file
4	fputc()	writes a character into the file
5	fgetc()	reads a character from file
6	fclose()	closes the file
7	fseek()	sets the file pointer to given position
8	fputw()	writes an integer to file
9	fgetw()	reads an integer from file
10	ftell()	returns current position
11	rewind()	sets the file pointer to the beginning of the file

File opening modes in C

Mode	Description
r	opens a text file in read mode
w	opens a text file in write mode
a	opens a text file in append mode
r+	opens a text file in read and write mode
w+	opens a text file in read and write mode
a+	opens a text file in read and write mode
rb	opens a binary file in read mode
wb	opens a binary file in write mode
ab	opens a binary file in append mode
rb+	opens a binary file in read and write mode
wb+	opens a binary file in read and write mode
ab+	opens a binary file in read and write mode

Print to a File

- Syntax:
  - fprintf(<FilePointer>, <String>, <Variables>);

- It's just like printf() but output is sent to a file, rather than standard output.
- It's used when we wish to values in a file . The first argument is a pointer to file.

Example

```
void main(){

    FILE *fp = fopen("MyFile.txt", "w");

    fprintf(fp, "Hello, this is my first C file...\nThank you");

    fclose(fp);
}
```

fscanf()

- fscanf() Reads data from the file stream and stores them according to the parameter format into the locations pointed by the additional arguments.
- Syntax:
  - fscanf(<FilePointer>, <String>, <MemAddress>);

```
FILE *fp = fopen("MyFile.txt", "r");
double i;
fscanf(fp, "%lf", &i);
```

- scanf() reads from stdin;
- fscanf() reads from a user specified stream

getc & putc

- getc() or fgetc() reads a single character from a given file and increment the file pointer position.

```
FILE *fp = fopen("MyFile.txt", "r");
char ch;
ch = getc(fp);
```

- putc() or fputc() writes a single character to the output file, pointed to by the file pointer.

```
FILE *fp = fopen("MyFile.txt", "w");
char ch = 'a';
ch = putc(ch, fp);
```

END of File

- There are few ways to check the end of the file condition.
  - Using function feof()
  - using macro EOF.

```c
void main() {
    FILE *fp = fopen("StudentRecords.txt", "r");
    char ch = getc(fp);

    while(ch!=EOF) {
        putchar(ch); //printf("%c", ch);
        ch = fgetc(fp);
    }

    if(feof(fp))
        printf("\nEnd of file reached!");
}
```

fwrite()

- This function is used to write an entire block of data structure to a given binary file.

Declaration:

size_t fwrite(const *ptr, size_t size, size_t nmemb, FILE *stream)

1. ptr -- This is the pointer to the array of elements to be written.
2. size -- This is the size in bytes of each element to be written.
3. nmemb -- This is the number of elements to be written to the file.
4. stream -- This is the pointer to a FILE object that specifies an output stream.

```
char ch[] = "University of Colombo";
fwrite(ch, sizeof(char), sizeof(ch), fp);
```

Note: The size_t data type in C is an unsigned integer type used to represent the size of objects in bytes.
It is defined in the stddef.h header and is commonly used to represent the size of arrays, memory blocks, and strings.

fwrite()

- This function returns the total number of elements successfully written is returned as a size_t object, which is an integral data type. If this number differs from the nmemb parameter, it will show an error.

- This function store data in the file in binary form, so you may not be able to read the file.
- However, you can read the information stored in the file using fread()

Reading and writing to a text file

- For reading and writing to a text file, we use the functions fprintf() and fscanf().
- They are just the file versions of printf() and Scanf().
- The only difference is that fprintf() and fscanf() expects a pointer to the structure FILE.

Write a C program to takes a number from the user and stores it in the text file.

Write a C program to takes a number from the user and stores it in the text file.

```
#include <stdio.h>
#include <stdlib.h>
 int main()
 { int num;
FILE *fptr; // use appropriate location if you are using MacOS or Linux
fptr = fopen("C:\\program.txt","w");
```

Write a C program to takes a number from the user and stores it in the text file.

```
if(fptr == NULL)
{
printf("Error!");
exit(1);
}
printf("Enter num: ");
 scanf("%d",&num);
fprintf(fptr,"%d",num);
fclose(fptr);
return 0;
```

Note : After you compile and run this program, you can see a text file ’program.txt’ created in C drive of your computer.

When you open the file, you can see the integer you entered.

Write a C program to read the number from the text file.

Write a C program to read the number from the text file.

```
#include <stdio.h>
 #include <stdlib.h>
 int main()
{
int num;
FILE *fptr;
if ((fptr = fopen("C:\\program.txt","r")) == NULL){
printf("Error! opening file"); // Program exits if the file pointer returns NULL.
exit(1);
}
```

Write a C program to read the number from the text file.

```
fscanf(fptr,"%d", &num);
 printf("Value of n=%d", num);
fclose(fptr);
return 0; }
```

- Note:
- This program reads the integer present in the ‘program.txt’ file and prints it onto the screen.
- Other functions like fgetchar(), fputc() etc. can be used in a similar way.

fread()

- This function is used to read an entire block of data structure from a given binary file.

Declaration:

size_t fread(const *ptr, size_t size, size_t nmemb, FILE *stream)

1. ptr -- This is the pointer to a block of memory with a minimum size of size*nmemb bytes..
2. size -- This is the size in bytes of each element to be read.
3. nmemb -- This is the number of elements, each one with a size of size bytes.
4. stream -- This is the pointer to a FILE object that specifies an input stream.

```
char ch[20];
fread(ch, sizeof(char), sizeof(ch), fp);
```

Question : Write a C program to read Student Name, age and Gpa from a disk file and print on the screen.

Example: fread()

```c
struct student {
    char name[20];
    int age;
    double gpa;

};

void main() {

    struct student s;

    FILE *fp = fopen("Records.txt", "rb");
    if(fp==NULL) {
        printf("Error opening the file.");
        exit(1);
    }
    fread(&s, sizeof(struct student), 1, fp);
    printf("Student name: %s\n", s.name);
    printf("Student age: %d\n", s.age);
    printf("Student gpa: %f\n", s.gpa);
    fclose(fp);

}//main()
```

fseek()

- This function is used for setting the file position pointer at the specified byte.

Declaration:

int fseek(FILE *stream, int offset, int origin)

1. fp -- This is the pointer to a FILE object that identifies the stream.
2. offset -- This is the number of bytes to offset from origin.
3. origin -- This is the position from where offset is added. It is specified by one of the following constants:

SEEK_SET	Seeks from beginning of file
SEEK_CUR	Seeks from current position
SEEK_END	Seeks from end of file

fseek()

- This function returns zero if successful, else it returns nonzero value.

```
void main() {
    FILE *fp = fopen("seek.txt", "w+");
    fputs("This is file handling notes", fp);
    printf("%d", fseek( fp, 7, SEEK_SET));
    fputs(" C Programming Langauge", fp);
    fclose(fp);

}
```

Output : This is C programming Language

Write down the output of the following program

```
#include <stdio.h>
void main(){
   FILE *fp;

   fp = fopen("myfile.txt","w+");
   fputs("This is javatpoint", fp);

   fseek( fp, 12, SEEK_SET );
   fputs(“Lesson Number 1", fp);
   fclose(fp);
}
```

Write down the output of the following program

```
#include <stdio.h>
void main(){
   FILE *fp;

   fp = fopen("myfile.txt","w+");
   fputs("This is javatpoint", fp);

   fseek( fp, 12, SEEK_SET );
   fputs(“Lesson Number 1", fp);
   fclose(fp);
}
```

OUTPUT:

This is Java Lesson Number 1

Example:

- Read and print int from a txt file

```
void main() {

    FILE * fp;
    fp = fopen ("file.txt", "r");
    int numin;
    while ((fscanf(fp, "%d", &numin)) == 1)
     printf("%d ", numin);
    fclose(fp);

}//main()
```

Writing to a binary file

- 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.

```
fwrite(addressData, sizeData, numbersData, pointerToFile);
```

Writing to a binary file

```
#include <stdio.h>
 #include <stdlib.h>
 struct threeNum
{
int n1, n2, n3;
 };
int main() { int n;
struct threeNum num;
FILE *fptr;
```

Writing to a binary file

```
if((fptr = fopen("C:\\program.bin","wb")) == NULL) {
printf("Error! opening file");
// Program exits if the file pointer returns NULL.
exit(1);
}
for(n = 1; n < 5; ++n)
 {
num.n1 = n;
num.n2 = 5*n;
num.n3 = 5*n + 1;
fwrite(&num, sizeof(struct threeNum), 1, fptr); }
fclose(fptr);
return 0;
}
```

- Write the output of the program.

Writing to a binary file

```
if((fptr = fopen("C:\\program.bin","wb")) == NULL) {
printf("Error! opening file");
// Program exits if the file pointer returns NULL.
exit(1);
}
for(n = 1; n < 5; ++n)
 {
num.n1 = n;
num.n2 = 5*n;
num.n3 = 5*n + 1;
fwrite(&num, sizeof(struct threeNum), 1, fptr); }
fclose(fptr);
return 0;
}
```

OUTPUT:

```
n1: 1   n2: 5   n3: 6
n1: 2   n2: 10  n3: 11
n1: 3   n2: 15  n3: 16
n1: 4   n2: 20  n3: 21
```

- We declare a structure threeNum with three numbers - n1, n2 and n3,
- and define it in the main function as ***num***.
- The first parameter takes the address of ***num*** and the second parameter takes the size of the structure ***threeNum***
- we're only inserting one instance of *num* the third parameter is ***1*** And, the last parameter ****fptr*** points to the file we're storing the data.

Question: Write a C program to read names, ages and GPAs of 10 students from the keyboard and write on a disk file.

Question: Write a C program to read names, ages and GPAs of 10 students from the keyboard and write on a disk file.

```c
void main(){
   FILE *fptr;
   char name[15];
   int age, n, i;
   double gpa;

   fptr = fopen("StudentRecords.txt", "w");

   printf("Enter number of students you have: ");
   scanf("%d", &n);

   for(i=1; i<=n; i++) {
       printf("Enter student %d name:", i);
       scanf("%s", name);
       printf("Enter student %d age:", i);
       scanf("%d", &age);
       printf("Enter student %d GPA:", i);
       scanf("%lf", &gpa);
       fprintf(fptr, "Student %d name: %s\n", i, name);
       fprintf(fptr, "Student %d age: %d\n", i, age);
       fprintf(fptr, "Student %d GPA: %.1f\n", i, gpa);
       printf("Student %d records added to the file.\n", i);
   }

   fclose(fptr);

}//main()
```

Example:

- Copy a file

```
void main() {

    FILE *fps, *fpt; //file pointers for source and target files
    fps = fopen ("Content.txt", "r"); //open the source file
    fpt = fopen ("Content-Copy.txt", "w"); // creating the target file
    char ch;
    printf("coping the file...\n");
    while ((fscanf(fps, "%c", &ch)) == 1)
     fprintf(fpt, "%c", ch);
    printf("file copied successfully.");
    fclose(fps);
    fclose(fpt);

}//main()
```

Write down the output of the program :

```
int main()
{
    FILE* fptr;
    fptr = fopen("file.txt", "w+");
    fprintf(fptr, “Sri Lanka Vs India \n");

    // using rewind()
    rewind(fptr);

    // reading from file
    char buf[50];
    fscanf(fptr, "%[^\n]s", buf);

    printf("%s", buf);

    return 0;
}
```

Write down the output of the program :

```
int main()
{
    FILE* fptr;
    fptr = fopen("file.txt", "w+");
    fprintf(fptr, “SriLanka Vs India \n");

    // using rewind()
    rewind(fptr);

    // reading from file
    char buf[50];
    fscanf(fptr, "%[^\n]s", buf);

    printf("%s", buf);

    return 0;
}
```

Sri Lanka Vs India

Question : Write down the output of the following program

```
#include <stdio.h>
main()
{
FILE *fp; fp = fopen("/tmp/test.txt", "w+");
fprintf(fp, "This is testing for fprintf...\n");
 fputs("This is testing for fputs...\n", fp);
fclose(fp);
}
```

Write down the output of the following program

```
#include <stdio.h>
main()
{
FILE *fp; fp = fopen("/tmp/test.txt", "w+");
fprintf(fp, "This is testing for fprintf...\n");
 fputs("This is testing for fputs...\n", fp);
fclose(fp);
}
```

- Output (in a file : ("/tmp/test.txt“)
- This is testing for fprintf...
- This is testing for fputs...

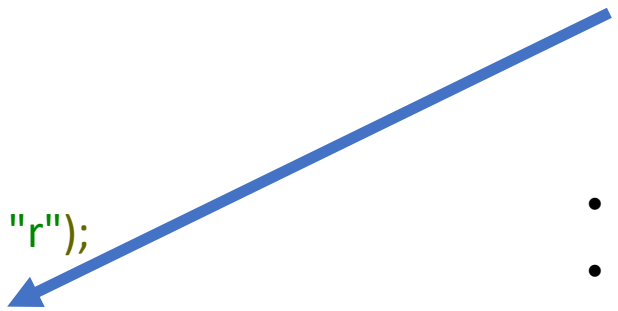
Write down the output of the following program if one read the contents of the previous file (("/tmp/test.txt“)

```
#include <stdio.h>
 main() {
FILE *fp;
char buff[255];
 fp = fopen("/tmp/test.txt", "r");
fscanf(fp, "%s", buff);
printf("1 : %s\n", buff );
fgets(buff, 255, (FILE*)fp);
printf("2: %s\n", buff );
fgets(buff, 255, (FILE*)fp);
printf("3: %s\n", buff );
fclose(fp);
}
```

-

Output of the above program

```
#include <stdio.h>
 main() {
FILE *fp;
char buff[255];
 fp = fopen("/tmp/test.txt", "r");
fscanf(fp, "%s", buff);
printf("1 : %s\n", buff );
fgets(buff, 255, (FILE*)fp);
printf("2: %s\n", buff );
fgets(buff, 255, (FILE*)fp);
printf("3: %s\n", buff );
fclose(fp);
}
```



- The string is only scanned till a whitespace is encountered

- Output :
- 1 : This
- 2: is testing for fprintf...
- 3: This is testing for fputs...

The newline character (\n) is called an escape sequence, and it forces the cursor to change its position to the beginning of the next line on the screen.