New Chapter

Date

Files and File Handling

Till now the operations using the C program are done on a prompt/terminal which is not stored anywhere. But in the software industry, most programs are written to store the information fetched from the program.
One such way is to store the fetched information in a file.

In C program, the following operations can be performed on a file:

* Creation of a new file
* Opening an existing file
* Reading from the file
* Writing to the file
* Deleting the file
* Moving to the specific location in a file
* Closing a file

Types of Files in C:

It is important to recognise two types of files when dealing with files:

* Text files
* Binary files

* **Text files:** Generally a text file contains alphabets, digits, and special characters or symbols in the form of ASCII characters and are generally used to store a stream of characters. Each line in a text file ends with a new line character ('\n').

Date

* **Binary files:** They contain data in the binary form (0's and 1's). Binary files are mostly the .bin files. They can hold a higher amount of data, and are not readable easily, and provides better security than text files.

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

Functions	Description
fopen()	opens new or existing file
fprint()	write data into the file
fscanf()	reads data from the file
fputc()	writes a character into the file
fgetc()	reads a character from file
fclose()	closes the file
fseek()	sets the file pointer to given position
fputw()	writes an integer to file
fgetw()	reads an integer from file.
ftell()	returns current position
rewind()	sets the file pointer to the beginning of the file.

Date

1.) Opening File: fopen()

We must open a file before it can be read, write, or update. The fopen() function is used to open a file. This function is defined in the stdio.h header file.

Syntax for opening a file:

```
FILE *fopen("filename", "mode");
```

This whole thing is a file pointer.

In another way,
We can first declare a file pointer variable, & then write complete form in two steps as:

```
FILE *ptr;
ptr = fopen("filename", "mode");
```

The fopen() function accepts two parameters:

* The file name (string): If the file is stored at some specific location, then we must mention the path at which the file is stored. For example, a file name in above syntax can be like:
  "C://some_folder/some_file.txt"

* mode: The mode in which the file is to be opened. This parameter is a string.

Date

Various mode that can be used in fopen() function :

Mode	Description	If file doesn't exist
r	Opens a text file in read mode. Searches file. If file is opened successfully, fopen() loads it into memory and sets up a pointer that points to the first character in it.	fopen() returns NULL
rb	Open for reading in binary mode.	fopen() returns NULL
w	Open for writing. Searches file. If the file exists, its contents are overwritten.	If file doesn't exist, a new file will be created.
wb	Opens a binary file in write mode. If file exists, contents are over-written.	A new file will be created.
a	Open for append. Data is added at the end of the file. Searches file. If file is opened successfully, fopen() loads it into memory and sets up a pointer that points to the last character in it (for appending)	A new file will be created.
ab	Opens a binary file in append mode. Data is added to end of the file.	A new file will be created.

Date

Mode	Description	If file doesn't exist
r+	Opens a file in read & write mode - pointer points to the first character in it.	fopen() returns NULL.
rb+	Opens binary file in read & write mode.	returns NULL.
w+	Open file for both reading & writing.	New file created.
wb+	Open a binary file for both reading & writing.	New file created.
a+	Open for both reading & appending. - after file loaded in memory, pointer points to last character in it.	New file created.
ab+	Open binary file for both reading & appending	New file created.

2.) Closing File: fclose()

This function is used to close a file. The file must be closed after performing all the operations on it.

Syntax:

```
fclose(fptr);
```

Here, 'fptr' is a file pointer associated with the file to be closed.

Date

3.) Reading & Writing to a text file: (fprintf() & fscanf())

For reading data from a file and writing data to a file, fprintf() and fscanf() functions are used.

fprintf() — for writing to a file
fscanf() — for reading from a file.

⇒ fprintf() is used to write set of characters into file.

Syntax:

```
fprintf(file_pointer, "text to write \n");
```

↑ must be placed to indicated 'end'.

⇒ fscanf() is used to read set of characters from file. It reads a word from the file and returns EOF at the end of file.

Syntax:

```
fscanf(file_pointer, "%s", char_array to store read data);
```

↳ This whole thing will be equal to 'EOF' when reading of file reaches end of file. So, if we want to read whole thing (until end), we can use while loop with condition that "until the above thing != EOF —

```
while(fscanf(fp, "%s", char_arry) != EOF){
    .....
```

↓ depends what we want to read. It may be "%d", &a

Date

If we are reading an integer,

```
fscanf(file_pointer, "%d", &a);
```

where, 'a' is an integer variable to store the integer that is read from file.

→ It depends what datatype it is that we are reading from file.

* A simple program to write a text (string) to a file named "test1.txt" using fprintf() function.

```
#include <stdio.h>

int main(){
    FILE *fp;          // file pointer declared
    fp = fopen("test1.txt", "w");  //opening the file for writing.

    if (fp == NULL){
        printf("test1.txt file failed to open.");
    }

    else{  // if program control comes here, means file is successfully opened.
        fprintf(fp, "I am studying file handling\n");
        fclose(fp);  // closing the file.
    }

    return 0;
}
```

Date

* A program to write a text in a file (as previous) using fprintf() but ask the uses to input the text to be written.

```
#include <stdio.h>

int main(){
    FILE * fptr;      // declaring a file pointer.
    char text[100]; // to store the text entered by user.

    fptr = fopen("user_text.txt", "w"); // opening a
                    file named user_text.txt for writing
                    into this file.

    if(fptr == NULL){
        priotf("Failed to open the file \n");
    }

    else{
        printf("Enter a text you want to save \n");
        scanf("%s", text); //stored in 'text' array

        //now, storing this text (writing) in file
        fprintf(fptr, "%s\n", text);

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

Date

* A program to write variety of data (not only string) like information of a student - Name, age, marks, into a file using fprintf()

```
#include <stdio.h>
int main(){
    FILE *fp;
    char name[30];
    int age;
    float marks;

    //opening a file "student.txt" for writing data in it.

    fp = fopen("student.txt", "w");

    if(fp == NULL){
        printf("Failed to open the file \n");
        exit(1);
    }
    //as we used exit(1) in 'if' condition, i.e. when file
    opening fails, now we don't need to use 'else'.

    printf("Enter the name:\n");   [ // name entered & stored in char array (not in file yet) ]
    scanf("%s", name);
    fprintf(fp, "Name = %s\n", name); //name stored in file
    printf("Enter the age: \n");
    scanf("%d", &age);
    fprintf(fp," Age= %d\n", age); //age stored in file
    printf("Enter marks :\n");
    scanf("%f", &marks);
    fprintf(fp, "Marks = %.2f\n", marks); //marks stored in file
    fprintf(fp, "Name=
    fclose(fp);
    return(0);
```

Date

* A program to read data (set of characters) from a file using fscanf() and display the content.

→ Here, we need to open an existing file in read ("r") mode. The read characters must be stored somewhere, for which we will declare a char array.
Moreover, to read whole content of the file, we must reach up to the end of the file. Until we reach there, the reading operation is to be continued — for this purpose, we will use "while" loop with condition that 'until reading reaches EOF'.

```
#include<stdio.h>
int main(){
    FILE *fp;
    char buffer[300];
    fp = fopen("file.txt", "r");

    while(fscanf(fp, "%s", buffer) != EOF){
        printf("%s", buffer); //displaying the read text.
    }
    fclose(fp);

}
```

"We will be back to fscanf() after learning more functions".

Date

4) fputs() and fgets():

These are used to write and read string from stream.

* Writing file: fputs() function

This function writes a line of characters into file. It outputs 'string' to a stream.

Syntax: fputs("text_to_write", file_pointer);

Example:

```
#include<stdio.h>
#include<conio.h>

void main(){
    FILE *fp;
    fp = fopen("myfile1.txt", "w"); //writing mode

    fputs("Hello File Handling Program", fp);

    fclose(fp);

    getch();
}
```

* Reading file: fgets() function

This function reads a line of characters from file. It gets string from a stream.

Syntax: fgets(char array to store read string, int n, file_pointer);

Here, n is the maximum length of the string that should be read.

Date

Example:

```
#include<stdio.h>
#include<conio.h>

void main(){
    FILE *fp;
    char text[300]; // to store the read string

    fp = fopen("myfile1.txt", "r"); // read mode

    // now read the string (1 line) from file
    fgets(text, 50, fp); // 50 characters will be read
                         //and stored in array 'text'.

    // to display the read text, we have two options:
    // 1st options is:
    printf("%s", fgets(text, 50, fp));

    // 2nd option: read text is stored in char array,
      so we display this array as:
    printf("The string is: %s", text);

    fclose(fp);

    getch();

}
```

Date

Before moving onwards, lets visualize the file content, placement, how pointer works:
For example, lets say a file contains these:

```
My name is John. \n
I live in Australia. \n
I live with my family. \n
EOF
```

when opened for reading, pointer is set up as pointing to 1st character as shown.

* while reading character-wise (function पढ़ने वाली कि से), if \n is encountered, one line is completed. Then the pointer comes to 1st character of another line.

* When EOF is encountered by pointer, reading file is completed.

* After reading is complete, the pointer has reached at EOF, hence, if we have anything to perform in that file from the start, the pointer must be brought back to start of file → done by rewind(file_pointer).

* In case of writing to a file, lets say we write one line of text (where we need to put \n at the end of the line to indicate the line is end). After placing \n at end of 1st line by programmer, the pointer then points to next line, so, anything written further is placed in 2nd line. '\n' in 2nd line makes the pointer to point to the 3rd line & so on. When writing is completed, remember that the pointer has reached to the last line. So, after writing completion, if we want to read the file and display content to the console, the pointer is to be placed at the start of the file before we start reading. Again, rewind(file_pointer) is required.

Date

Now, going back to fscanf() function to see different aspect of reading a file than we did earlier.

Previously, we saw:

```
fscanf(fp, "%s", char_array);
```

"%s" indicates the data type of the data to be read from file.

char_array indicates the place where the read data to be stored.

* A program to demonstrate the use of fscanf() bit differently than we did earlier to enhance understanding on working of this function.

- First of all, we will store some data to a new file — opening file in "w+" mode, i.e. 'write' as well as 'read'. Storing means 'writing' operation at first.
- Secondly, we will read those contents of newly created file using fscanf()
- Note: as we need to write & read both, the file opening mode must support both, i.e. "w+".
  Also, after completing the 'write' operation, we have to use rewind() function to set the pointer at the start of the file so that reading operation can be held next.

Date

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    FILE *fp;
    char str1[20], str2[20], str3[20]; // to store the read strings from file.
    int year; // to store the read integer from file.

    fp = fopen("myfile2.txt", "w+");

    // now we are writing some data to this file.
    // this can also be done using fprintf() but
    // here, it is done using fputs().

    fputs("We are in 2023", fp);
    fputs("\n", fp); // end of line indication

    // writing is completed. Pointer reached at last. Before
    // reading, lets bring pointer back to start.

    rewind(fp);

    // Now, we will read the content, i.e "We are in 2023\n".
    // Each word will be stored in separate char arrays,
    // str1, str2, str3 and year, where 1st 3 words are
    // strings and 4th is integer that will be stored in 'year'.

    fscanf(fp, "%s %s %s %d", str1, str2, str3, &year);
    // reading is done - stored in variables. To confirm, lets
    // display those variables - str1, str2, str3 and year.
    printf("Read String1 |%s|\n", str1);
    (do this for all.....)
    fclose(fp);
    return 0;     } // closing of main().
```

Date

5.) fputc() and fgetc()

* Writing file : fputc() function

It is used to write a single character into file. It outputs a character to a stream.

Syntax :

```
fputc('character to write', file_pointer);
```

Example:

```
#include <stdio.h>
int main(){
    FILE * fptr;
    fptr = fopen("myfile3.txt", "w");

    fputc('a', fptr); // writing single character into file

    fclose(fptr);

    return 0;
}
```

* Reading file : fgetc() function

It returns a single character from the file. It gets a character from the stream. It returns EOF at the end of file.

→ Everytime this function is executed (called), the next character is returned and at last it returns EOF.

Date

→ Lets say there are 15 characters. To read all of them, we needs to write fgetc() function 15 times to get all the characters. For this, we can apply 'loop' ~~beat~~ which must iterate 15 times to call the function 15 times. But the question is, how would we know how many characters are there in the file — i.e., how to know how many times the loop should iterate. In such situation, we use 'while' loop. The condition within 'while' loop will be — received_character_from_file != EOF
i.e. when the pointer reaches end of file, the program control comes out of the loop.

Syntax:

```
fgetc(file_pointer);
```

Example:

```
#include <stdio.h>
#include<conio.h>
void main(){
   FILE* fp;
   char c;    // to store the received character from file
   fp = fopen("myfile4.txt", "r");

   //now reading characters one-by-one via while loop

   while((c=fgetc(fp)) != EOF){ // fgetc(fp) receives a char
      printf("%c", c);          from file & stored in 'c'
   }                            which, if not the end, then
   fclose(fp);                  continue looping
   getch();
}
```

Date

* Write a program to read characters from keyboard (i.e. from user), store it in a file called text.data (i.e. writing into file) and display it (i.e. read from file & then display in console)

Note: in program, we will ask user to enter some characters. We will store each entered character in a char variable but it seems like next entered character will override (replace) the first character if we keep on storing it in the same variable. Also, we are not storing entered charactered in any char array because our main target is to store them in a file. So, it is simple to understand this way:

- User enters a first character
- we save it in a char variable using either scanf() or getchar()
- We, ~~them~~ then, immediately write (store) that character in the file before the user enters another character, i.e. before the char variable is replaced by a new character.

If we think in a normal way, we will do program like this. [char c;]

One character stored in the file.

```
printf("Enter a character: \n");
c = getchar(); //entered character stored in char c
fputc(c, file_pointer);
```

another character stored in file

```
printf("Enter a character: \n");
c = getchar();
fputc(c, file_pointer);
```

If we are about to implement the program in this manner, the lines of code will be very greater for many characters to be stored in file.
Moreover, it will not be in the hand of user that how many characters to be entered, instead it is in the hand of programmer. If programmer writes those codes (for 1 char as shown above) 10 times, user is asked only 10 times to enter characters.

Writing the same code many times will be solved by using 'loop'. But, as in 'for-loop', if the programmer iterates the loop for a particular number of times, then,

- problem of writing longer code is solved, but
- problem regarding the no. of characters ~~to be decided by the~~ being decided by the programmer instead of user is not yet solved.

In such scenario, when the no. of iteration in loop is not to be predefined, we use 'while' loop. Having 'while' loop, the important thing is to know the condition within 'while', i.e. the condition to decide the no. of iterations. Here, we are receiving entered characters one-by-one & saving it in a file. Those characters will be placed in single line, & we know that, the line in a file must terminate ~~at~~ with "\n". It means, unless the user enters '\n', the loop should continue and the user will keep entering characters. So, the condition within 'while' loop will be like:

```
while (user_entered_char != '\n') {
        store char in file;
}
```

By this, it means, when user wants stop entering further characters, he/she will have to enter \n.

Date

```c
#include <stdio.h>
#include<stdlib.h>
int main(){
    FILE* fptr;
    char c; // to store used entered character
    fptr = fopen("text.dat", "wb+"); // binary file

    if(fptr == NULL){
        printf("file could not be opened");
        return 0;
        exit(0); // exit from program
    }

    printf("Enter some characters(\n for end) : ");

    while((c=getchar()) != '\n'){
        fputc(c, fptr);
    }

    // now we will read those characters from file & display it using putc() function, but, before that, file pointer is to be set at start of file.
    rewind(fptr);

    printf("\n The characters are \n");

    while((c=fgetc(fptr)) != EOF){
        putc(c,); // displaying character
                  ↳ stdout
    }

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

Date

6.) fwrite() and fread()

These functions are used to write and read binary files.
Usually, for writing in file, it is easy to write string or int to file using fprintf() and fputc(), but a difficulty is faced when writing contents of struct. fwrite() and fread() make task easier when we want to write and read blocks of data.

(a) fwrite() function is used to write block of data (structure or arrays) into a file.

Syntax:

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

ptr - pointer to array of elements or pointer to structure variable whose data to be written in file.
(array_name or &structure_variable).

size - This is the size in bytes of each element to be written.
(sizeof(struct structure_name))

nmemb - This is the no. of elements / no. of records to be written

stream - This is the file pointer of the file where data is to be written

Date

* Program for writing struct to file

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

//defining a structure
struct person{
    int id;
    char first_name[20];
    char last_name[20];
};

int main(){
   FILE *fp;
   fp = fopen("person.dat", "wb");  //writing to binary file

   if(fp == NULL){
       printf("File couldn't be opened \n");
       return 0;
       exit(1);
   }
   // creating two structure variables
   struct person p1 = {1, "Rohan", "sharma"};
   struct person p2 = {2, "Bikash", "Bhandari"};

   // writing struct to file now
    fwrite(&p1, sizeof(struct person), 1, fp);
    fwrite(&p2, sizeof(struct person), 1, fp);

    if(fwrite != 0){
        printf("contents are successfully written to file");
    }
```

Date

```
    else{
        printf("error writing file");
    }
    fclose(fp);
    return 0;

} // closing of main()
```

(b) fread(): This function is used to read a block of data (array or structure) from a file.

Syntax:

fread(void *ptr, size_t size, size_t, nmemb, FILE *stream);

ptr - pointer to a block of memory where the read content of file will be loaded (stored)
(&structure_variable)
↳ it means, the contents from file will be read and read data will be stored in mentioned structure, so, we provided the address of structure variable.

size - size in bytes of each element to be read
(sizeof(struct structure_name))

nmemb - number of elements/records

stream - file pointer

* Program for reading struct from a file.

Note: when reading from file, if we know the no. of records (i.e. no. of structure variables), then we can write fread() function as many times as the no. of records, or we can use 'for loop' and write fread() within it. If we don't know the no. of records, use 'while loop' and place "fread()" within its condition. It will run until all records are fetched from file.

Date

For now, we will use the file "person.dat" used in previous example which has two records as p1 and p2.

```
#include <stdio.h>
#include <stdlib.h>

// creating structure named 'person' with 3 members as id, first_name and last_name. This is defined so as to store the data read from the file "person.dat".
struct person{
    int id;
    char first_name[20];
    char last_name[20];
};

int main(){
    FILE* fp;
    struct person p; // although there are two records in file, we declared only one variable here. Because, when we read the data from file, we read only one record at a time - it will be stored in 'p' and we will print it immediately. When printed, then only we read another record from file - stored again in 'p', i.e. previous content of 'p' is replaced but it doesn't matter because previous content has already been displayed at this moment.
```

Date

```
// opening the file for reading purpose

fp = fopen("person.dat", "rb");
if (fp == NULL){
    printf("File couldn't be opened \n");
    return 0;
    exit(1);
}

// now read the content of the file
fread(&p, sizeof(struct person), 1, fp);
printf("id = %d  name = %s  %s\n", p.id, p.first_name, p.last_name);

// fetching/reading another record
fread(&p, sizeof(struct person), 1, fp);
printf("id = %d  name = %s  %s\n", p.id, p.first_name, p.last_name);

fclose(fp);

return 0;

}  // closing of main()
```

★ Now Lets write a program to write a struct with multiple records (i.e. creating array of structure) into a file. After writing, we then read those contents. (It means, the file to be opened in write & read modes both, & as it is binary file, the mode will be wb+). But, among the read data, we will display certain records only which satisfies the mentioned (in question) condition. Also note that, after completion of writing, we should rewind() before starting reading.

Date

* Create a file called university.dat. Input 'n' records of college in a structure called university having name of college, address and no. of faculty, and display (read then printf) name of those colleges whose address is Kathmandu.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>

struct university {
    char name[30];
    char address[30];
    int no_of_faculty;
};

void main(){
    struct university u[100];
    int i,n;  // 'i' used in 'for' loop. 'n' for no. of colleges
    FILE *fp;
    fp=fopen("university.dat","wb+");
    if(fp==NULL){
        printf("File couldn't be opened\n");
        getch();
        exit(0);
    }

    printf("Enter the no. of colleges: \n");
    scanf("%d",&n);

    printf("Enter college information: \n");
```

Date

```
for(i=0; i<n; i++){ //iterating for 'n' colleges
    printf("Enter college name:\n");
    gets(u[i].name); //stored in structure right now
    printf("Enter address:\n");
    gets(u[i].address);
    printf("Enter no. of faculty:\n");
    scanf("%d", &u[i].no_of_faculty);

    // till now, information of single college is stored in
       structure.
    // now, we store them in file
    fwrite(&u, sizeof(u), 1, fp);
}

// storing into file completed. Now, to display particular
   college name, we have to read them from file &
   before reading, rewind() to be done.
   ~~new~~
rewind(fp);

printf("Name of colleges having address Kathmandu:\n");

for(i=0; i<n; i++){ //to read all records from file
    fread(&u, sizeof(u), 1, fp); //reading all records
    // but displaying only those with address 'Kathmandu'.

    if(strcmp(u[i].address, "Kathmandu")==0){
        printf("\n College Name= %s", u[i].name);
    }
}

fclose(fp);
getch();
} // closing of main()
```

Date

Significance/Importance of File Handling

The output of a C program is generally deleted when the program is closed. Sometimes, we need to store that output for purposes like data analysis, result presentation, comparision of output for different conditions etc. The use of file handling is exactly what the situation calls for.

Some reasons of use of file handling:

* **Reusability**: When saved in file, can be reused.

* **Portability**: Without losing any data, files can be transferred to another location in the computer system. The risk of flawed coding is minimized with this approach.

* **Efficient**: It allows to easily access a part of a code.

* **storage Capacity**: Files allow to store data.