UNIT-5

FILES AND PREPROCESSORS

IN C

* Files:-

→ A file is on external collection of related dat treated as a unit

→ A file is a place on a disk where a group of relat data is stored and retrieved when ever necessary with destroying data

→ The primary purpose of a file is keep record of data.

→ The two common forms of secondary storage a

- disks (hard disk, CD and DVD)
- tapes.

→ Each file ends with an end of file (EOF) at a specified byte numbers, recorded in file structure

→ A file must first be opened properly before it can be accessed for reading or writing

→ when a file is opened an object (buffer) is created and a stream is associated with the object.

* There are 2 types of files

- Text files
- Binary files

* Text files:

A text files stores textual information like alphabets, numbers and special symbols etc

→ Actually the ASCII code of textual characters is stored in text files

→ Ex of some text files include: c, java, c++ source code files and files with .txt extensions

* BINARY FILES:

→ Text mode is inefficient for storing large amount of numerical data because it occupies large space

→ Only solution to this is to open a file in binary mode, which takes lesser space than the text mode

→ These files contain the set of bytes which stores the information in binary form.

→ One main drawback of binary files is data is stored in human unreadable form

→ Ex of binary files are .exe files, video stream files, image files. etc

* modes of opening files:

~~The~~ To store the data in a file we have to specify three things

① File name

② Datastructure

③ Purpose

* File name :-

It is a string of characters that make up a valid file name which may contain two parts, a Primary name and an optional period with the extension

Prog1.c

My first.java

Data.txt store

* Data structure:

→ It is defined as FILE in the library of standard I/O function definitions.

→ Therefore all files are declared as type FILE

→ FILE is define data type

FILE * fp;

* Purpose:
It defines the reason for which a file is opened and the mode does this job

fp = open ("filename", "mode");

* The different modes of opening files are:

→ "r" (read) mode:
Syntax: fp = fopen ("filename", "r");

→ "w" (write) mode
Syntax: fp = fopen ("filename", "w");

→ "a" (append mode
Syntax: fp = fopen ("filename", "a");

→ "r+" (read and write) mode
Syntax: fp = fopen ("file name", "r+");

→ "w+" (read and write) mode
Syntax: fp = fopen ("filename", "w+");

→ "a+" (append and read) mode
Syntax: fp = fopen ("filename", "a+");

* BASIC OPERATIONS ON a FILE:

FUNCTION NAME	OPERATION
fopen()	creates a new file for use or opens an existing file for use

fclose()	closes a file which has been opened for use
fclose all()	closes all files which are opened
getc()/fget(c)	Reads a character from a file
puts()/fputc()	Writes a character to a file
fprintf()	Writes a set of data values to files
fscanf()	Reads a set of data values from files.
getw()	Reads an integer from file
Putw()	writes an integer to a file.
gets()	Reads a string from a file
puts()	writes a string to a file
fseek()	Sets the position to a desired point in a file
ftell()	Gives the current position in the tale
rewind()	Sets the position to the beginning of the file.

write a c program to read and display the contents of a file.

```
# include<stdio.h>
int main()
{
   FILE *fp;
   char ch;
   Printf("\n Enter the text and press @ to stop");
   fp=fopen("data.txt", "w");
   while((ch=getchar())!='a')
   fputc(ch, fp);
   fclose(fp);
   Printf("\n The entered data is:");
   fp=fopen("data.txt", "r");
   while((ch=fgetc(fp))!=EOF)
```

```
    Putchar(ch);
    fclose(fp);
    return0;
```

* Write a C program to append the data in a file

```
#include<stdio.h>
int main()
{
  ~~file~~ FILE *fp;
      char ch;
  Printf("\n Enter the text and press @ to stop");
  fp=fopen("data.txt", "a");
  while((ch=getchar())!='@')
  fputc(ch, fp);
  fclose(fp);
   Printf("\n The entered data is:");
   fp=fopen("data.txt","r");
   while((ch=fgetc(fp))!=EOF)
   Putchar(ch);
   fclose(fp);
   return 0;
}
```

Output:

data.txt

```
welcomehello.
```

* Write a C program to copy the contents of one file into another

```
#include<stdio.h>
int main()
{
```

```
FILE *fp1, *fp2;
char ch;
Printf("\n Enter the text and press @ to stop");
fp1 = fopen("data.txt", "w");
while ((ch=getchar())!='@')
fputc(ch, fp1);
fclose(fp1);
Printf("\n The entered data is :");
fp1 = fopen("data.txt", "r");
fp2 = fopen("abc.txt", "w");
while ((ch= fgetc(fp1))!=EOF)
fputc(ch, fp2);
fclose(fp1);
fclose(fp2);
fp2= fopen("abc. txt", "r");
while((ch=fgetc(fp2))!=EOF)
Putchar(ch);
fclose(fp2);
return 0;
}
```

* write a C program to merge two files and copy in third one.

```
#include<stdio.h>
int main()
{
    FILE *fp1, *fp2, *fp3;
    char ch;
    Printf("\n Enter the text in file 1 and press @
                          to stop");

    fp1=fopen("data.txt", "w");
    while((ch=getchar())!='@')
    fputc(ch, fp1);
    fclose(fp1);
Printf("\n Enter the text in file 2 and press @ to stop");
fp2=fopen("abc.txt","w");
 while((ch=getchar())!='@')
  fputc(ch,fp2);
  fclose(fp2);
  Printf("\n The entered data is: ");
  fp1=fopen("data.txt","r");
  fp3=fopen("xyz.txt","w");
 while((ch=fgetc(fp1))!=EOF)
  fputc(ch, fp3);
  fclose (fp1);
  fp2=fopen("abc.txt","r");
  while((ch=fgetc(fp2))!=EOF)
   fputc(ch,fp3);
   fclose(fp2);
   fclose(fp3);
   fp3=fopen("xyz.txt","r");
   while((ch=fgetc(fp3))!=EOF)
    Putchar(ch);
    fclose(fp3);
    return 0;
}
```

* FILE I/O FUNCTIONS

=> ~~[illegible]~~ getc() and fgetc()

-> putc() / fputc()

-> fprintf() and fscanf()

-> getw() and putw()

-> fputs() and fgets()

* Reading the data from the files :-

1. fgetc(): It is used to read characters from file that has been opened for read operation.

Syntax: c = fgetc (file pointer);

This statement reads a character from file pointed to by file pointer and assign to c. It returns an end of file marker EOF when end of the file has been reached.

2. fscanf():- The function is similar to that of scanf function except that it works on files

Syntax: fscanf(fp, "control string", list);

Ex: fscanf(f1, "%s%d", str, &num);

The above statement reads string type data and integer type data from file

3. ~~fget(w)~~ fgetw():- This function reads an integer from file.

Syntax: fgetw(file pointer);

4. fgets(): This function reads a string from a file pointed by a file pointer. It also copies the string to a memory location referred by an array

Syntax: fgets(string, no.of bytes, file pointer);

5. fread():- This function is used for reading an entire structure block from a given file

Syntax:
fread (&struct_name, sizeof (struct_name) 1, file pointer);

* Writing data to a file:

1. fputc():- This function writes a character to a file that has been opened in write mode

Syntax: fputc(c, fp);

This statement writes the character contained in character variable c into a file whose pointer is

2. fprintf():- This function performs function similar to that of printf

Syntax: fprintf("fp, "%s %d", str, num);

3. fputw():- It writes an integer of to a file

Syntax: fputw(variable fp);

4. Fputs(): This function writes a string into a file pointed by a file pointer.

Syntax: fputs(string, file pointer);

5. fwrite(): This function is used for writing an entire block structure to a given file

Syntax: fwrite (&struct_name, sizeof(struct_name), 1, file pointer);

* Random access to files:-

At times we needed to access only a particular Part of the file rather than accessing all the data sequentially which can be achieved with the help of ~~8~~ functions 3 main functions.

They are

1) ftell()
2) Rewind()
3) fseek()

* ftell() :-

. This function returns the value of the current Pointer position in the file.

. The value is count from the beginning of the file

Syntax:- ftell(fptr);

fptr → file pointer.

* rewind()

. This function is used to move the file pointer to the beginning of the given file

Syntax:-rewind (fptr);

* fseek() :-

. This function is used for seeking the pointer Position in the file at the specified byte

Syntax:- fseek (file pointer, displacement, pointer position);

* File pointer:- It is the pointer which points to the file

* Displacement (or) offset:- It is +ve or -ve. This is the no of bytes which are skipped backward (if negative) or forward (if positive) from the current position

This is attached with L because this is long integer

* Pointer position:- This sets the pointer position in the file.

POSITION VALUE	VALUE CONSTANT	MEANING
0	SEEK_SET	Beginning of fi
1	SEEK_CUR	current positio
2	SEEK_END	END OF FILE

Ex: fseek (P, 10L, 0)

0 means pointer position is on beginning of the file from this statement pointer position. i.e skipped 10 bytes from the beginning of the file

2) fseek (P, 5L, 1)

1 means current position of the pointer position. from this statement pointer position is skipped 5 bytes forward from the current position

3) fseek (P, -5L, 2)

from this statement pointer position is skipped 5 bytes from backward.

Operation	Description
fseek(fptr, 0, 0)	This will take us to the beginning of the file
fseek(fptr, 0, 2)	This will take us to the end of the file.
fseek(fptr, N, 0)	This will takes us to (N+1) th bytes in the file.
fseek (fptr, N, 1)	This will take us N bytes forward from the current position in the file
fseek(fptr, -N, 1)	This will takes us N bytes backward from the current Position in the file
fseek (fptr, -N, 2)	This will takes us N bytes backward from the end position in the file.

Ex:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
  FILE *fp;
  char ch;
  clrscr();
  fp=fopen("sample.txt", "r");
  fseek(fp, 5, SEEK_SET); (or) fseek(fp, 5, 0);
  ch=fgetc(fp);
  while(fp=null)
{
    Printf("%c\t\n", ch);
    Printf("%d", ftell(fp));
    ch=fgetc(fp);
}
rewind (fp);
Printd("the last position is:%d", ftell(fp));
fclose(fp);
getch();
}
```

* Error handling in files:-

→ It is possible that an error may occur during I/O operations on a file. Typical error situation include

→ Trying to read beyond the end of file mark

→ Device overflow.

→ Trying to use a file that has not been opened

→ Trying to perform an operation on a file, when the file is opened for another type of operation.

→ opening a file with an invalid filename

→ Attempting to write a protected file.

* feof()

→ The feof() function can be used to test for an end of file condition

→ It takes, a file pointer as its only argument and returns a non zero integer value, if all the data from the specified file has been read and return zero otherwise

→ If fp is a pointer to file that has just opened for reading then the statement

```
if (feof(fp))
    Printf("End of data")
```

would display the message "End of data" on reaching the end of file location

* ferror()
→ The ferror() function reports the status of the file indicated
→ It also takes a file pointer as its argument and returns a non-zero integer if an error has been detected up to that point, during processing otherwise.
→ It returns zero
→ The statement

```
if(ferror(fp)!=0)
    Printf("an error has occured\n");
```

would print an error message if the reading is not successful.

* fp== NULL
→ we know that whenever a file is opened using fopen function, a file pointer is returned
→ If the file cannot be opened for some reason,
then the function returns a value pointer
→ This facility can be used to test whether a fil is has been opened or not

```
if (fp==NULL)
```

Printf("file could not be opened.\n");

* Perror()

→ It is a standard library function which Prints the error messages specified by the compiler

for ex: if (ferror(fp))
    Perror(filename);

* Introduction to stdin, stdout and stderr.

→ There are 3 special FILES that are always defined for program

→ They are

1) stdin (standard input)
2) stdout (standard output)
3) Stderr (Standard error)

Standard file	file Pointer	Device
Standard input	stdin	Keyboard
Standard output	stdout	Screen
" error	stderr	Your screen

* Standard Input:

→ standard input is where things come from when you use scanf().

→ In other words,

Scanf("%d", &val);

is equivalent to the following

fscanf(): fscanf(stdin, "%d", &val)

* standard output:-

→ Similarly, standard output is exactly where things go when you use printf()

→ In other words,

printf("value = %d\n", val);

is equivalent to the following

fprintf() : fprintf(stdout, "value = %d\n", val);

→ The std ~~out~~ in is normally associated with the keyboard and stdout with the screen, unless redirection is used

* Standard error:

→ standard error is where you should display error messages

→ fprintf (stderr, "Can't open input file in. list (\n');

→ stderr is normally associated with the same place as stdout

→ however, redirecting stdout does not redirect stderr

→ For ex: % a.out > outfile

→ only redirects stuff going to stdout to the file outfile.. anything written to stderr goes to the screen.

* Enumeration Data type:-

* Another user-defined datatype is - enumerat
datatype (enum.)

Syntax: enum identifier {value 1, value 2, ... value

⇒ where identifier is user-defined datatype which is used to declare variables that can have one of the values enclosed within the braces. value 1, value 2, ... value n all these are known as enumerat constants.

Ex: enum identifier v_1, v_2 ... v_n

v_1 = value 1 ; v2 = value 2;

Ex: enum day {Monday, Tuesday ..... sunday};
enum day week - f, week - end
week - f = Monday.

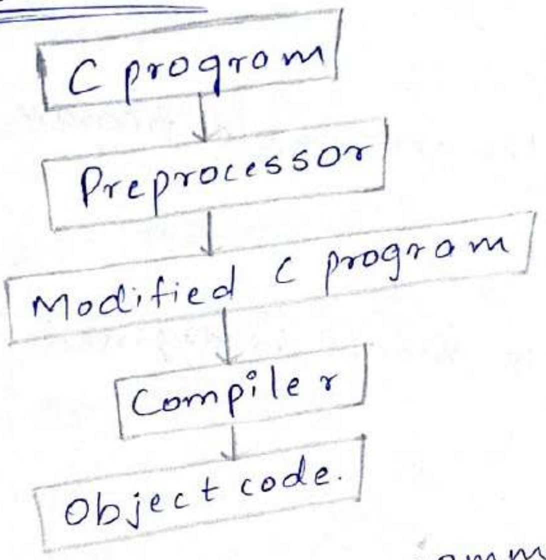
* Write a C program to implement enumeration datatype.

```
#include<stdio.h>
enum week{Mon, Tue, Wed, Thur=6, fri, sat, sun};
int main()
{
  enum week day;
  day=Wed;
Printf("\n Mon=%d", Mon);
Printf("\n Wed=%d", day);
Printf("\n fri=%d", fri);
Printf("\n Sun=%d", Sun);
return 0;
}
```

Output:-

```
Mon= 0
Wed= 2
fri= 7
Sun= 9
```

* PREPROCESSOR:-



⇒ All the preprocessor commonds begin with a pound symbol #

→ There are 3 Kinds of preprocessor directives.

1) File inclusion (# include)

include: Inserts a particular header from another file.

Eg. #include <stdio.h>

2) conditional compilation. (#if, #ifdef, #ifndef, # elif, #else, #endif)

*# if:-

→ The #if directive tests an expression.

→ The #if directive is true, if the expression evaluates to true (nonzero).

Ex

Eg:
```
#include <stdio.h>
   #define WINDOWS 1
   int main()
   {
     #if WINDOWS
     Printf("Windows");
     #endif
     return 0;
   }
```
Here the output of the executable program: Windows

* #ifdef:-

Returns true if this macro is defined

* #ifndef:

Returns true if this macro is not defined

Eg:
```
#ifndef MESSAGE
  #define MESSAGE "You wish!"
  # endif
```

* #else: The alternative for #if

* #elif:- #else an #if in one statement

* #endif: Ends preprocessor conditional

* Macros (#define, #undef, #error, #pragma)

define: substitutes a preprocessor macro

eg: #define MAX-ARRAY-LENGTH 20.

This directive tells the CPP to replace instances of MAX-ARRAY-LENGTH 20

Use #define for constants to increase readability

* #undef:- Undefines a preprocessor macro.

Eg: #unded FILE-SIZE

#define FILE-SIZE 42

This tells the CPP to undefine existing FILE-SIZE and define it as 42.

error: Prints error message on stderr

pragma: Issues special commands to the compiler using a standardized method.

* Pre-define macros are.

* _DATE_ The current date as a character literal in "MMM DD YYYY" format

* _TIME_ The current time as a character literal in "HH:MM:SS" format

* _FILE_ This contains the current filename as a string literal.

* _LINE_ This contains the current line number as a decimal constant

* _STDC_ Defined as 1 when the compiler complies with ANSI standard.

* W.A.C.P t/o implement predefined macros.

```
#include <stdio.h>
void main()
{
 Printf("File : %s\n", _FILE_);
 Printf(" Date : %s\n", _DATE_);
 Printf(" Time : %s\n", _TIME_);
 Printf(" Line : %d\n", _LINE_);
 Printf(" ANSI : %d\n", _STDC_);
```

Output:-

```
File : main. c
Date : Dec 28 2023.
Time : 08:16:31
Line : 15
ANS1 : 1
```

* Command line arguments:-

=) Command line argument is the parameter supplied to a program when the program is invoked.

-) This parameter may represent a filename the Program should process

* for ex; if we want to execute a program to copy the contents of a file named X_FILE to another one name Y_FILE then we may use a command line like

```
C:> program X-FILE-Y-FILE.
```

=) Program is the filename where the executable code of the program is stored

=) This eliminates the need for the program to request the user to enter the file names during execution.

=) The main function can take two arguments. called argc, argv and information contained in the command line is passed on to the program to these arguments, when main is called up by the system.

=) The variable argv is an argument vector and represents an array of characters pointers that Point to the command line arguments.

→ The arg c is an argument counter that counts the number of arguments on the command line. The size of this array is equal to the value of argc. In order to access the command line arguments, we must declare the main function and its Parameters as follows:

```
main (arg c, arg v)
int argc;
char *argv[ ];
{
          }
}.
```

argv[0] represents the program name.

* W.A.C.P to add two numbers using command line arguments.

```
#include<stdio.h>
int main (int arg c, char *arg v[])
{
 int x, sum=0;
 Printf("\n Number of arguments are: %d", argc);
 Printf("\n The arguments are: ");
 for(x=0; x<argc; x++)
 {
  Printf("\n argv[%d]= %s", x, argv[x]);
  if(x<2)
  continue;
  sum=sum+atoi(argv[x]);
 }
Printf("\n Program name: %s", argv[0]);
Printf("\n name is: %s", argv[1]);
Printf("\n Sum is: %d", sum);
return(0);
}
```