File System

1.File Pointers

It is not enough to just display the data on the screen. We need to save it because memory is volatile and its contents would be lost once program terminated, so if we need some data again there are two ways one is retype via keyboard to assign it to particular variable, and other is regenerate it via programmatically both options are tedious. At such time it becomes necessary to store the data in a manner that can be later retrieved and displayed either in part or in whole. This medium is usually a "file" on the disk.

Introduction to file

Until now we have been using the functions such as scanf,printf, getch, putch etc to read and write data on the variable and arrays for storing data inside the programs. But this approach poses the following programs.

1. The data is lost when program terminated or variable goes out of scope.
2. Difficulty to use large volume of data.

Introduction to file

Until now we have been using the functions such as scanf,printf, getch, putch etc to read and write data on the variable and arrays for storing data inside the programs. But this approach poses the following programs.

1. The data is lost when program terminated or variable goes out of scope.
2. Difficulty to use large volume of data.

We can overcome these problems by storing data on secondary devices such as Hard Disk. The data is storage on the devices using the concept of "file".A file is collection of related records, a record is composed of several fields and field is a group of character.

The most straight forward use of files is via a file pointer.

FILE *fp;
fp is a pointer to a file.

The type FILE, is not a basic type, instead it is defined in the header file stdio.h, this file must be included in your program.

2.File Operation

1. Create a new file.
2. Open an existing file
3. Read from file
4. Write to a file
5. Moving a specific location in a file(Seeking)
6. Closing File

3.Opening a File

fp = fopen(filename, mode);
The filename and mode are both strings.

Here modes can be
"**r**" read
"**w**" write, overwrite file if it exists
"**a**" write, but append instead of overwrite
"**r+**" read & write, do not destroy file if it exists
"**w+**" read & write, but overwrite file if it exists
"**a+**" read & write, but append instead of overwrite
"**b**" may be appended to any of the above to force the file to be opened in binary mode rather than text mode.

Eg.

```
FILE *fp;
fp=fopen("input.txt","r");
//Opens inputs.txt file in read mode
fclose(fp); //close file
```

Sequential file access is performed with the following library functions.

1. **fopen()** - Create a new file
2. **fclose()** - Close file
3. **getc()** - Read character from file
4. **putc()** - Write character to a file
5. **getw()** - Read Integer from file
6. **putw()**-Write Integer to a file
7. **fprintf()** - Write set of data values
8. **fscanf()** - Read set of data values

C Preprocessor Directives

Before a C program is compiled in a compiler, source code is processed by a program called preprocessor. This process is called preprocessing.

Commands used in preprocessor are called preprocessor directives and they begin with "#" symbol. Below is the list of preprocessor directives that C language offers.

1 Macro
Syntax:
#define

This macro defines constant value and can be any of the basic data types.

2 Header file inclusion
Syntax:
#include

The source code of the file "file_name" is included in the main program at the specified place.

3 Conditional compilation
syntax:
#ifdef, #endif, #if, #else, #ifndef

Set of commands are included or excluded in source program before compilation with respect to the condition.

4 Other directives
syntax
#undef, #pragma

#undef is used to undefine a defined macro variable. #Pragma is used to call a function before and after main function in a C program.