Introduction

- Data files to store the data
  - The data can be some output generated by a program, or they can be input for a program.
  - Useful in engineering problem solutions, as they often involve large amounts of data.
  - It is not a good idea to print large amounts of data to the screen or to read large amounts of data from the keyboard.

Opening Data Files

- File pointer
  - `FILE *fp;`
- Open
  - `fp=fopen("c:\\myFile.txt", "r");`
- Example

```
#define FILENAME  "myFile.txt"
FILE *fp;
fp = fopen (FILENAME , "r");
```

File open mode

Mode	Description
r	Opens an existing text file for reading.
w	Opens a text file for writing*. C program will write content from the beginning of the file.
a	Opens a text file - writing in appending mode*. Here your program will start appending content in the existing file content.
r+	Opens a text file for reading and writing.
w+	Opens a text file for reading and writing*. It first truncate the file to zero length if it exists.
a+	Opens a text file for reading and writing*. The reading will start from the beginning but writing can only be appended.

* if file does not exist, a new file is created.

Check for File Open

```
fp = Fopen (FILENAME, "r") ;
if  (fp == NULL)
   printf ("Error opening  input file \n ") ;
else {
   /* your program here*/
}
```

- fopen may fail even if the file that does not exist (or maybe write-protected). In those cases, fopen will return 0, the NULL pointer.
- One way to be sure that your program can find a data file is to store the data file in the same folder as your program file.

Closing Data Files

- File Close

```
int fclose( FILE *fp );
```

- Example

```
fclose(fp);
```

- The **fclose( )** function returns zero on success, or **EOF** if there is an error in closing the file.
- This function actually, flushes any data still pending in the buffer to the file, closes the file, and releases any memory used for the file. The EOF is a constant defined in theheader file **stdio.h**.

Reading with fscanf()

- Reads formatted input from a file
- Example

```
fscanf (myFile, "%f %f " , &myVar1, &myVar2) ;
```

- Stops reading after the first space character encounters.

Reading with fgetc()

- Reads a character from the input file referenced by the file pointer. The return value is the character read, or in case of any error it returns **EOF**.

- int fgetc(FILE *fp);

Reading with fgets()

- Reads a line from the input file referenced by the file pointer and stores it into the string. The return value is the string read, or in case of any error it returns **EOF**.

- char * fgets(char *str, int n, FILE *fp);

- If the End-of-File is encountered and no characters have been read, the contents of str remain unchanged and a null pointer is returned.
- If an error occurs, a null pointer is returned.

Example

```
#include <stdio.h>
int main ()
{
  FILE *fp;
  int c,  n = 0;
  fp = fopen("file.txt","r");
  if(fp == NULL) {
    printf("Error in opening file");
    return(-1);
  }
  do {
    c = fgetc(fp);
    if( feof(fp) ) {
       break ;
    }
    printf("%c", c);
  }while(1);
  fclose(fp);
  return(0);
}
```

Writing with fprintf()

- Writes a formatted stream to the file.
- Example

```
FILE *fp;
fp=fopen("c:\\myFile.txt", "w");
fprintf(fp, “Hello my file.\n");
```

Writing with fprintf()

- Write a formatted output to the file
- Example

```
int fprintf(FILE *fp, char *format,…);
```

- *fp – the file pointer
- char *– the content that to be written to the file
- Example:

fprintf(fp, “%f”,average);

average – is a float variable.

Writing with fputc()

- Write a character at a time
- Example

```
int fputc(int char, FILE *fp);
```

- char – the character to be written
- *fp – the file pointer
- Example:
- fputc(97,fp);

Writing with fputs()

- Write a string to the specified file
- Example

```
fputs("Hello world for fputs...\n", fp);
```

- The first argument, should be in the range of an unsigned char and should be a valid character.
- The second argument, fp is the file to write to.
- On success, fputs will return the value,and otherwise, it will return **EOF**.

Example A

```
#include <stdio.h>  main()
{
   FILE *fp;
   fp = fopen(“myFile.txt", "w+");
   fprintf(fp, “%s”, “Hello world for fprintf...\n");
   fputs(" Hello world for fputs...\n", fp);
   fclose(fp);
}
```

Example B

```
#include <stdio.h>
main(){
   FILE*fp;
   char buff[255];

   fp = fopen(“myFile.txt", "r");
   fscanf(fp, "%s", buff);
   printf("1 : %s\n", buff);

   fgets(buff, 255, fp);
   printf("2: %s\n", buff);

   fgets(buff, 255, fp);
   printf("3: %s\n", buff);
   fclose(fp);
}
```

Output

- It reads the file created in Example A (myFile.txt), and produces the following result:

  1 : Hello

  2: world forfprintf...

  3: Hello world forfputs...

- 1. **fscanf()** read just **Hello** because it encountered a space,
- 2. **fgets()** read the remaining line till it encountered end of line.
- 3. **fgets()** read second line completely.

fgets()

- reads up to n - 1 characters from the input stream referenced by fp. It copies the read string into the buffer **buff**, appending a **null** character to terminate the string.
- If this function encounters a newline character '\n' or the end of the file EOF before they have read the maximum number of characters, then it returns only the characters read up to that point including new line character.