Chapter - 11 : File Processing

Writing and Reading File Data

- A program that uses a disk file can write data to a file, read data from a file, or a combination of the two.
- One can write data to a disk file in three ways:
- First way:
  - One can use formatted output to save formatted data to a file.
  - One should use formatted output only with text-mode files.
  - The primary use of formatted output is to create files containing text and numeric data to be read by other programs such as spreadsheets or databases.
- Second way:
  - One can use character output to save single characters or lines of characters to a file.
  - It′s possible to use character output with binary-mode files, it can be tricky.
  - One should restrict character-mode output to text files.
  - The main use of character output is to save text (but not numeric) data in a form that can be read by C, as well as other programs such as word processors.
- Third way:
  - One can use direct output to save the contents of a section of memory directly to a disk file.
  - This method is for binary files only.
- When one wants to read data from a file, he has the same three options: formatted input, character input, or direct input.
- The data will be read in the same mode that it was saved in.

Character Input

- There are three character input functions: getc( ) and fgetc( ) for single characters, and fgets( ) for lines.
- The functions getc() and fgetc() are identical and can be used interchangeably.
- They input a single character from the specified stream.
- Here is the prototype of getc(), which is in STDIO.H:

  int getc (FILE *fp);
- The argument fp is the pointer returned by fopen() when the file is opened.
- The function returns the character that was input or EOF on error.
- getc( ) was used in earlier programs to input a character from the keyboard.
- This is another example of the flexibility of C′s streams--the same function can be used for keyboard or file input.

Example

```
#include<stdio.h>

int main( )
{FILE *fp;
  int c, n=0;

  fp=fopen("file.txt","r");
  if(fp==NULL)
  {
   printf("Error in File");
   return(-1);  // indicate errro
  }

  do {
     c=fgetc(fp);

     if(feof(fp) ) break;

     printf("%c",c);
     } while(1);

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

The fgets() Function

- To read a line of characters from a file, use the fgets( ) library function.
- The prototype is: **char *fgets(char *str, int n, FILE *fp);**
- The argument str is a pointer to a buffer in which the input is to be stored, n is the maximum number of characters to be input, and fp is the pointer to type FILE that was returned by fopen() when the file was opened.
- Characters are read until a newline is encountered or until n-1 characters have been read, whichever occurs first.
- If successful, fgets() returns str.

Example

```
#include<stdio.h>

int main( )
{
  FILE *fp;
  char str[60];

  fp=fopen("file.txt","r");
```

```
 if(fp==NULL)
   {
    printf("Error in File");
    return(-1);  // indicate errro
   }
  if (fgets(str,60,fp) != NULL)
      puts(str);
  fclose(fp);
  return(0);
}
```

The putc() Function

- The library function putc() writes a single character to a specified stream.
- Its prototype in STDIO.H is as shown below: **int putc(int ch, FILE *fp);**
- The argument ch is the character to output.
- The argument fp is the pointer associated with the file (the pointer returned by fopen() when the file was opened).
- The function putc() returns the character just written if successful or EOF if an error occurs.
- The symbolic constant EOF is defined in STDIO.H, and it has the value -1.
- Because no "real" character has that numeric value, EOF can be used as an error indicator (with text-mode files only).

Example

```
#include<stdio.h>

int main( )
{
  FILE *fp;
  int ch;

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

 for(ch=33; ch<=100; ch++)
    fputc(ch,fp);

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

The fputs() Function

- To write a line of characters to a stream, use the library function fputs().
- This function works just like puts().
- The only difference is that with fputs() one can specify the output stream.
- Also, fputs() doesn′t add a newline to the end of the string; one must explicitly include it.
- Its prototype in STDIO.H is:
- char fputs(char *str, FILE *fp);
- The argument str is a pointer to the null-terminated string to be written, and fp is the pointer to type FILE returned by fopen() when the file was opened.
- The string pointed to by str is written to the file, minus its terminating \0.
- The function fputs() returns a non-negative value if successful or EOF on error.

Example

```
#include<stdio.h>

int main( )
{
  FILE *fp;
  int ch;

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

  fputs("This is a C Programming", fp);

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