CHAPTER - 11
FILE OPERATIONS

CHAPTER 11

FILE OPERATIONS

INTRODUCTION **(Reading)**

STANDARD FILE POINTERS

OPENING AND CLOSING

CHARACTER INPUT AND OUTPUT WITH FILES **(Reading)**

READING FROM FILES

WRITING TO FILES

OTHER FILE OPERATIONS

FILE OPERATIONS

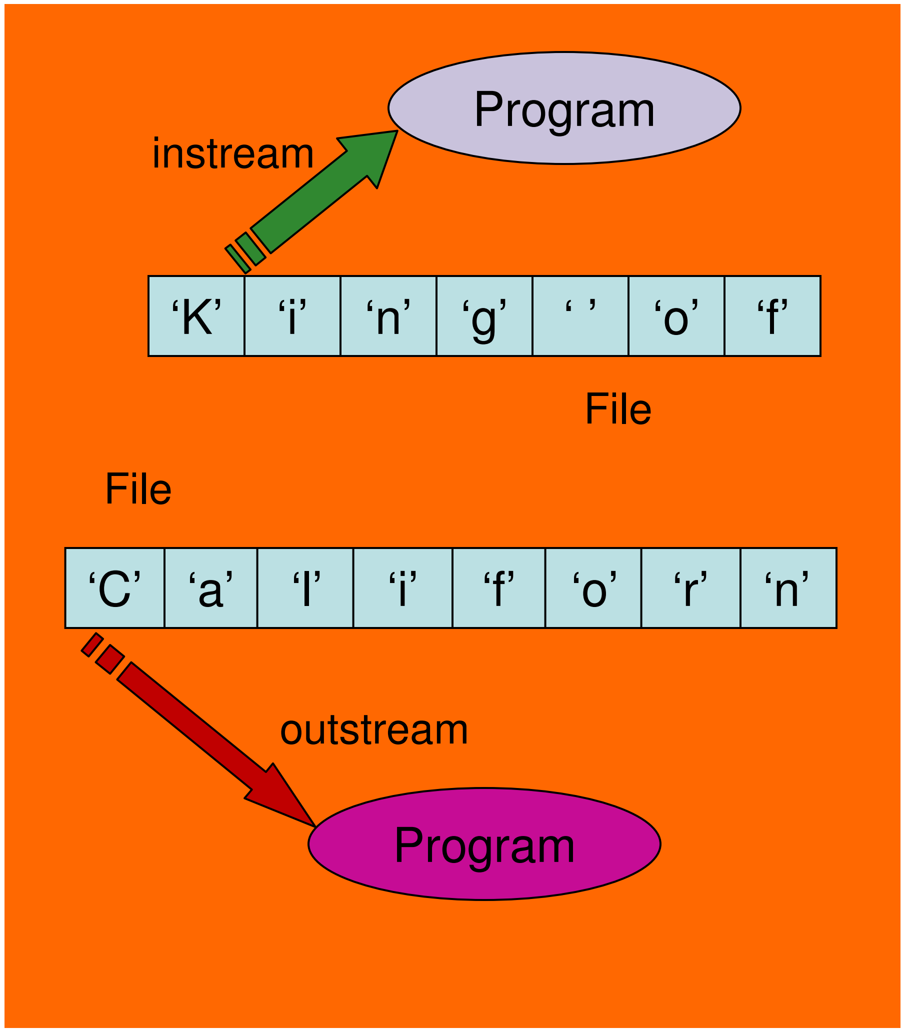
STANDARD FILE POINTERS

- The standard files, standard input (keyboard), the standard output (screen), and the standard error (screen) and their associated streams are automatically opened when program execution begins.
- Streams provide communication channels between files and programs.
- Opening a file returns a pointer to a FILE structure that contains information used to process the file.
- This structure include a file descriptor which is an index into an operating system array called open file table.
- Each array element contains a file control block that the operating system uses to administer a particular file.
- The standard input, standard output, and standard error are manipulated using file pointers stdin, stdout, and stderr.

FILE OPERATIONS

STREAMS IN FILE OPERATION

- A stream is a sequence of bytes. They contain the data that is written to a file, and that gives more information about a file than attributes and properties.
- Each stream that is associated with a file has its own allocation size, actual size, and valid data length.
- Each stream also maintains its own state for compression, encryption, and sparseness.
- Stream is a conduit from or to the file for the program code.



FILE OPERATIONS

OPENING AND CLOSING

OPENING A FILE POINTER

- *fopen* function opens a file, which returns the required file pointer.
- If the file cannot be opened for any reason then the value NULL will be returned.
- fopen takes two arguments, both are strings, the first is the name of the file to be opened, and the second is an access character.

```
FILE * out_file
if ((out_file = fopen ("output_file",
                              "w")) == NULL)
    printf (stderr, "Cannot open %s\n",
                              "output_file");
```

CLOSING A FILE POINTER

- The *fclose* command can be used to disconnect a file pointer from a file.
- This is usually done so that the pointer can be used to access a different file.
- Systems have a limit on the number of files which can be open simultaneously, so it is a good idea to close a file when you have finished using it.
- If files are still open when a program exits, the system will close them for you.

```
fclose (out_file);
```

FILE OPERATIONS

FILE OPEN MODES

- File I/O operations take place in one of two translation modes, text or binary. Data files are usually processed in text mode.

The allowed modes for *fopen* are as follows:

r - open for reading

w - open for writing (file need not exist)

a - open for appending (file need not exist)

r+ - open for reading and writing, start at beginning

w+ - open for reading and writing (overwrite file)

a+ - open for reading and writing (append if file exists)

- With the mode specifiers above the file is open as a text file. In order to open a file as a binary file, a "b" character has to be included in the mode string. This additional "b" character can either be appended at the end of the string (thus making the following compound modes:

  "rb", "wb", "ab", "r+b", "w+b", "a+b") or be inserted between the letter and the "+" sign for the mixed modes ("rb+", "wb+", "ab+").

- Additional characters may follow the sequence, although they should have no effect, "t" is sometimes appended to make explicit the file is a text file.

FILE OPERATIONS

READING FROM FILES

- Text files are used for storing character strings in a file.
- When finished using the file must always be closed.
- fscanf - Read formatted data from stream (function)
- fwrite -Write block of data to stream
- fputs - Write string to stream
- fprintf - Write formatted output to stream (function)
- fread - Read block of data from stream
- fgets - Get string from stream
- fgetc - Get character from stream
- fputc - Write character to stream

```
#include<stdio.h>
int main ()
{
      char mystring [100];
      FILE *pFile;
      int n;
      pFile = fopen ("myfile.txt" , "r");
      if (pFile == NULL)
              perror ("Error opening file");
      else
      {
              if ( fgets (mystring , 100 , pFile)
                                            != NULL )
                   puts (mystring);
              fclose (pFile);
      }
      return 0;
}
```

FILE OPERATIONS

READING FROM FILES

```
#include<stdio.h>
int main ()
{
     FILE *pFile;
     int c;
     int n = 0;
     char name [100];

     pFile = fopen ("test.txt", “r");
      if (pFile==NULL)
             perror ("Error opening file");
```

```
     else
     {
           do {
                     c = fgetc (pFile);
                     if (c == '$')
                        n++;
                 }
           while (c != EOF);
           fclose (pFile);
           printf ("The file contains
                            %d dollar sign
              characters ($).\n", n);
      }
     return 0;
}
```

FILE OPERATIONS

WRITING TO FILES

- size_t corresponds to the integral data type returned by the language operator *sizeof* and is defined in the <cstring> header file (among others) as an unsigned integral type.

```
#include<stdio.h>
int main ()
{
    FILE * pFile;
    char buffer[] = { 'x' , 'y' , 'z' };
    pFile = fopen ("myfile.bin", "wb");
    fwrite (buffer , 1 , sizeof (buffer) ,
                                    pFile );
    fclose (pFile);
    return 0;
}
```

```
#include<stdio.h>
int main ()
{
    FILE *pFile;
    int n;
    char name [100];
    pFile = fopen ("test.txt", "w");
    for (n = 0 ; n < 3 ; n++)
    {
            puts ("please, enter a
                                name: ");
            gets (name);
            fprintf (pFile, "Name %d
                    [%-10.10s]\n",n,name);
    }
    fclose (pFile);
    return 0;
}
```

FILE OPERATIONS

WRITING TO FILES

```
#include <stdio.h>
int main ()
{
     FILE * pFile;
     char sentence [256];

     printf ("Enter sentence to append: ");
     fgets (sentence, 255, stdin);
     pFile = fopen ("mylog.txt", "a");
     fputs (sentence, pFile);
     fclose (pFile);
     return 0;
}
```

```
/* fwrite example : write buffer */
#include<stdio.h>
int main ()
{
     FILE * pFile;
     char buffer[] = { 'x' , 'y' , 'z' };
     pFile = fopen ("myfile.bin", "wb");
     fwrite (buffer , 1 , sizeof (buffer) ,
                                  pFile );
     fclose (pFile);
     return 0;
}
```

FILE OPERATIONS

OTHER FILE OPERATIONS

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
     long lSize;
     char * bufCP;
     size_t result;
     FILE * pFile;

    pFile = fopen ("myfile.bin" , "rb" )
    if (pFile==NULL)
    {
           fputs ("File error", stderr);
           exit (1);
    }
```

```
// obtain file size:
fseek (pFile, 0 , SEEK_END);
lSize = ftell (pFile);
rewind (pFile);

// allocate memory for whole file:
bufCP = (char *) malloc (sizeof (char) *
                                  lSize);
if (bufCP == NULL)
{
       fputs ("Memory error", stderr);
       exit (2);
}

// copy the file into the buffer:
result = fread (bufCP, 1, lSize, pFile);
```

FILE OPERATIONS

OTHER FILE OPERATIONS

- continued -

```
    if (result != lSize)
    {
            fputs ("Reading error",
    stderr);
            exit (3);
    }
    /* the whole file is now loaded in
    the memory buffer. Terminate */
     fclose (f);
     free (bufCP);
    return 0;
}
```

```
#include <stdio.h>
int main ()
{
    char str [80];
    float f;
    FILE * pFile;
    pFile = fopen ("myfile.txt", "w+");
   fprintf (pFile, "%f %s", 3.1416, "PI");
    rewind (pFile);
    fscanf (pFile, "%f", &f);
    fscanf (pFile, "%s", str);
    fclose (pFile);
    printf ("I have read: %f and %s
                              \n", f, str);
    return 0;
}
```