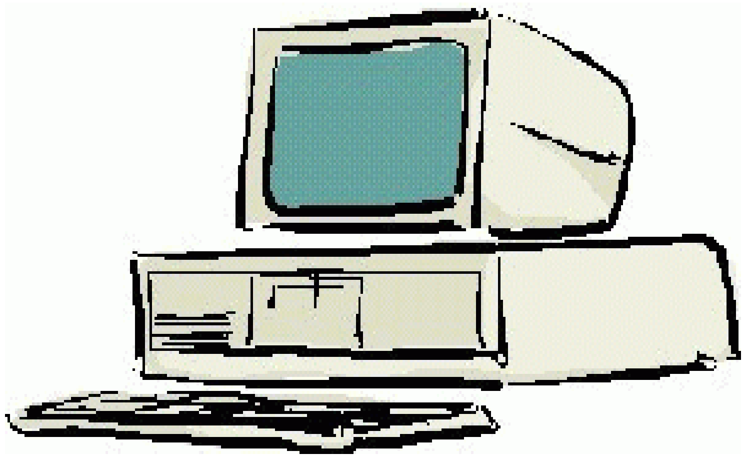
Programming in C



Session 8

Seema Sirpal
Delhi University Computer Centre

File I/O & Command Line Arguments

An important part of any program is the ability to communicate with the world external to it.

Reading input from files and writing results to files are simple, but effective ways to achieve that.

Command line arguments provide a way to pass a set of arguments into a program at run time.

These arguments could be the names of files on which to operate, user options to modify program behavior, or data for the program to process.

Library Functions for File I/O

Fopen is used to open a file for formatted I/O and to associate a stream with that file.

A stream is a source or destination of data. It may be a buffer in memory, a file or some hardware device such as a port.

The prototype for fopen is:

FILE *fopen(const char *filename, const char *mode);

Fopen returns a file pointer on success or NULL on failure.

The file pointer is used to identify the stream and is passed as an argument to the routines that read, write or manipulate the file.

The filename and mode arguments are standard null-terminated strings.

Library Functions for File I/O

The valid modes are shown below.

Mode	**Use**
r	open for reading
w	open or create for writing. Truncate (discard) any previous contents.
a	open or create for writing. Append (write after) any previous contents.

Library Functions for File I/O

Fflush is used to flush any buffered data out of an output stream.

Output streams may be buffered or unbuffered.

With buffered output, as data is written to the stream, the operating system saves this data to an intermediate buffer.

When this buffer is full, the data is then written to the file. This is done to reduce the number of system calls needed to write out the data.

The whole buffer is written at once. This is done to make the program more efficient and without any programmer involvement.

Fflush forces the buffer to be written out to the associated file.

Library Functions for File I/O

```
int fflush(FILE *stream);
```

It accepts a file pointer as an argument.

It returns zero on success and EOF on failure.

EOF is a special constant used to designate the end of a file.

Library Functions for File I/O

Files are closed with the function fclose. Its prototype is:

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

Fclose returns zero on success and EOF on failure.

Library Functions for File I/O

Data is written to a file using fprintf.

This function is very similar to printf.

Printf was used to write to standard output, stdout.

Fprintf has one additional argument to specify the stream to send data.

Its prototype is:

int fprintf(FILE *stream, const char* format, ...);

Fprintf returns the number of characters written if successful or a negative number on failure.

Library Functions for File I/O

Data is read from a file using fscanf.

This function is very similar to scanf which is used to read from standard input, stdin.

Fscanf has one additional argument to specify the stream to read from. Remember that the argument to store data must be pointers.

The prototype for fscanf is:

```
int fscanf(FILE *stream, const char* format, ....);
```

Other Useful Standard Library Functions for Input & Output

```
int fgetc(FILE *stream);
```

This function returns the next character in the input stream, or EOF if the end of the file is encountered.

It returns int rather than char because the "end of file", EOF, character must be handled.

EOF is an int and is too large to fit in a char variable.

Other Useful Standard Library Functions for Input & Output

int *fgets(char *s, int n, FILE *stream);

It returns a pointer to the character string if successful, or NULL if end of file is reached or if an error has occurred.

The string is also read into the character array specified as an argument. The character string will be terminated be a "\0", which is standard for C.

At most n-1 characters will be read. This allows for storage of the string terminator, '\0'.

Other Useful Standard Library Functions for Input & Output

int fputs(const char s*, FILE *stream);

This function writes a string to the stream. It returns EOF on failure.

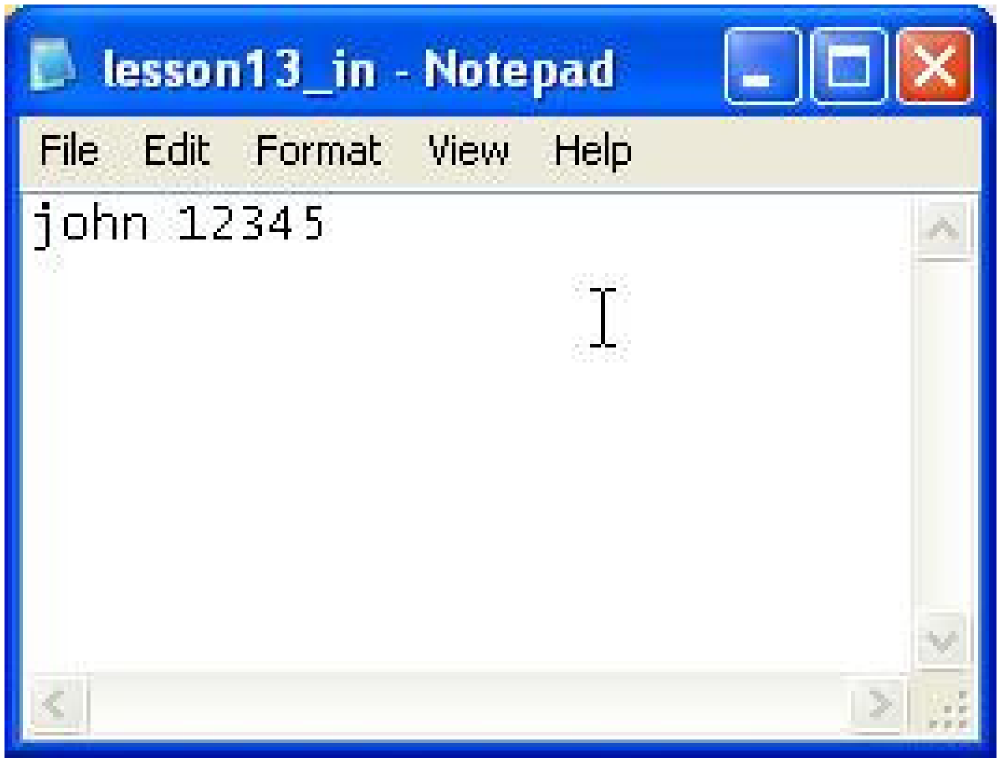
int fputc(int c, FILE *stream);

Fputc writes a single character to the output stream. It returns EOF on failure.

Example

The following program opens a file containing a name and an ID number.

It then opens an output file and writes this data to an output file. This input file was created with a text editor.



```
john 12345
```

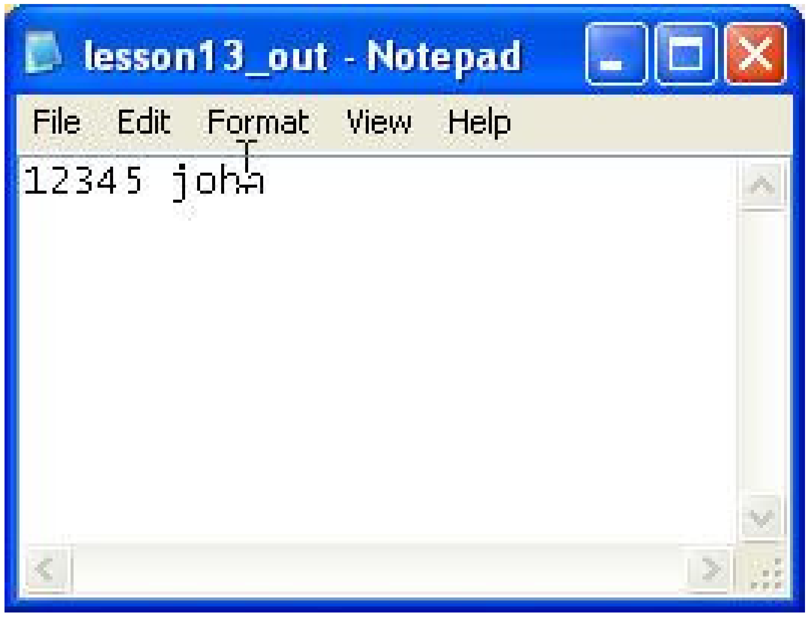
Sample Program

program

```
#include <stdio.h>

int main()
{
   char ifilename[] = "c:/lesson13_in.txt";
   char ofilename[] = "c:/lesson13_out.txt";
   char name[30];
   int idNum;
   FILE *ofp, *ifp;

   /* Open file for input */
   ifp = fopen(ifilename,"r");

   /* Read data */
   fscanf(ifp,"%s %d",name,&idNum);

   /* Open file for output */
   ofp = fopen(ofilename,"w");

   /* Write out data */
   fprintf(ofp,"%d %s\n",idNum, name);

   /* Close Files */
   fclose(ifp);
   fclose(ofp);

   return 0;
}
```

Sample Program

output



```
12345 john
```

Practice

Extend the above example so it will read and write multiple pairs of names and ids.

Hint: Create a loop and check the return code of fscanf for EOF.

Solution

```
#include <stdio.h>

int main()
{
   char ifilename[] = "c:/lesson13_in.txt";
   char ofilename[] = "c:/lesson13_out.txt";
   char name[30];
   int idNum;
   FILE *ofp, *ifp;

   /* Open file for input */
   ifp = fopen(ifilename,"r");

   /* Open file for output */
   ofp = fopen(ofilename,"w");

   /* Read and write data */
   while (fscanf(ifp,"%s %d",name,&idNum) != EOF)
   {
   /* Write out data */
   fprintf(ofp,"%d %s\n",idNum, name);
   }

   /* Close Files */
   fclose(ifp);
   fclose(ofp);

   return 0;
}
```

stdin, stdout, stderr

Stderr and stdout are predefined streams available for output.

Both are usually defined to be the computer's screen.

Stdout is a buffered stream;

stderr is unbuffered.

Stdin is a predefined input stream, usually defined to be the keyboard.

In previous Sessions, output was written to stdout using the printf function.

printf("Hello World\n"); */* Buffered output */*

stdin, stdout, stderr

Output may also be directed to stdout or stderr using the fprintf function.

fprintf(stdout,"Hello World\n"); */* Buffered output, same as printf() */*

fprintf(stderr,"Hello World\n"); */* Unbuffered output */*

Buffered vs. unbuffered output has important implications for error messages and prints used for debugging.

If a program dies, core dumps, while running, buffers are not flushed.

That means error messages sent to stdout may not be seen. Since stderr is unbuffered, all error messages are immediately output. Thus, error messages and debugging statements should be directed to stderr rather than stdout.

stdin, stdout, stderr

Stdin is usually the keyboard. In previous lessons, input was read from stdin using scanf().

```
scanf("%d",&someInt);
```

This may also be done using fscanf as follows.

```
fscanf(stdin,"%d",&someInt);
```

Command Line Arguments

C provides a mechanism to pass command line arguments into a program when it begins to execute.

When execution begins, "main" is called with two arguments, a count and a pointer to an array of character strings.

The count is by convention called argc.

The pointer is usually called argv. The use of argv is somewhat tricky. Since argv is a pointer to an array of character strings, the first character string is referenced by argv[0] (or by *argv). The second character string is referenced by argv[1] (or by *(argv + 1)), the third by argv[2], and so on. The first character string, argv[0], contains the program name. Command line arguments begin with argv[1]. If the number of arguments will be fixed, the count, argc, should always be checked. Here is a simple example program that performs an echo. Note that the number of strings to echo need not be hard coded. Argc is used to determine the number of strings passed as command line arguments.

Practice

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int i;
    fprintf(stdout,
        "The number of command line arguments is %d\n",
        argc);
    fprintf(stdout,"The program name is %s\n",argv[0]);

    for (i = 1; i < argc; i++)
    {
        fprintf(stdout,"%s",argv[i]);
    }
    fprintf(stdout,"\n");
    return 0;
}
```