I/O in C

Input/Output in C

- C has no built-in statements for input or output.
- A library of functions is supplied to perform these operations. The I/O library functions are listed the “header” file <stdio.h>.
- You do not need to memorize them, just be familiar with them.

Streams

- All input and output is performed with streams.
- A "stream" is a sequence of characters organized into lines.
- Each line consists of zero or more characters and ends with the "newline" character.
- ANSI C standards specify that the system must support lines that are at least 254 characters in length (including the newline character).

Types of Streams in C

- Standard input stream is called "stdin" and is normally connected to the keyboard
- Standard output stream is called "stdout" and is normally connected to the display screen.
- Standard error stream is called "stderr" and is also normally connected to the screen.

Formatted Output with printf

printf ( ) ;

- This function provides for formatted output to the screen. The syntax is:
  printf (“format”, var1, var2, …) ;
- The “format” includes a listing of the data types of the variables to be output and, optionally, some text and control character(s).
- Example:
```
float a ;  int b ;
scanf ( “%f%d”, &a, &b ) ;
printf ( “You entered %f and %d \n”, a, b ) ;
```

Formatted Output with printf

- Format Conversion Specifiers:
  d -- displays a decimal (base 10) integer
  l -- used with other specifiers to indicate a "long"
  e -- displays a floating point value in exponential notation
  f -- displays a floating point value
  g -- displays a number in either "e" or "f" format
  c -- displays a single character
  s -- displays a string of characters

Input/Output in C

scanf ( ) ;

- This function provides for formatted input from the keyboard. The syntax is:

  scanf (“format” , &var1, &var2, …) ;

- The “format” is a listing of the data types of the variables to be input and the & in front of each variable name tells the system WHERE to store the value that is input. It provides the address for the variable.
- Example:

```
float a;  int b;
scanf (“%f%d”, &a, &b);
```

Input/Output in C

getchar ( ) ;

- This function provides for getting exactly one character from the keyboard.
- Example:

```
char ch;
ch = getchar ( ) ;
```

Input/Output in C

putchar (char) ;

- This function provides for printing exactly one character to the screen.
- Example:

```
char ch;
ch = getchar ( ) ;  /* input a character from kbd */
putchar (ch) ;       /* display it on the screen */
```

Input/Output in C

getc (*file) ;

- This function is similar to getchar( ) except the input can be from the keyboard or a file.
- Example:

```
char ch;
ch = getc (stdin) ;   /* input from keyboard */
ch = getc (fileptr) ; /* input from a file */
```

Input/Output in C

putc (char, *file) ;

- This function is similar to putchar ( ) except the output can be to the screen or a file.
- Example:

```
char ch;
ch = getc (stdin) ;   /* input from keyboard */
putc (ch, stdout) ;   /* output to the screen */
putc (ch, outfileptr) ;        /*output to a file */
```

File I/O in C

Files in C

- In C, each file is simply a sequential stream of bytes. C imposes no structure on a file.
- A file must first be opened properly before it can be accessed for reading or writing. When a file is opened, a stream is associated with the file.
- Successfully opening a file returns a pointer to (i.e., the address of) a file structure, which contains a file descriptor and a file control block.

Files in C

- The statement:

```
        FILE *fptr1, *fptr2 ;
```

  declares that *fptr1* and *fptr2* are pointer variables of type FILE. They will be assigned the address of a file descriptor, that is, an area of memory that will be associated with an input or output stream.
- Whenever you are to read from or write to the file, you must first open the file and assign the address of its file descriptor (or structure) to the file pointer variable.

Opening Files

- The statement:

```
        fptr1 = fopen ( "mydata", "r" ) ;
```

  would open the file *mydata* for input (reading).
- The statement:

```
        fptr2 = fopen ("results", "w" ) ;
```

  would open the file *results* for output (writing).
- Once the files are open, they stay open until you close them or end the program (which will close all files.)

Testing for Successful Open

- If the file was not able to be opened, then the value returned by the *fopen* routine is NULL.
- For example, let's assume that the file *mydata* does not exist. Then:

```
   FILE *fptr1 ;
   fptr1 = fopen ( "mydata", "r") ;
   if (fptr1 == NULL)
   {
           printf ("File 'mydata' did not open.\n") ;
   }
```

Reading From Files

- In the following segment of C language code:

```
        int a, b ;
        FILE *fptr1, *fptr2 ;
        fptr1 = fopen ( "mydata", "r" ) ;
        fscanf ( fptr1, "%d%d", &a, &b) ;
```

  the *fscanf* function would read values from the file "pointed" to by *fptr1* and assign those values to *a* and *b*.

End of File (1)

- The end-of-file indicator informs the program when there are no more data (no more bytes) to be processed.
- There are a number of ways to test for the end-of-file condition. One is to use the *feof* function which returns a true or false condition:

```
   fscanf (fptr1, "%d", &var) ;
   if ( feof (fptr1) )
   {
           printf ("End-of-file encountered.\n");
   }
```

End of File (2)

- Another way to test for the end-of-file condition is to use the value returned by the *fscanf* function:

```
int istatus ;
istatus = fscanf (fptr1, "%d", &var) ;
if ( istatus == EOF )
{
        printf ("End-of-file encountered.\n”) ;
}
```

Writing To Files

- In the following segment of C language code:

```
int a = 5, b = 20 ;
FILE  *fptr2 ;
fptr2 = fopen ( "results", "w" ) ;
fprintf ( fptr2, "%d %d\n", a, b ) ;
```

the fprintf functions would write the values stored in *a* and *b* to the file "pointed" to by *fptr2*.

Closing Files

- The statements:

```
fclose ( fptr1 ) ;
fclose ( fptr2 ) ;
```

will close the files and release the file descriptor space and I/O buffer memory.

Reading and Writing Files

```
#include <stdio.h>
int main ( )
{
   FILE *outfile, *infile ;
   int b = 5, f ;
   float a = 13.72, c = 6.68, e, g ;

   outfile = fopen ("testdata", "w") ;
   fprintf (outfile, "%6.2f%2d%5.2f", a, b, c) ;
   fclose (outfile) ;
   infile = fopen ("testdata", "r") ;
   fscanf (infile,"%f %d %f", &e, &f, &g) ;

   printf ("%6.2f,%2d,%5.2f\n", e, f, g) ;
}
```

Output:

```
12345678901234567890
**********************
 13.72 5 6.68
```