Input and Output

Chapter 7 K &R deals with Input and Output

Input and Output

Input and Output facilities are not a part of the C language.

All I/O operations must be carried out through function calls.
e.g.   getchar ( ), printf ( ), scanf ( )

Since these functions are not a part of the language itself they may differ from one machine to another.

The ANSI standard defines these (and other) library functions to ensure compatibility of systems that run C

Standard Library

The standard library provides a set of functions for C programs:

- Input and Output
- String Handling
- Storage Management
- Mathematical Routines
- etc...

Standard Library

The properties of library functions are included in several headers such as :

< stdio.h>, < string.h >, < ctype.h>, ...

e.g. #include < stdio.h>

This include file contains declarations and macro definitions associated with Standard I/O Library.

Whenever using a function from this library we include this file at the front of the program.

Stream Model for Standard I/O

The model of input and output supported by the Standard Library is a simple one.

Text input or output regardless of its origin or destination is modeled as a stream of characters.

A text stream is a sequence of characters divided into lines. Each line consists of zero or more characters followed by a newline character.

Stream Model for Standard I/O (contd)

It is the responsibility of the library to make each input or output stream conform to this model.

for e.g. the library may convert Carriage Return (CR) and Line Feed (LF) to newline on input and back again on output.

A C program using the library need not worry about how lines are represented outside the program.

getchar ( )

The simplest input mechanism to read one character at a time from the standard input (keyboard) is

```
int   getchar (void)
```

getchar returns the next input character each time it is called, or EOF when it encounters end of file. EOF is a symbolic consatnt defined in   <stdio.h>.

Typically EOF has the value -1. But tests should be written in terms of EOF as to be independent of the specific value.

e.g.
```
while  ( (c  =  getchar ( ) )  !=  EOF)
                putchar (c);
```

Input Redirection

A file may be substituted for the keyboard by input redirection:

```
prog   <infile
```

The above command line causes the program prog that uses getchar to read characters from the file infile (instead of reading from the keyboard)

Here the switching of the input is done in a way that the program prog itself is unaware of the change.

Input Redirection

A pipe (|) allows the output of a program to form the input of another program .

```
who | grep john | wc -l
```

Input switching is also invisible if the input comes from another program via a pipe mechanism.

e.g `prog1 | prog2`

causes the standard output of prog1 to be piped into the standard input for prog2

Output and Output Redirection

int putchar (int)

The function putchar (c) puts the character c on the standard output (screen by default).
putchar returns the character written or EOF if an error occurs.

printf output goes to standard output

```
prog  >outfile
```

will write the standard output to outfile instead.

```
prog1 | prog2
```

puts the standard output of prog1 to the standard input of prog2.

Formatted Output - printf

```
int  printf( char *format,  arg1,  arg2, ...  )
```

printf function converts, formats, and prints its argument on the standard output under control of format.

It returns the number of characters printed.

Formatted Output - printf

```
int i  = 425;
printf (" % d    %o     %x   %X \n" ,  i ,  i,  i, i);
```

The output will be

```
425   651   1a9  1A9
```

```
float f  = 12.978;
printf (" % 7.2f    %7.2e\n" ,  f ,  f);
```

The output will be

```
  12.98   1.30e+01
```

Formatted Output - printf

```
double  d  = -97.4583;
printf (" % .*f\n" ,  3, d);
```

The asterisk after the period in the format specification instructs printf to take the next argument to the function as the value of the precision.

The output will be

```
-97.458
```

Formatted Output - printf

```
char  s[ ]  = "abcdefghij";
printf (" % .5s\n" ,  s);
```

The output will be

```
abcde
```

```
printf (" % .*s\n" , max,  s);
```

To print at most max characters from string s

```
printf (" %15s\n" ,  s);
     abcdefghij   /* right justified output */
```

```
printf (" %-15s\n" ,  s);
abcdefghij        /* left justified output */
```

Formatted Output - sprintf

```
int  sprintf( char *string, char *format, arg1, ...  )
```

sprintf ( ) formats the arguments arg1, ..., according to the format and places the result in string.

```
sprintf  (text, "%d + %d", 20, 50);
```

will place the character string "20 + 50" in text

Formatted Input - scanf

int scanf(char *format, arg1, arg2, ...)

scanf function is the input analog of printf providing many of the same conversion facilities in the opposite direction.

scanf reads the characters from standard input, interprets them according to specification in format and stores the results through the remaining arguments.

Each of the arguments (i.e. arg1, arg2, ...) must be a pointer

Formatted Input - scanf

scanf stops when it exhausts the format string or when some input fails to match the control specification.

scanf returns as its value the number of successfully matched and assigned input items

On the end of file, EOF is returned

Formatted Input - scanf

```
scanf (" % d:%d:%d" , &hour, &minutes,&seconds);
```

Three integer values are to be read and stored in the variables hour, minutes and seconds respectively.

Here in the format string the : characters specify that colons are expected as separaters between the three integer values.

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

To specify a percentage sign is expected as input.

Formatted Input - scanf

```
scanf  ("%d%c", &i, &c);
```

with the text line

```
        29    w
```

would assign the value 29 to i and a blank space character to c since this is the character that appears immediately after the character 29 on the input.

```
scanf  ("%d  %c", &i, &c);
```

would assign the value 29 to i and w to c (here scanf ignores all leading white spaces after 29 is read)

Formatted Input - sscanf

int sscanf(char *string, char *format, arg1, ...)

sscanf reads from a string instead of the standard input, according to the format specification and stores the results through arg1, ...

sscanf ("Nov 28", "%s%d", month, &day);

stores the string "Nov" inside month, assumed to be a character array, and will assign the integer value 28 to day (an integer variable)

File Access

stdio.h contains a structure declaration called FILE

A file has to be opened before reading or writing

The file has to be opened using the library function fopen

fopen takes the file name and returns a pointer called file pointer which can be used for subsequent reading or writing of the file.

fopen ( )

```
FILE    *fp;       /* declares fp as a pointer to FILE */
FILE    *fopen (char  *name,  char *mode);

     fp  =  fopen (name, mode);
```

where name is a character string containg the name of the file and

mode also a character string indicates how one intends to use the file.

Allowed modes are read ("r"), write ("w") and append ("a"),

fopen ( )

```
FILE    *fp;       /* declares fp as a pointer to FILE */
FILE    *fopen (char  *name,  char *mode);

     fp  =  fopen ("data_file", "r");
```

opens a file called data_file in read mode
and assigns to fp the unique pointer that identifies
the file.

If the file cannot be opened for some reason , then
fopen() returns the value NULL (also defined in
the standard I/O include file)

fopen ( )

```
FILE     *fp;       /* declares fp as a pointer to FILE */
FILE     *fopen (char  *name,  char *mode);

   if ( (fp  =  fopen ("data_file", "r") ) == NULL)
            printf ( " file cannot be opened \n");
```

will indicate whether the fopen ( ) is successful or not.

getc ( ) and putc ( )

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

getc returns the next character from the file referred to by fp , it returns EOF for end of file or error.
naturally getc is used for reading a file.

```
int  putc ( int c, FILE *fp);
```

putc writes a character c to the file fp and returns the character written or EOF if error occurs.

fscanf ( ) and fprintf ( )

fscanf ( ) and fprintf ( ) used for formatted input and output of files respectively

int fscanf (FILE *fp, char *format, ...)

int fprintf (FILE *fp, char *format, ...)

fclose ( )

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

fclose is the inverse of fopen, it breaks the connection between the file pointer and the external name that was established by fopen ( ), freeing the file pointer for another file.

stdin, stdout, stderr

Whenever a C program is executed, three "files" are automatically opened by the system for use by the program.

These files are identified by the constant FILE pointers stdin, stdout and stderr which are defined in stdio.h

The FILE pointer stdin identifies the standard input of the program and is associated with the terminal.

stdout is the standard output treated similarly

stdin, stdout, stderr

The FILE pointer stderr identifies the standard error file. The error messages produced by the program are written here.

stderr is normally associated with the terminal

```
if ( (fp  =  fopen ("data_file", "r") ) == NULL)
     fprintf ( stderr,"file cannot be opened \n");
```

exit ( )

A program can signal error in two ways:

stderr and exit

exit ( ) terminates program execution when it is called.
The argument of exit ( ) is available to whatever
process called this one.

return value 0 signals no error. Non zero values
usually signals abnormal situations.