Formatted Input and Outputs

Consoled input output functions can be classified into 2 categories:

* Formatted
* Unformatted

Formatted Functions allow the input read from the keyboard or output to the screen to be formatted as per our requirements.

* Formatted functions for char, float, string and int are scanf and printf.
* Unformatted functions for char() getChar() and putChar() are string => gets() and puts().

Field Width (w) - Digits specifying the field width.

Precision (d) - The number of places after the decimal point.

## Width

The field width tells printf, how many columns on the screen should be used while printing a value.

* In the value to be printed happens not to fill up the entire field, the value is right justified and is padded with blanks on the left.
* If we include the (-) sign in the format specifier this means left justification is desired and the value will be padded with blanks on the right.
* If the field width used turns out to be less than what is required to print the number, the field width is ignored and the complete number is printed.

# Precision

Specifying field width can be useful in creating tables of numeric values with numbers lined up properly.

* Specifier “%10.1f” specifies that a float be printed right-aligned within ten columns with one place beyond the decimal point.
* All escape sequences begin with “/“.

\n = New Line

\v = Vertical Tab

\r = Carriage return

\a = Alert

\“” = “”

\? = ?

\t = Tab

\b = Backspace

\“\” = Backslash a.k.a \

\’’ = ‘’

* \n takes the cursor to the beginning of the new
* \v moves the cursor in the next line below it’s current position
* \b moves the cursor moves the cursor one position to the left of its current position.
* \a alerts the user by sounding the speaker in the computer.
* /r positions the cursor to the beginning of the line in which it is currently placed
* /t = while using /t an 80 column screen is divided into 10 print zones of 8 columns.
* Using a /t moves the cursor to the beginning of the next print zone
* Characters ‘’, “”, \ and ? Can be printed by preceding them with the \

## Ignoring Inputs

Sometimes we may wish to ignore some of the characters supplied as input, this can be done using %\*c.

## Mismatch

* If there is a mismatch in the specifier and the type of value being printed, printf tend to perform the specified conversion and does it’s best to produce a proper result.
* Sometimes the result will be any random garbage value, as in the case when we ask it to print a string using %d.

## sprintf() and sscanf()

The s printf function works similar to the printf function except for one small difference, instead of sending the output to the screen as printf does, this function writes the output to a string.

The counter part of sprint is the scan function, it allows us to read characters from a string and to convert and store them in c variables according to specified formats.

Unformatted I/O Functions

## fgetchar() and fputchar()

The **fgetchar()** lets you read a single character entered through the keyboard. The character that is typed has to followed by the enter key, it’s counterpart is **fputchar()** which displays the character on the screen.

## gets()

gets receives a string from the keyboard, it is terminated when an enter key is hit. Thus, space and char are perfectly acceptable as part of the input string. Gets function gets a new line as a function \n terminated string of characters from the keyboard and replaces the \n with a zero

## puts()

The puts() works exactly opposite to gets() it outputs a string to the screen. Unlike **printf()**, puts can output only one string at a a time. If we attempt to print two string using **puts()**, only the first one gets printed. Similarly, unlike **scanf()**, **gets()** can be used to read only one string at a time.