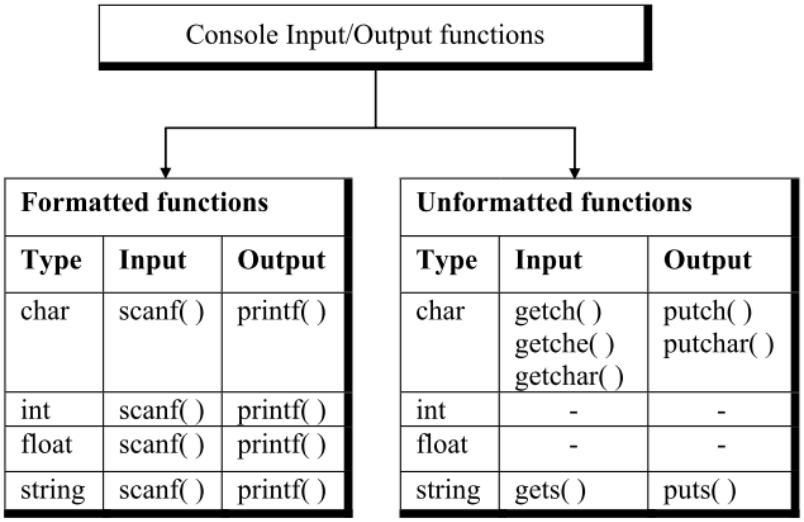
Chapter 4

Input and Output

One of the essential operations performed in a C language is to provide input values to the program and output the data produced by the program to a standard output device. We can assign values to variable through assignment statements such as x = 5, a = 0; and so on. Another method is to use the Input, **scanf**() which can be used to read data from a key board. For outputting results we have used extensively the function **printf**() which sends results out to a terminal. There exists several functions in 'C' language that can carry out input-output operations. These functions are collectively known as **Standard Input/Output** Library. Each program that uses standard input/output function must contain the statement. **#include<stdio.h>** at the beginning.



Formatted functions		
Type	**Input**	**Output**
char	scanf( )	printf( )
int	scanf( )	printf( )
float	scanf( )	printf( )
string	scanf( )	printf( )

Unformatted functions		
Type	**Input**	**Output**
char	getch( ) getche( ) getchar( )	putch( ) putchar( )
int	-	-
float	-	-
string	gets( )	puts( )

Single Character Input/Output:

The basic operation done in input/output is to read a characters from the standard input device such as the keyboard and to output or writing it to the output unit usually the screen. The getchar function can be used to read a character from the standard input device. The scanf can also be used to achieve the function. The getchar has the following form:

variable_name = getchar():

Example#1:

```
# include<stdio.h>
#include<conio.h>
void main( ){
    char C;
    printf("Type one character:") ; // message to user
    C = getchar() ; // get a character from key board and  stores it in variable C.
    printf(" The character you typed is = %c", C) ; // output
    getch();}
```

The putchar function which in analogous to getchar function can be used for writing characters one at a time to the output terminal. The general form is

putchar (variable_name);

where variable is a valid C type variable that has already been declared. **putchar( );** displays the value stored in variable C to the standard screen.

Example#2: Program shows the use of getchar function in an interactive environment.

```
#include<stdio.h>
#include<conio.h>
void main( ){
    char in; // character declaration of variable in.
    printf (" please enter one character"); // message to user
    in = getchar( ) ; // assign the keyboard input value to in.
    printf(" The character you typed is %c.\n", in) ; // output
    putchar(in); // output 'in' value to standard screen.
    getch();
}
```

String Input/Output:

The *gets* function relieves the string from standard input device while *puts* function outputs the string to the standard output device. A string is an array or set of characters.

The function *gets* accepts the name of the string as a parameter, and fills the string with characters that are input from the keyboard till newline character is encountered. (That is till we press the enter key). All the end function gets appends a null terminator as must be done to any string and returns. The *puts* function displays the contents stored in its parameter on the standard screen.

The standard form of the *gets* function is **gets (str);** Here, str is a string variable.

The standard form for the *puts* function is **puts (str);** Where, str is a string variable.

Example#3: Program involving both gets and puts

```
#include<stdio.h>
#include<conio.h>
void main( ){
    char s[80]; printf("Type a string less than 80 characters: ");
    gets(s);
    printf("\nThe string types is: ");
    puts(s);  getch();
}
```

Example#4: Checking Character Type

```
#include<stdio.h>
#include  <ctype.h>
void main(){
      char character; printf("Press any key\n");
      character = getchar();
      if (isalpha(character) > 0){
        printf("The character is a letter.");
      }else{
        if (isdigit (character) > 0){
          printf("The character is a digit.");
        }else{
          printf("The character is not alphanumeric.");
        }
      }
}
```

Output

```
Press any key
h
The character is a letter.
```

Example#5: Reading & Writing of Alphabets in Reverse Case

```
#include<stdio.h>
#include <ctype.h>
void  main(){
      char alphabet;
      printf("Enter an alphabet");
      putchar('\n');   /* move to next line */
      alphabet = getchar();
      if (islower(alphabet)){ putchar(toupper(alphabet));
      }else{ putchar(tolower(alphabet)); }
  }
```

Output

```
Enter an alphabet
a
A
Enter an alphabet
Q
q
Enter an alphabet
z
Z
```

Formatted Input

The formatted input refers to input data that has been arranged in a particular format. Input values are generally taken by using the scanf function.

The scanf function has the general form:

scanf ("control string", arg1, arg2, arg3, …, argn);

The *format field* is specified by the *control string* and the arguments arg1, arg2, …, argn specifies the address of location where address is to be stored.

The *control string* specifies the field format which includes *format specifications* and optional number specifying *field width* and the conversion character % and also blanks, tabs and newlines.

The blanks tabs and newlines are ignored by compiler. The conversion character % is followed by the type of data that is to be assigned to variable of the assignment. The field width specifier is optional.

Input Specifications for Integer Number

The general format for reading a integer number is **%xd**

Here percent sign (**%**) denotes that a specifier for conversion follows and **x** is an integer number which specifies the width of the field of the number that is being read. The data type character d indicates that the number should be read in integer mode.

Example: scanf("%d %d", &a, &b);

If the values input are 175 and 1342 here value 175 is assigned to **a** and 1342 to **b**.

Suppose the input data was follows 1342 and 175 for the program with scanf("%3d %d", &x, &y); The number 134 will be assigned to **x** and **y** has the value 2 because of %3d the number 1342 will be cut to 134 and the remaining part is assigned to second variable **y**. If floating point numbers are assigned then the decimal or fractional part is skipped by the computer. To read the long integer data type we can use conversion specifier **%ld** & **%hd** for short integer.

Input Specifications for Real Number

Field specifications are not to be use while representing a real number therefore real numbers are specified in a straight forward manner using **%f** specifier. The general format of specifying a real number input is

scanf ("%f", &variable);

Example: scanf ("%f %f %f", &a, &b, &c); with the input data **321.76, 4.321, 678**

The values 321.76 is assigned to **a**, 4.321 to **b** & 678 to **c**. If the number input is a double data type then the format specifier should be **%lf** instead of **%f**.

Input Specifications for a Character

Single character or strings can be input by using the character specifiers. The general format is **%xc or %xs** where **c** and **s** represents *character* and *string* respectively and **x** represents the *field width*. The address operator need not be specified while we input strings.

Example: scanf ("%c %15c", &ch, name): Here suppose the input given is "**a, Sunil"** then **a** is assigned to *ch* and **Sunil** will be assigned to *name*.

Formatted Output

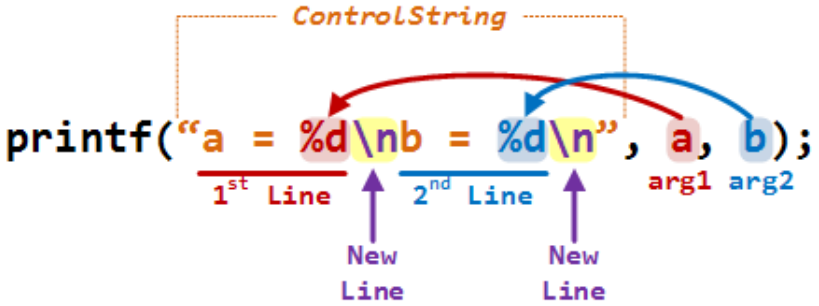
Printing One Line

printf(); The most simple output statement can be produced in C' Language by using printf statement. It allows you to display information required to the user and also prints the variables we can also format the output and provide text labels.

Conversion Strings and Specifiers

The printf( ) function is quite flexible. It allows a variable number of arguments, labels and sophisticated formatting of output.

The general form of the printf( ) function is **printf ("conversion string", variable_list);**



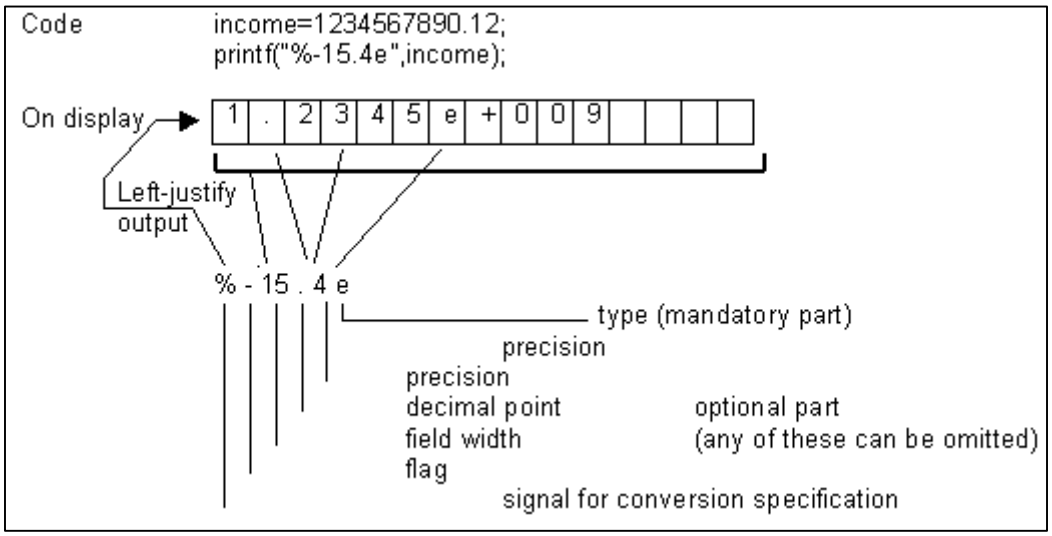
The conversion string includes all the text labels, escape character and conversion specifiers required for the desired output. The variable includes all the variable to be printed in order they are to be printed. There must be a conversion specifies after each variable.

Integer Output

Format	Output					
printf("%d",9876)	9	8	7	6		
printf("%6d",9876)			9	8	7	6
printf("%2d",9876)	9	8	7	6		
printf("%-6d",9876)	9	8	7	6		
printf("%06d",9876)	0	0	9	8	7	6

*Note: **0** pads with zeros the leading blanks, **-** used for left justification*

Real Number Output



float y = 98.7654;

Format	**Output**										
printf("%7.4f",y)	9	8	.	7	6	5	4				
printf("%7.2f",y)			9	8	.	7	7				
printf("%-7.2f",y)	9	8	.	7	7						
printf("%f",y)	9	8	.	7	6	5	4				
printf("%10.2e",y)			9	.	8	8	e	+	0	1	
printf("%11.4e",-y)	-	9	.	8	7	6	5	e	+	0	1
printf("%-10.2",y)	9	.	8	8	e	+	0	1			
printf("%e",y)	9	8	.	7	6	5	4				

Some system supports **printf("%*.*f", width, precision, number)**. For Example:

printf("%.*f", 7, 2, number)* **is equivalent to** *printf("%7.2f", number)*

String Output

"NEW YEAR 2075"

Specification	**Output**																			
%s	N	E	W		Y	E	A	R		2	0	7	5							
%20s								N	E	W		Y	E	A	R		2	0	7	5
%20.10s											N	E	W		Y	E	A	R		2
%.5s	N	E	W		Y															
%-20.10s	N	E	W		Y	E	A	R		2										
%5s	N	E	W		Y	E	A	R		2	0	7	5							

Mixed Data Output

printf("%d %s %f %c", a, b, c, d); // Outputs Integer String Float Character

Commonly Used printf Format Codes

Specifier	Meaning
%c	Print a character
%d	Print a Integer
%i	Print a Integer
%e	Print float value in exponential form.
%f	Print float value
%g	Print using %e or %f whichever is smaller
%o	Print octal value
%s	Print a string
%x	Print a hexadecimal integer (Unsigned) using lower case a-f
%X	Print a hexadecimal integer (Unsigned) using upper case A-F
%a	Print a unsigned integer.
%p	Print a pointer value
%hx	Print hex short
%lo	Print an octal long
%ld	Print a long integer