

CE143: COMPUTER CONCEPTS & PROGRAMMING

UNIT-4

Managing Input & Output Operations


N. A. Shaikh

nishatshaikh.it@charusat.ac.in

- Introduction
- Reading a Character
- Writing a Character
- Various library functions from ctype.h
- Formatted Input
- Formatted Output

- **Reading, processing, and writing** of data are the three essential functions of a computer program.
- Most programs take some data as input and display the processed data(information / results) as output.
- So we need some methods that can read input and write output on a suitable input/output medium.

So far we have seen two methods of providing data to the program variables.

1. One method is to assign values to variables through the assignment statements such as **x=5; a=0;** and so on.
 2. Another method is to use the input function **scanf** which can read data from a keyboard.
-
-  For outputting results we have used the function **printf** which sends results out to a terminal.

- All input/output operations are carried out through function calls such as **printf** and **scanf**
- There exist several functions that have more or less become standard for input and output operations in C.
- These functions are collectively known as the **standard input and output library**
- Each program that uses a standard input/output function must contain the statement

#include<stdio.h>

getchar(): Reads single character

Syntax:

```
variable_name = getchar();
```

Example:

```
char name;
```

```
name = getchar();
```

NOTE: getchar() accepts any character keyed in including RETURN and TAB.

putchar(): Writes a single character, one at a time.

Syntax:

putchar(variable_name);

Example:

char var='Y';

putchar(var);

putchar('\n'); //new line

getchar() & putchar()

```
#include <stdio.h>
int main()
{
    char c;
    printf("Enter some character\n");
    c = getchar();

    printf("Entered character is:\n");
    putchar(c);

    return 0;
}
```

```
Enter some character
g
Entered character is:
g
```

```
Enter some character
Hello
Entered character is:
H
```

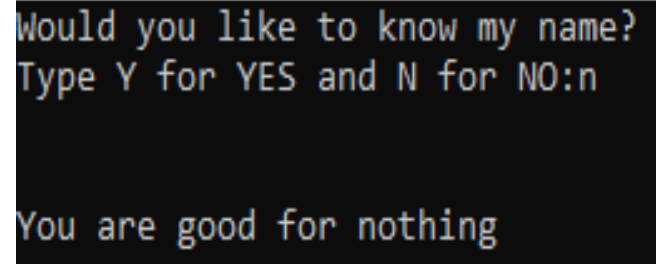
```
Enter some character
756
Entered character is:
7
```


getchar()

```
#include<stdio.h>
void main()
{
    char ans;
    printf("Would you like to know my name?\n");
    printf("Type Y for YES and N for NO:");

    ans=getchar();

    if(ans=='Y' || ans=='y')
        printf("\n\nMy name is BUSY BEE\n");
    else
        printf("\n\nYou are good for nothing\n");
}
```

A screenshot of a terminal window showing the output of the C program. The first two lines of output are "Would you like to know my name?" and "Type Y for YES and N for NO:", which correspond to the first two printf statements in the code. The third line of output is "You are good for nothing", which is the result of the else branch being executed after the user has entered a character (likely 'N' or 'n').

Would you like to know my name?
Type Y for YES and N for NO:n

You are good for nothing



Self-Study:

1. `getc()` & `putc()`
2. `getch()` & `putch()`
3. `getche()` & `putche()`

- Useful for **testing** and **mapping** characters.
- Program that uses a these character functions must contain the statement

`#include<ctype.h>`

ctype.h Library Function

Functions	Description
isalpha()	checks whether character is alphabetic
isdigit()	checks whether character is digit
isalnum()	Checks whether character is alphanumeric
isspace()	Checks whether character is space
islower()	Checks whether character is lower case
isupper()	Checks whether character is upper case
isxdigit()	Checks whether character is hexadecimal
iscntrl()	Checks whether character is a control character
isprint()	Checks whether character is a printable character
ispunct()	Checks whether character is a punctuation
isgraph()	Checks whether character is a graphical character
tolower()	Checks whether character is alphabetic & converts to lower case
toupper()	Checks whether character is alphabetic & converts to upper case

Character Class	Description
Digits	This is a set of whole numbers { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }.
Hexadecimal digits	This is the set of { 0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f }.
Lowercase letters	This is a set of lowercase letters { a b c d e f g h i j k l m n o p q r s t u v w x y z }.
Uppercase letters	This is a set of uppercase letters {A B C D E F G H I J K L M N O P Q R S T U V W X Y Z }.
Letters	This is a set of lowercase and uppercase letters.
Alphanumeric characters	This is a set of Digits, Lowercase letters and Uppercase letters.
Punctuation characters	This is a set of ! " # \$ % & ' () * + , - . / : ; < = > ? @ [\] ^ _ ` { } ~
Graphical characters	This is a set of Alphanumeric characters and Punctuation characters.
Space characters	This is a set of tab, newline, vertical tab, form feed, carriage return, and space.
Printable characters	This is a set of Alphanumeric characters, Punctuation characters and Space characters.
Control characters	In ASCII, these characters have octal codes 000 through 037, and 177 (DEL). Example :'\a' (alert), '\b' (backspace), '\f' (form feed), '\n' (new line), '\r' (carriage return), '\t' (horizontal tab), '\v' (vertical tab) and '\0' (null)
Blank characters	These are spaces and tabs.
Alphabetic characters	This is a set of Lowercase letters and Uppercase letters.

//C program to check whether a character is alphabet, digit or special character

```
#include<stdio.h>
```

```
#include<ctype.h>
```

```
void main()
```

```
{
```

```
    char c;
```

```
    printf("Press any key\n");
```

```
    c=getchar();
```

```
    if(isalpha(c)>0)
```

```
        printf("The character is a alphabet");
```

```
    else if(isdigit(c)>0)
```

```
        printf("The character is a digit");
```

```
    else
```

```
        printf("The character is a special character");
```

```
}
```

isalpha()

isdigit()

Press any key

z

The character is a alphabet

Press any key

5

The character is a digit

Press any key

\$

The character is a special character

//C program to checks whether character is alphanumeric or not.

#include<stdio.h>

#include<ctype.h>

isalnum()

int main()

{

char ch;

printf("Enter a character: ");

ch=getchar();

if(isalnum(ch))

printf("%c is an alphanumeric character.\n",ch);

else

printf("%c is not an alphanumeric character.\n",ch);

return 0;

}

Enter a character: f
f is an alphanumeric character.

Enter a character: 5
5 is an alphanumeric character.

Enter a character: \$
\$ is not an alphanumeric character.

```

/*C program to check whether a character is alphabet,
  digit or special character using if...else if*/
#include <stdio.h>
void main()
{
    char ch;
    printf("Enter any character: ");
    scanf("%c", &ch);

    if((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))
    {
        printf("'%c' is alphabet.", ch);
    }
    else if(ch >= '0' && ch <= '9')
    {
        printf("'%c' is digit.", ch);
    }
    else
    {
        printf("'%c' is special character.", ch);
    }
}

```



```
/*C program to check whether a character is alphabet,  
  digit or special character using ASCII value*/  
#include <stdio.h>  
void main()  
{  
    char ch;  
    printf("Enter any character: ");  
    scanf("%c", &ch);  
  
    if((ch >= 97 && ch <= 122) || (ch >= 65 && ch <= 90))  
    {  
        printf("'%c' is alphabet.", ch);  
    }  
    else if(ch >= 48 && ch <= 57)  
    {  
        printf("'%c' is digit.", ch);  
    }  
    else  
    {  
        printf("'%c' is special character.", ch);  
    }  
}
```

```

/*C program to print a character in reverse case
(lowercase->uppercase / uppercase->lowercase)*/
#include<stdio.h>
#include<ctype.h>
void main()
{
    char c;
    printf("Enter any alphabet");
    putchar('\n');    //move to next line
    c=getchar();

    if(islower(c))
        putchar(toupper(c));
    else
        putchar(tolower(c));
}

```

islower()

tolower()

toupper()

```

Enter any alphabet
b
B

```

```

Enter any alphabet
D
d

```

isspace()

```
#include <stdio.h>

void main()
{
    char ch;
    printf("Enter any character\n");
    scanf("%c", &ch);

    if ( isspace ( ch ) )
        printf ( "\nEntered character is space" ) ;
    else
        printf ( "\nEntered character is not space" ) ;
}
```

Enter any character

Entered character is space

isxdigit()

```
#include <stdio.h>

void main()
{
    char ch;
    printf("Enter any character\n");
    scanf("%c", &ch);

    if ( isxdigit ( ch ) )
        printf ( "\nEntered character is hexadecimal" ) ;
    else
        printf ( "\nEntered character is not hexadecimal" ) ;
}
```

Enter any character

A

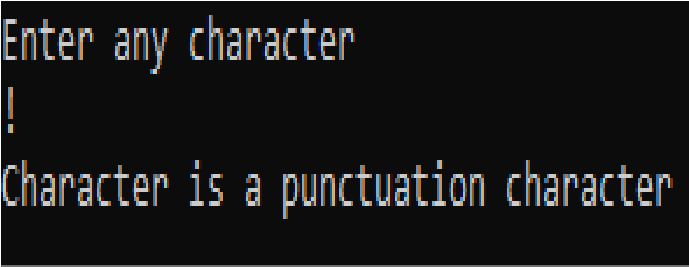
Entered character is hexadecimal

ispunct()

```
#include <stdio.h>

void main()
{
    char ch;
    printf("Enter any character\n");
    scanf("%c", &ch);

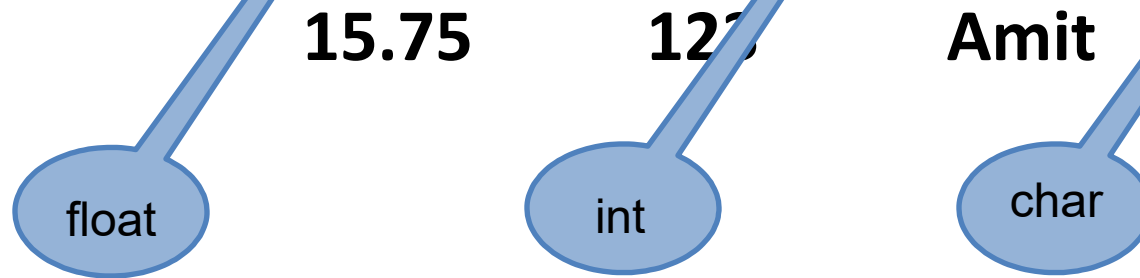
    if ( ispunct ( ch ) )
        printf ( "Character is a punctuation character" ) ;
    else
        printf ( "\nCharacter is not a punctuation character" ) ;
}
```



```
Enter any character
!  
Character is a punctuation character
```

- It refers to an input data that has been arranged in a particular format

Example:



- This is possible using **scanf** (scan formatted) function

Syntax:

```
scanf("control_string" , arg1 , arg2 , ....argn);
```

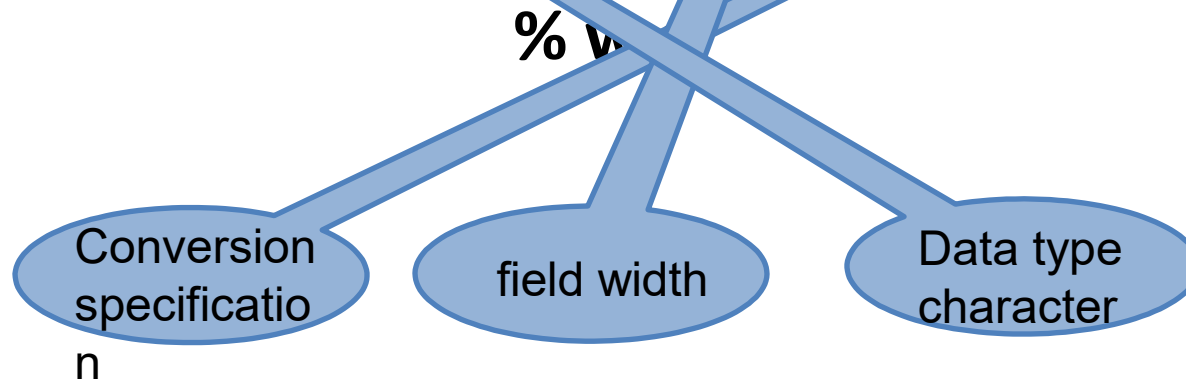
where,

control string / format string : specify the field format in which the data is to be entered

- Field(or format) specification includes,
 - conversion character %
 - Data type character(type specifier)
 - Field width **w**(optional)

Arguments(arg1,arg2...) : specify the address of locations where the data is stored.

The field specification for reading integer number:



Example:

```
scanf("%2d %5d", &num1, &num2);
```

Case-1: For input 50 and 31426

num1=50 & num2=31426

Case-2 : For input 31426 and 50

num1=31 & num2=426

NOTE: In Case-2, 50 is unread & will be assigned to first variable in next scanf call

Use the field specification without the field width specification to avoid errors

Example:

```
scanf("%d %d", &num1 , &num2);
```

For input 31426 and 50, num1=31426 & num2=50

- An input field may be skipped by specifying * in the place of field width

Example:

```
scanf("%d %*d %d", &a , &b);
```

For input

123 456 789

a=123 b=789

NOTE : 456 will be skipped because of *

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    int a,b,c,x,y,z,p,q,r;
```

```
    printf("Enter three numbers a, b and c\n");
```

```
    scanf("%d %d %d",&a,&b,&c);
```

```
    printf("%d %d %d\n\n",a,b,c);
```

```
    printf("Enter two 4-digits numbers x and y\n");
```

```
    scanf("%2d %4d",&x,&y);
```

```
    printf("%d %d\n\n",x,y);
```

```
    printf("Enter two integers numbers a and x\n");
```

```
    scanf("%d %d",&a,&x);
```

```
    printf("%d %d\n\n",a,x);
```

```
    printf("Enter a 9-digits numbers \n");
```

```
    scanf("%3d %4d %3d",&p,&q,&r);
```

```
    printf("%d %d %d\n\n",p,q,r);
```

```
    printf("Enter two three digits numbers\n");
```

```
    scanf("%d %d",&x,&y);
```

```
    printf("%d %d",x,y);
```

```
}
```

Enter three numbers a, b and c

1 2 3

1 3 0

Enter two 4-digits numbers x and y

6789 4321

67 89

Enter two integers numbers a and x

44 66

4321 44

Enter a 9-digits numbers

123456789

66 1234 567

Enter two three digits numbers

123 456

89 123

- It is legal to use a non-whitespace character between field specification.
- However, the `scanf` expects a matching character in the given location.

Example:

```
scanf("%d-%d", &a , &b);
```

For input

123-456

a=123 b=456

- Field width of real numbers is not to be specified
- scanf reads real numbers using simple specification **%f** for both notations
 - **Decimal** point notation
 - **Exponential** notation

Example:

```
scanf("%f %f %f", &x , &y , &z);
```

For input

475.89 43.21E-1 678

x=475.89 y=4.321 z=678.0

- For double type, specification should be %lf
- A number may be skipped using %*f specification

```
#include<stdio.h>
void main()
{
    float x,y;
    double p,q;

    printf("Enter value of x and y\n");
    scanf("%f %e",&x,&y);
    printf("\n x=%f \n y=%f",x,y);

    printf("\n-----\n");

    printf("Enter value of p and q\n");
    scanf("%lf %lf",&p,&q);
    printf("\n p=%.12lf \n q=%.12e",p,q);
}
```

```
Enter value of x and y
12.3456
17.5e-2
```

```
x=12.345600
y=0.175000
```

```
-----
Enter value of p and q
4.142857142857
18.5678901234567890
```

```
p=4.142857142857
q=1.856789012346e+001
```

- scanf function can input single character as well as strings(containing more than one character).
- The field specification for reading character strings:

%ws or %wc

- **%wc** used to read a single character
- **%ws** terminates the reading at the encounter of blank space.

Example:

```
char str[20];  
scanf("%4s", str);
```

- scanf also supports following conversion specifications for strings:

%[characters]

%[^characters]

%[characters]

- Only the characters specified within the brackets are permissible in the input string.
- The string will be terminated at the first encounter of other unspecified character.

%[^characters]

- The characters specified after the circumflex(^) are not permitted in the input string.
- The string will be terminated at the encounter of one of these characters.

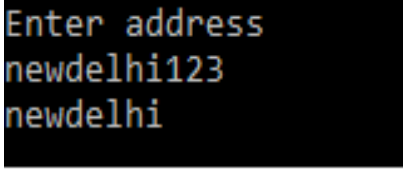
```
#include<stdio.h>
void main()
{
    char address[80];

    printf("Enter address\n");
    scanf("%[a-z]", address);
    printf("%s", address);
}
```

```
#include<stdio.h>
void main()
{
    char address[80];

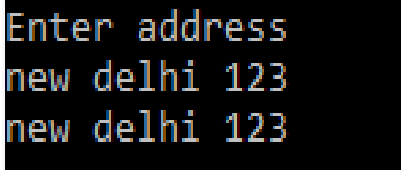
    printf("Enter address\n");
    scanf("%[^\\n]", address);
    printf("%s", address);
}
```

%[characters]



```
Enter address
newdelhi123
newdelhi
```

%[^characters]



```
Enter address
new delhi 123
new delhi 123
```


- It is possible to use one scanf statement to input a data line containing mixed data

Example:

```
scanf("%d %c %f %s",&count,&code,&ratio,name);
```

will read the following data correctly

15 p 1.575 cofee

NOTE:

- An attempt to read any unmatched type, the scanf function does not read any further item
- scanf function returns the value of number of items that are successfully read.

```
#include<stdio.h>

void main()
{
    int a;
    float b;
    char c;

    printf("Enter values of a, b and c\n");
    if(scanf("%d %f %c", &a, &b , &c)==3)
        printf("a=%d b=%f c=%c\n",a,b,c);
    else
        printf("Error in input.\n");
}
```

```
Enter values of a, b and c
12 3.45 A
a=12 b=3.450000 c=A
```

```
Enter values of a, b and c
23 78 9
a=23 b=78.000000 c=9
```

```
Enter values of a, b and c
8 A 5.25
Error in input.
```

```
Enter values of a, b and c
Y 12 67
Error in input.
```

```
Enter values of a, b and c
15.75 23 x
a=15 b=0.750000 c=2
```

Reading Mixed Data Types

Code	Meaning
%c	Read a single character
%d	Read a decimal integer
%e	Read a floating point value
%f	Read a floating point value
%g	Read a floating point value
%h	Read a short integer
%i	Read a decimal, hexadecimal or octal integer
%o	Read an octal integer
%s	Read a string
%u	Read an unsigned decimal integer
%x	Read a hexadecimal integer
%[..]	Read a string of word(s)

Following letters may be used as prefix

h for short integers

l for long integers or double

L for long double

General points to keep in mind while writing a scanf:

1. All function arguments, except the control string, must be pointers to variables.
2. Format specifications contained in the control string should match the arguments in order.
3. Input data items must be separated by spaces and must match the variables receiving the input in the same order.
4. The reading will be terminated, when scanf encounters a 'mismatch' of data or a character that is not valid for the value being read.
5. When searching for a value , scanf ignores line boundaries and simply looks for the next appropriate character
6. Any unread data items in a line will be considered as part of the data input line to the next scanf call
7. When the field width specifier w is used, it should be large enough to contain the input data size

- Each variable to be read must have a field specification
- For Each field specification, there must be a variable address of proper type
- Any non-whitespace character used in format string must have a matching character in the user input.
- Never end the format string with whitespace. It is an fatal error!
- The scanf reads until:
 - The white space character found in a numeric specification or
 - The maximum number of character have been read or
 - An error is detected or
 - The end of file is reached

- `printf` function is used for printing captions and numerical results.

Syntax:

`printf("control string", arg1, arg2, ..., argn);`

Control string consists,

1. Characters that will be printed on the screen as they appear
2. Format specifications that define the output format for display of each item
3. Escape sequence characters such as `\n`, `\t`, and `\b`.

Example:

```
printf("Programming in C");  
printf(" ");  
printf("\n");  
printf("%d", x);  
printf("a=%f \n b=%f", a, b);  
printf("\n\n");
```

The format specification for printing an integer number:

%w

Conversion
specificatio
n

Minimum
field width

Data type
character

Number is written **right justified** in given field width

Format

Output

```
printf("%d", 9876);
```

9	8	7	6
---	---	---	---

```
printf("%2d", 9876);
```

9	8	7	6
---	---	---	---

```
printf("%6d", 9876);
```

		9	8	7	6
--	--	---	---	---	---

```
printf("%-6d", 9876);
```

9	8	7	6		
---	---	---	---	--	--

```
printf("%06d", 9876);
```

0	0	9	8	7	6
---	---	---	---	---	---

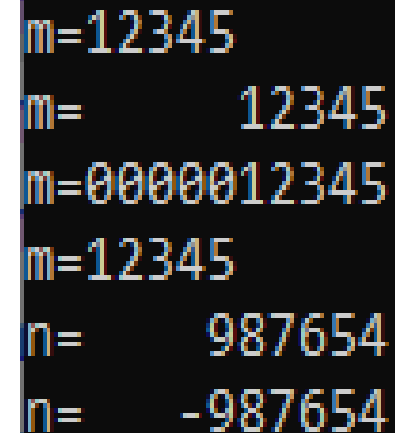
The minus(-) and zero (0) are known as **flags**.

- **minus(-)**: force the printing to be left-justified
- **zero (0)**: to pad with zeros the leading blanks

Output of Integer Numbers

```
#include<stdio.h>
void main()
{
    int m=12345;
    long n=987654;

    printf("m=%d \n",m);
    printf("m=%10d \n",m);
    printf("m=%010d \n",m);
    printf("m=%-10d \n",m);
    printf("n=%10ld \n",n);
    printf("n=%10ld \n",-n);
}
```

A screenshot of a terminal window showing the output of the C program. The output is as follows:

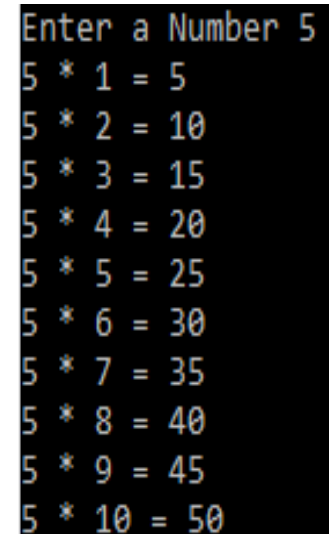
```
m=12345
m=      12345
m=0000012345
m=12345
n=      987654
n=     -987654
```

Write a C Program to Print multiplication table of number entered by user.(Use formatted input/output, do not use for loop)

```
#include <stdio.h>
void main()
{
    int n, i=1;

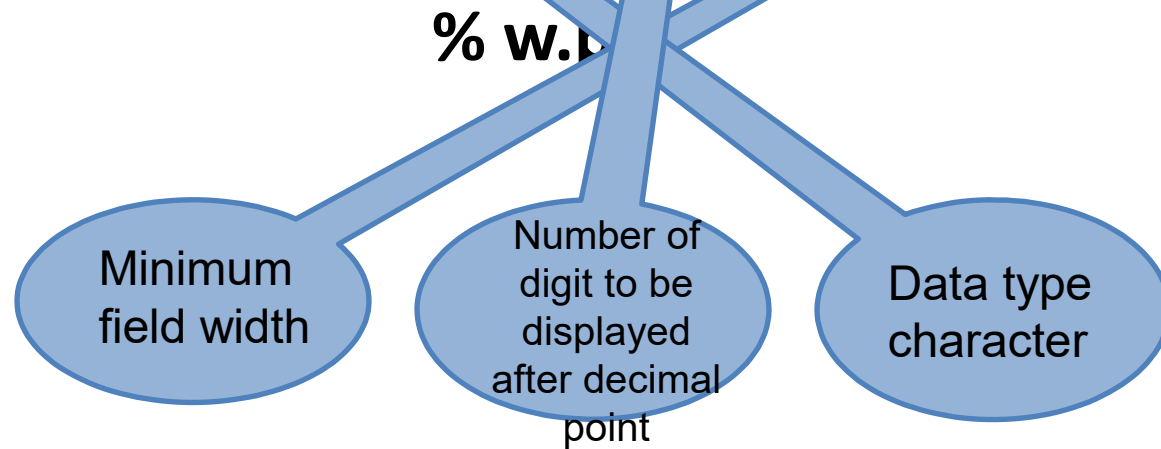
    printf("Enter a Number ");
    scanf("%d", &n);

L:
    if(i<=10)
    {
        printf("%d * %d = %d \n", n, i, n*i);
        i++;
        goto L;
    }
}
```



```
Enter a Number 5
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

The format specification for printing Real number:



- Number will be **rounded off** and printed **right justified** in given field width **w**
- The default precision is **6 decimal places**

The format specification for printing Real number in **exponential notation**:

% w.p e

The display takes the form

m.nnnne[±]xx

The field width should satisfy the condition

$w \geq p + 7$

For y=98.7654

Format

```
printf("%7.4f", y);
```

```
printf("%7.2f", y);
```

```
printf("%-7.2f", y);
```

```
printf("%f", y);
```

```
printf("%10.2e", y);
```

```
printf("%11.4e", -y);
```

```
printf("%-10.2e", y);
```

```
printf("%e", y);
```

Output

9	8	.	7	6	5	4
---	---	---	---	---	---	---

		9	8	.	7	7
--	--	---	---	---	---	---

9	8	.	7	7		
---	---	---	---	---	--	--

9	8	.	7	6	5	4
---	---	---	---	---	---	---

		9	.	8	8	e	+	0	1
--	--	---	---	---	---	---	---	---	---

-	9	.	8	7	6	5	e	+	0	1
---	---	---	---	---	---	---	---	---	---	---

9	.	8	8	e	+	0	1		
---	---	---	---	---	---	---	---	--	--

9	.	8	7	6	5	4	0	e	+	0	1
---	---	---	---	---	---	---	---	---	---	---	---

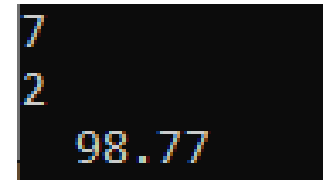
User can define the field size at **run time**:

```
printf("%*.*f", width, precision, number);
```

The advantage of this format is that the values for width and precision are supplied at run time, thus making format a **dynamic one**.

Example:

```
int w,p;  
float number=98.7654;  
  
scanf("%d %d",&w,&p);  
printf("%*.*f",w,p,number);
```



```
7  
2  
98.77
```

Output of Real Numbers

```
#include<stdio.h>
void main()
{
    float y=98.7654;
    printf("%7.4f\n", y);
    printf("%f\n", y);
    printf("%7.2f\n", y);
    printf("%-7.2f\n", y);
    printf("%07.2f\n", y);
    printf("%*.*f\n", 7, 2, y);
    printf("\n");
    printf("%10.2e\n", y);
    printf("%12.4e\n", -y);
    printf("%-10.2e\n", y);
    printf("%e\n", y);
}
```

```
98.7654
98.765404
  98.77
98.77
0098.77
  98.77

  9.88e+001
-9.8765e+001
9.88e+001
9.876540e+001
```


A single character can be displayed in a desired position using

% w c

- The character will be displayed **right-justified**
- By placing a **minus sign** we can make display **left-justified**
- The default value for **w** is **1**

The format specification for outputting strings:

%w



- The display is **right-justified**

String : NEW DELHI 110001

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

%s

N	E	W		D	E	L	H	I		1	1	0	0	0	1				
---	---	---	--	---	---	---	---	---	--	---	---	---	---	---	---	--	--	--	--

%20s

				N	E	W		D	E	L	H	I		1	1	0	0	0	1
--	--	--	--	---	---	---	--	---	---	---	---	---	--	---	---	---	---	---	---

%20.10s

										N	E	W		D	E	L	H	I	
--	--	--	--	--	--	--	--	--	--	---	---	---	--	---	---	---	---	---	--

%.5s

N	E	W		D	E	L	H	I											
---	---	---	--	---	---	---	---	---	--	--	--	--	--	--	--	--	--	--	--

N	E	W		D	E	L	H	I		1	1	0	0	0	1				
---	---	---	--	---	---	---	---	---	--	---	---	---	---	---	---	--	--	--	--

Printing of Strings

```
#include<stdio.h>
void main()
{
    char a='A';
    char name[20]="ANIL KUMAR GUPTA";

    printf("Output of the characters\n");
    printf("%c\n%3c\n%5c\n", a, a, a);
    printf("%3c\n%c\n\n", a, a);

    printf("Output of the strings\n");
    printf("%s\n", name);
    printf("%20s\n", name);
    printf("%20.10s\n", name);
    printf("%.5s\n", name);
    printf("%-20.10s\n", name);
    printf("%5s\n", name);
}
```

Output of the characters

A

A

A

A

A

Output of the strings

ANIL KUMAR GUPTA

ANIL KUMAR GUPTA

ANIL KUMAR

ANIL

ANIL KUMAR

ANIL KUMAR GUPTA

It is permitted to mix data types in one printf statement

Example:

```
printf("%d %f %s %c", a, b, c, d);
```

- The format specifications should match the variables in number, order and type.

Mixed Data Output

Code	Meaning
%c	print a single character
%d	print a decimal integer
%e	print a floating point value in exponent form
%f	print a floating point value without exponent
%g	print a floating point value either e-type or f-type
%i	print a signed decimal integer
%o	print an octal integer without leading zero
%s	print a string
%u	print an unsigned decimal integer
%x	print a hexadecimal integer, without leading 0x

Following letters may be used as prefix

h for short integers

l for long integers or double

L for long double

Write a program to convert the decimal number into octal and hexadecimal format. Hint: %o and %x

```
#include<stdio.h>

void main()
{
    int n;
    printf("Enter a number(Decimal):");
    scanf("%d",&n);

    printf("Decimal value is: %d\n",n);
    printf("Octal value is: %o\n",n);
    printf("Hexadecimal value is (Alphabet in small letters): %x\n",n);
    printf("Hexadecimal value is (Alphabet in capital letters): %X\n",n);
}
```

```
Enter a number(Decimal): 10
Decimal value is: 10
Octal value is: 12
Hexadecimal value is (Alphabet in small letters): a
Hexadecimal value is (Alphabet in capital letters): A
```

Mixed Data Output

Flag	Meaning
-	Output is left-justified within the field. Remaining field will be blank.
+	+ or – will precede the signed numeric item.
0	Causes leading zeros to appear.
#(with o or x)	Causes octal and hex items to be preceded by 0 and 0x, respectively.
\$(with e, f or g)	Causes a decimal point to be present in all floating point numbers, even if it is whole number. Also prevents the truncation of trailing zeros in g-type conversion.

End of Unit-04