Chapter 7

DATA INPUT AND OUTPUT

LEARNING OBJECTIVES

After reading this chapter, the readers will be able to

- *understand input and output concepts as they apply to C programs.*
- *use different input and output functions available in the C library.*
- *understand formatted input & output using prinf() & scanf() functions.*

7.1 INTRODUCTION

One of the essential operations performed in a C language program 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; or initialize variables in the type declaration statement like

```
  int a= 10;     float b=25.4;
```

Another method is to use **scanf()** function which can be used to read data from the 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.

7.2 SINGLE CHARACTER INPUT OUTPUT

The basic operation done in input/output is to read characters from the standard input device such as the keyboard and to output or writing it to the output unit usually the screen.

7.2.1 getchar() function

The getchar() function can be used to read a character from the standard input device. The scanf() function can also be used to achieve the purpose.. The getchar() function has the following form

```
variable _name = getchar();
```

where variable_name is any valid C identifier that has been declared as single character type. When this statement is encountered, the compiler waits until a key is pressed and then assigns this character as a value to getchar(). Since getchar() is used on the right-hand side of an assignment statement, the character value of getchar() is in turn assigned to the variable_name on the left-hand side.

For example:

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

will assign the character ’H’ to the single character variable ch when we press the key ’H’ on the keyboard. Since getchar() is a function, it requires a set of parentheses as shown in the above example. It accepts no arguments and returns a single character constant.

Program 7.1 Program for reading & writing a character

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

```
void main ( )
{
    char ch;
    printf (“Type one character:”) ;
    ch = getchar () ;    // gets a character from key board   and  stores it in the variable  ch.
     printf (” \nThe character you  typed  is = %c”, ch) ; //  displays value of ch on the screen.
 }
```

Output:

```
Type one character
K
The character you  typed  is =K
```

C supports many other similar functions which are given in the table below .These character functions are contained in header file **ctype.h .**Assume ch is declared as character type variable

Function	**Test**	**Description**
isalnum(ch)	Is ch an alphanumeric character?	Returns value 1 if true or 0 otherwise
isalpha(ch)	Is ch an alphabetic character?	Returns value 1 if true or 0 otherwise
isdigit(ch)	Is ch a digit?	Returns value 1 if true or 0 otherwise
islower(ch)	Is ch a lowercase letter?	Returns value 1 if true or 0 otherwise
isupper(ch)	Is ch a uppercase letter?	Returns value 1 if true or 0 otherwise
isprint(ch)	Is ch a printable character?	Returns value 1 if true or 0 otherwise
ispunc(ch)	Is ch a punctuation mark?	Returns value 1 if true or 0 otherwise
isspace(ch)	Is ch a whitespace character?	Returns value 1 if true or 0 otherwise
toupper(ch)	If the variable ch is assigned with lowercase alphabet converts it to uppercase alphabet	Converts to uppercase
tolower(ch)	If the variable ch is assigned with uppercase alphabet converts it to lowercase alphabet	Converts to lowercase

Program7.2 Program to test the type of input character

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

```
Output:
Press any key
H
The character is a letter.
 Press any key
5
The character is a digit
Press any key
&
The character is not alphanumeric
```

7.2.2 putchar() function

The putchar() function which is analogus to getchar() function can be used for writing character data one at a time to the output terminal. The general form is

putchar(variable /constant /expression);

The only one argument that is specified in the pair of parentheses should be either a single character variable (or) a single character constant (or) an integer constant (or) any expression whose result should be of single character type. No other data type is allowed.
variable is any C identifier containing a character declared as a single character type. If variable is used as an argument to putchar(), the character that is stored in that variable is displayed.
Constant is any single character constant (or) an integer constant. If a single character constant is used as an argument to the putchar(), it is directly displayed on the output terminal whereas if an integer constant is used as an argument to putchar(), the character whose ASCII value is equivalent to the specified integer constant will be displayed on the output terminal.

If an **expression** is used as an argument to the putchar() function, the result of that expression which is of single character type will be displayed directly on the output terminal. If the expression whose result is of integer type is used as an argument, its ASCII character will be displayed on the output terminal.
Example 7.1

```
(a)      char ch;       }
         ch=’y’;        }  would print character constant ’y’
        putchar(ch);    }
(b)      putchar(’f’);       //prints single character constant ’f’ on the screen
(c)      putchar(65);       // prints A whose ASCII value is 65.
(d)      putchar(65+35);  // prints d whose ASCII value is 65+35(i.e.,100)
(e)      putchar(65.2);      // cause an error as floating point type is not allowed.
```

Program7.3 Program to read and display a single character

```
#include<stdio.h>
main()
{
    char in;
     printf(”please Enter one character”);
     in = getchar ( ) ; // assign the keyboard   input value to in.
     putchar(in); // out put ‘in’ value to  screen.
}
```

```
Output:

please Enter one character

H

H
```

Program 7.4 Program to read an alphabet from terminal and displaying it after case conversion.(lower case to upper case & vice versa) .

```
#include<stdio.h>
#include<ctype.h>
main()
{
    char ch;
    printf(“Enter an alphabet \n”);
    ch=getchar();
    if (islower(ch))
        putchar(toupper(ch);
    else
        putchar(tolower(ch));
}
```

```
Output

Enter an alphabet

a

A

Enter an alphabet

Q

q
```

7.3 STRING INPUT AND OUTPUT

7.3.1 gets() function

The **gets()** function reads string from standard input 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). At the end function **gets( )** appends a null character as must be done to any string and returns.

The standard form of the gets function is

gets (str)

Here “str” is a string variable. For example,

```
char str[15];
puts(“enter any string:”);
gets(str);
```

If the string entered is “GREEN FIELDS” ,it is stored as

G	R	E	E	N		F	I	E	L	D	S	\0

The character ‘\0’ is appended automatically, indicating termination of string.

7.3.2 puts() function

puts() is a function that copies a string to the standard output device, usually the display screen. When we use **puts()**, we include the standard input/output header file (stdio.h). **puts()** also appends a newline character to the end of the string that is printed. The format string can contain escape sequences.

The standard form for the puts function is

```
puts (str)
```

Where str is a string variable.

Example7.2 puts("This is printed with the puts( ) function!");

```
Output
This is printed with the puts() function!
```

Example 7.3

```
puts("This prints on the first line. \nThis prints on the second line.");
puts("This prints on the third line.");
puts("If we used printf( ) instead of puts( ), all four lines would be on two lines!");
```

```
Output
This prints on the first line.
This prints on the second line.
This prints on the third line.
If we used printf( ) instead of puts( ), all four lines would be on two lines!
```

Program 7.5 Program to read and display a string

```
#include<stdio.h>
main()
{
  char s[80];
  printf (“Type a string less than 80 characters”);
  gets(s);
  printf(“\n the string typed is\n”);
  puts(s);
}
```

```
Output:
Type a string less than 80
characters
the string typed is
 K L UNIVERSITY
```

7.4 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”,&variable1,&variable2,….);
```

The control string contains format of the data being received. The ampersand symbol (&) before each variable name is the address operator that specifies variable name’s address. The use of & is must in **scanf()** function.

The control string also specifies the field format which includes format specifications. Each format specification must begin with % sign followed by conversion character which indicates the type of corresponding data item. and optional number specifying field. width.

The blanks, tabs and newlines will be real but are ignored. Multiple format specifiers can be contiguous, or they can be separated by white space characters. If whitespace characters are used to separate the formats, then they will be read but are ignored.

7.4.1 Commonly used format specifications

%c – read a single character
%d – read a decimal Integer
%e – read a floating point value in exponential form.
%f – read a floating point value
%i – read a decimal, hexadecimal or octal Integer
%h- read a short integer
%x – read a hexadecimal integer (Unsigned) using lower case a – f
%X – read a hexadecimal integer (Unsigned) using upper case A – F
%o – read an octal integer
%s – read a string
%u –read an unsigned decimal integer.
%[character set]-read only the characters specified with in brackets when inputting string
%[^character set]- The characters specified after ^(circumflex) are not permitted in the input string.

7.4.2 Input specifications for Integer

The general format for reading an integer number is :

%wd

Here

- percent sign (%) denotes that a specifier for conversion follows
- w 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 to be read integer mode.

Example7.4

scanf (“%3d %4d”, &sum1, &sum2);

If the input is
175 1342

175 is assigned to sum1 and 1342 to sum 2.

175 1342
sum1 sum2

If the input is
1342 175 .

134 2

The number 134 will be assigned to sum1 and 2 is assigned tosum2. Because of %3d first three digits of 1342 is taken and is assigned to sum1 and the remaining digit2 is assigned to the second variable sum2. If floating point numbers are assigned then the decimal or fractional part is skipped by the computer and scanf() skips reading further input.

Program 7.6 Program to read integer numbers using scanf()

```
#include<stdio.h>
main()
{
  int a,b,c,x,y,z;
  printf (“\n Enter three integers\n”);
  scanf(“%3d %3d %3d”,&a,&b,&c);
  printf(“ %d %d %d”,a,b,c);
}
```

Output

Suppose the input data items are entered as
12 35 45
Then the following assignments will result.
a=12 , b=35, c=45.
If the data had been entered as
212 359 458
Then the assignment would be
a=212 b=359 c=458
Now suppose that the data had been entered as
304685923
Then the assignment would be
a=304 b= 685 c=923
Finally, suppose that the data had been entered as
2385 4372 10
The resulting assignment would be
a=238 b=5 c=437

The remaining digit 2 and the number 10 would be ignored, unless they were read by a subsequent scanf() statement.

Assignment Suppression

An input data item may be skipped without assigning it to the designated variable or array by placing * after the % sign. For example,

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

If the input is
123 456 789
123 assigned to a
456 skipped (because of *)
789 assigned to b

Program7.7 Program to read integer numbers

```
#include<stdio.h>
main()
{
    int a,b,c;
    printf (“\n Enter three integers\n”);
    scanf(“%d %*d %d”,&a,&b,&c);
    printf(“ %d %d %d”,a,b,c);
}
```

```
Output:
Enter three integers
12  23  45
12   45  3050
```

In the above example, as the second format contains the input suppression character ‘*’, the second input item 23 is not assigned to any variable. Hence, 12 and 45 are assigned to a and b and c holds garbage value.

7.4.3 Input specifications for floating point constants

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

scanf (“%f ”, &variable);

or

scanf (“%e”, &variable);

For example,

scanf (“%f %f % f”, &a, &b, &c);

with the input data 321.76, 4.321, 678

321.76	4.321	678

The value 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.
- Similarly if the number input is a long double data type then the format specifier should be % Lf

Program 7.8 Program to read real numbers using scanf()

```
#include<stdio.h>
main()
{
     float x,y;
     double p,q;
     printf(“Enter values of x & y”);
     scanf(“%f %e “,&x,&y);
     printf(“\nx=%f\t y= %f\n”,x,y);
     printf(“Enter values of p & q”);
     scanf(“%lf %lf “,&p,&q);
     printf(“\np=%.12lf\t q= %e”,p,q);
}
```

```
Output:
Enter values of x & y    12.3456    17.5e-2
x=12.345600           y=0.175000
Enter values of  p & q      4.142857142857      18.5678901234567890
P=4.142857142857   q=  1.856789e+01
```

7.4.4 Input specifications for single character and strings

In section 7.2.1 we have seen that a single character can be read from the terminal using **getchar()** function .The same can be achieved using **scanf()** function also using ‘%c’ format specifier or %1s specifier The general format is %c or %1s. For example,

```
char ch;
scanf(“%c”, &ch);
```

Suppose the input data item is V, then the character V is assigned to ch.

If the control string contains multiple character formats same care must be taken to skip whitespace characters in the control string. As the whitespace character is also interpreted as a data item, to skip such whitespace characters and read the next nonwhite space character , the format %1s can be used.

Example 7.5 Consider the following program

```
#include<stdio.h>
main( )
{
     char ch1, ch2, ch3;
     scanf(“%c%c%c”, &ch1, &ch2, &ch3);
     printf(“%c   %c     %c \n”, ch1,  ch2,  ch3);
}
```

If the input data consists of

```
        p     q     r
```

then the following assignments would result:

ch1=p , ch2 =<blank space> , ch3= q

We could have written the scanf function as

scanf("%c %c %c", &ch1, &ch2, &ch3);

with blank spaces seperaing the format specifier %c or we could have used original scanf statement with the input data as consecutive characters without blanks; i.e., pqr.

A **scanf()** function with %wc or %ws can be used to input strings containing more than one character. The general format is

% wc or %ws

Where c and s represents character and string respectively and w represents the field width.

scanf () function supports the following conversion specifications for strings.

%[character set] and **%[^character set]**

%[character set] specification means that only the characters specified with in brackets are permissible in the input string. If the input string contains any other character, the string will be terminated at the first occurrence of such a character.

%[^character set] specification does exactly the reverse.. i.e. The characters specified after ^(circumflex) are not permitted in the input string .The reading of string will be terminated when one of these characters is encountered.

- The address operator need not be specified while we input strings.
- The specification %s terminates reading at the encounter of a white space character.

For example

```
        char str[25]
        scanf("%4c", str);
```

Reads characters from keyboard until the user enters a white space character. . Only first four characters of the string will be assigned to the string variable str.

1. If the user input is
   "abcd ef" ,the string variable str holds "abcd"
2. scanf("%s",str)
   Read characters form keyboard and the specification %s terminates reading at the encounter of a white space.
   If the user input is "abc def" ,the string variable str holds "abc"
3. scanf("%[a-z]",str);
   Read characters form keyboard and terminates reading at the encounter of non alphabet.
   If the user input is "abc123" , the string variable str holds "abc"
4. scanf("%[^z]",str);
   Read characters form keyboard and terminates reading at the encounter of character 'z'.
   If the user input is "abc123 z" , the string variable str holds "abc123".

Program 7.8 Program to read strings from keyboard

```
main()
{
    char name1[15], name2[15], name3[15];
    printf(“Enter name1:\n”);
    scanf(“%15c”,name1);
    printf(“Enter name2:\n”);
    scanf(“%s”,name2);
    printf(“\n %15s”,name2);
    printf(“Enter name3:\n”);
    scanf(“%15s”,name3);
    printf(“\n %15s”,name3);
}
```

```
Output:

Enter name1
K L UNIVERSITY
K L UNIVERSITY
Enter name2
GUNTUR DIST
GUNTUR
Enter name3
DIST
```

In the above example;

- string name1= K L UNIVERSITY”
- The specification %s terminates reading at the encounter of a blank space ,so name 2=”GUNTUR” only.
- Second part of name2 will be automatically assigned to name3.Hence name3wiill be printed as DIST

Program 7.9 Program to illustrate %[character set] specification

```
main()
{       char address[80];
         printf(“Enter address \n”);
         scanf(“%[a-z]”,address);
         printf(“%s\n\n “,address);
}
```

```
Output:
Enter address
Vijayawada  520001
Vijayawada
```

The reading of the string is terminated when blank space is encountered. Hence the string will be read as Vijayawada only.

Program 7.10 Program to illustrate %[^character set] specification

```
main()
{
 char address[80];
 printf(“Enter address \n”);
 scanf(“%[^\n]”,address);
 printf(“%-80s\n\n “,address);
}
```

```
Output:
Enter address
 Vijayawada  520001
 Vijayawada 520001
```

In the above example when newline(\n) is entered, reading will be terminated. Hence string variable address will be “Vijayawada 520001”

7.4.5 Reading Mixed data Types

scanf() function can be used to input a data line containing mixed mode data. Care should be taken to ensure that input data items match the control specifications in order and type

For example:

scanf(“%d%c % f%s”,&count,&code,&ratio,name);

If the input is

35 x 42.76 ABC

the character after 35 is space,so it is taken as value of character variable code and the character x is not assigned to any of the variable.

To avoid this mistake one way is giving input without space between 35 and x i.e the input is
35x 42.76 ABC
Another way is skip the space by leaving a space in control string of scanf() function. The function should be rewritten as
scanf("%d %c % f %s",&count,&code,&ratio,name);

7.5 FORMATTED OUTPUT

General format of **printf( )** is as follows :

```
printf("control string",exp1,exp2,exp3,…,expn);
```

The control string contains format of the data to be displayed and exp1,exp2,exp3…,expn are output expressions.

Control string consists of three items

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

A simple format specification has the following form:

```
% flag w.p type-specifier
```

Where

- Flag- used for print modifications - justification, padding, sign
- w - is an integer specifies total number of columns for output value
- p - is an integer specifies the number of digits to right of decimal part of real number or no of characters to be printed from string

Note: flag , w , p are optional

7.5.1 Output of Integer Numbers

Format specification for printing an integer is

```
%wd
```

where

- w-specifies minimum field width for the output .However , if a number is greater than specified width it will be printed in full
- d- specifies that value to be printed is an integer.
- The number will be right justified
- Negative numbers will be printed with – sign

For example,

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

- placing - sign after % causes the output left justified with in the field. Remaining field will be blank
- placing 0 before the filed width specifier causes leading blanks padded with zeros.

%2hd - output is short integer with 2 print positions
%4d - output is integer with 4 print positions
%8ld - output is long integer with 8 print positions(not 81)

7.5.2 Output of floating point constants

The output of floating point constants may be displayed in decimal notation using the following format specification:

% w.p f

- w indicates minimum number of positions that are to be used for the display of the value
- p indicates number of digits to be displayed after decimal point (precision).Default precision is 6 decimal columns
- The number will be right justified in the field of w columns
- Negative numbers will be printed with – sign

The output of floating point constants may also be displayed in exponential notation using the following format specification:

% w.p e

Let x=98.7654

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

7.5.3 Printing of single Character

A single character can be displayed in a desired position using following format specification

%wc

- The character will be displayed right justified in the filed of w columns
- We can make display left justified by placing - sign before the field width w
- Default value of w is one

```
Output
Enter  any character:
K
K
```

Example 7.6

K

```
main()
{
    char ch;
    printf("Enter  any character:\n");
    scanf("%c",&ch);
    printf("\n % c",ch);
    printf("\n % 5c",ch);
}
```

7.5.4 Printing of single strings

The format specification for outputting string is similar to that of real numbers. It is of the form

%w.ps

- w specifies filed width for display
- p instructs only first p characters of the string are to be displayed
- The display is right justified

The following examples show the effect of a variety of specifications in printing a string "K L UNIVERSITY"

Specification	**Output**																			
%s	K		L		U	N	I	V	E	R	S	I	T	Y						
%20s						K		L		U	N	I	V	E	R	E	S	I	T	Y
%20.5s																K		L		U
%.5s	K		L		U															
%-20.8s	K		L		U	N	I	V												
%5s	K		L		U	N	I	V	E	R	S	I	T	Y						

7.5.5 Mixed Data Output:

We can mix data types in one printf() statement. **printf ()**uses its control string to decide how many variables to be printed and what their type are. The format specifications should match variable in number ,order, type .For example

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

7.5.6 Commonly used printf() format codes

%c – print a single character

%d – print a decimal Integer

%e – print a floating point value in exponential form.

%f – print a floating point number

%g– print a floating point value using either e-type or f-type, conversion depending on value. Trailing zeros and trailing decimal points will not be displayed

%i – print a decimal, hexadecimal or octal Integer

%h– print a short integer

%u – print an unsigned integer

%x – print a hexadecimal integer (unsigned) using lower case a – f

%X –hexadecimal integer (unsigned) using upper case A – F

%o – print an octal integer without leading zero
%s – print a string
The following letters may be used as prefixes for certain conversion characters'

- h for short integers
- l for long integers or for double
- L for long double

7.5.7 Output Format Flags

Flag Type	Flag code	Meaning
Justification	none	Output is right-justified
	-	Output is left-justified
sign	None	Positive Value: no sign Negative Value: -
	+	Positive Value: + Negative Value: -
padding	None	Space padding
	0	Causes leading 0's to appear(Zero padding)
	#(with 0 or 0x)	Causes octal and Hexa decimal numbers to be preceeded by 0 or 0x,respectively
	#(with e,f,g)	Causes a decimal point to be present in all floating point numbers even for whole number. Prevents truncation of trailing 0's in g-type

Example 7.7

```
 #include<stdio.h>
 main()
 {
      int  a,b,c,d;
      float x,y,z;
      printf ("\n Enter three integers\n");
      scanf("%d%d%d",&a,&b,&c);
      printf ("\n Enter three floating numbers:\n");
      scanf("%f %f %f",&x,&y,&z);
      printf(" \n%5d\t %5d \t%5d",a,b,c);
      printf(" \n%05d\t %05d\t%05d",a,b,c);
      printf(" \n%-5.2f\t %-5.2f \t%-5.2f",a,b,c);

   }
```

```
Output
Enter three integers
12  34  45
 Enter three floating numbers
23.456     89.1234  12.456
      12          34         45
 00012     00034    00045
 23.46      89.12     12.46
```

Example 7.8

```
#include<stdio.h>
main()
{
     int  a,b,;
     printf (“\n Enter two integers\n”);
     scanf(“%d %d”,&a,&b);
     printf(“ \n%5d\t %5d”,a,b);
     printf(“ \n%+5d\t %+5d”,a,b);
     printf(“ \n%05d\t %05d”,a,b);
}
```

```
Output
          Enter  two integers
          12    -23
             12          -23
           +12          -23
          00012        00-23
```

Example 7.9

```
#include<stdio.h>
main()
{
int  a,b,c,d;
printf (“\n Enter two octal integers\n”);
scanf(“%o %o”,&a,&b);
printf (“\n Enter two hexadecimal integers\n”);
scanf(“%x %x”,&c,&d);
printf(“ \n%o\t %o”,a,b);
printf(“ \n%x\t %x”,c,d);
printf(“ \n%#o\t %#o”,a,b);
printf(“ \n%#x\t %#x”,c,d);
}
```

```
Output
    Enter  two octal  integers
    12    45
    Enter  two hexadecimal integers
    94     2d
    12    45
    94     2d
    012    045
    0x94     0x2d
```

Suggested Reading:

1. Chapter-9 : C for Engineers and Scientists by Harry H.Cheng.
2. Chapters-4: Programming with C by Byron S.Gottfried

EXERCISES

Review Questions

7.1.What will the values of each variable be after the input command:

data input: Tom 34678.2AA4231

scanf("%s %3d %f %c %*c %1d %x",name,&m,&x,&ch,&i,&j);

name:Tom   m :346   x :78.2
ch :A   i :4   j :231

7.2. What output does each of these produce?

a)putchar('a');
b)putchar('\007');
c) putchar('\n');
d) putchar('\t');
e) n = 32; putchar(n);
f) putchar('\"');

7.3. For the different values of n, what is the output?

printf("%x %c %o %d",n,n,n,n);

a) n = 67   b) n = 20
c) n = 128   d) n = 255

7.4. What is wrong with each of the following statements?

a) scanf("%d",i);
b) #include stdio.h
c) putchar('\n');
d) puts("\tHello");
e) printf("\nPhone Number: (%s) %s",phone_number);
f) getch(ch);
g) putch() = ch;
h) printf("\nEnter your name:",name);

7.5. Which numbering system is not handled directly by the **printf()** conversion specifiers?

a) decimal	b) binary
c) octal	d) hexadecimal

7.6. Which one of the following conversion specifiers cannot be used for a number represented in binary form in the computer?

a) %b b) %d c) %o d) %x

Comprehensive questions

7.1. Write a small program that will prompt for the input of a value for each of the following types:
%c - Yes or No
%s - Your Name
%f - Your Height
%ld - The Circumference of the Earth
%f - Your Bank Balance
%lf - The Distance from the Earth to the Moon
Read the values with **scanf()** and then print those values out on the display using **printf()**.

7.2. Write a program that will prompt for the input of a temperature in Fahrenheit and will display as output both the Fahrenheit value and the temperature converted to Celsius. Use the formula
Celsius Degrees = (Fahrenheit Degrees - 32) * 5/9

7.3. Write a program that uses **scanf()** and **printf()** statements to prompt for your first name, last name, street, city, state and zip code. After input of the values, then print the values out with the following format:
Name:
Street:
City:
State:
Zip:

7.4. Write a program that converts and prints a user supplied measurement in inches into

a.foot(12 inches)	b. yard (36 inches)
c.centimeter (2.54/inch)	d. meter (39.37 inches)

7.5. Write a program to accept three integers and to display them
- with %d format specifier
- with column width 5
- with sign (+ or -) as prefix
- with column width 5 and left justified
- with column width 5 & Padding 0's

7.6. Write a program to accept octal & hexadecimal integers and to display them
- Using %o ,%x format specifiers
- with column width 5
- padding 0 for octal & 0x for hexadecimal

- with column width 5 and left justified
- with column width 5 & Padding 0's

7.7. Let a=6.789654,b=1.3e+02,write a 'C' program to display the values of a &b in the following format

- Display a & b values in floating point notation with precision of 3digits
- Display a & b values in exponential notation with left justified

For exercise 8 & 9 The variables count, price city declared as – int ,float, char [] data type and have the values. Count=1275, Price=235.74, City=Guntur

7.8. Show the exact output that the following statements will produce.

a. printf("%d %f ",count,price);
b. printf("%2d \n%f ",count,price);
c. printf("%d %f ", price,count);
d. printf("%10dxxxx%5.2f ", count price);
e. printf("%s", city);
f. printf("%-10d%-15s",count, city);

7.9. State what (if anything) is wrong with each of the following output statements.

a. printf("%d 7.2%f,year ,amount);
b. printf("%-s,%c"\n,city,code);
c. printf("%f, %d, %s,price,city,code);
d. printf("%c%d%s\n",amout,code,year);