**PSG College of Technology, Coimbatore- 4**

**Department of Applied Mathematics and Computational Sciences**

**I Sem MSc DS- Problem Solving and C Programming**

**Problem Sheet –IV**

**scanf ( ) and printf ( ) Functions**

**printf()**

printf is a predefined function in "stdio.h" header file, by using this functionone can print the data or user defined message on console or monitor. While working with printf(), it can take any number of arguments but first argument must be within the double quotes (" ") and every argument should separated with comma ( , ) Within the double quotes, whatever we pass, it prints same, if any format specifies are there, then that copy the type of value.

**Syntax**

printf("user defined message");

prinf("Format specifers",arg1, arg2,.., argn);

Where Format specifiers (or) control string refers to a string that contain formatting information, and arg1, arg2, …,argn are the arguments that represent the individual output data items.

The arguments can be constants, single variable or array names, or more complex expressions.

| **Format specifier** | **Description** | **Supported data types** |
| --- | --- | --- |
| %c | Character | char unsigned char |
| %d | Signed Integer | short unsigned short int long |
| %e or %E | Scientific notation of float values | float double |
| %f | Floating point | float |
| %g or %G | Similar as %e or %E | float double |
| %hi | Signed Integer(Short) | short |
| %hu | Unsigned Integer(Short) | unsigned short |
| %i | Signed Integer | short unsigned short int long |
| %l or %ld or %li | Signed Integer | long |
| %lf | Floating point | double |
| %Lf | Floating point | long double |
| %lu | Unsigned integer | unsigned int unsigned long |
| %lli, %lld | Signed Integer | long long |
| %llu | Unsigned Integer | unsigned long long |
| %o | Octal representation of Integer. | short unsigned short int unsigned int long |
| %p | Address of pointer to void void \* | void \* |
| %s | String | char \* |
| %u | Unsigned Integer | unsigned int unsigned long |
| %x or %X | Hexadecimal representation of Unsigned Integer | short unsigned short int unsigned int long |
| %n | Prints nothing |  |
| %% | Prints % character |  |

* A minimum field width can be specified by preceding the conversion character by an unsigned integer.
  + If the number of characters in the corresponding data item is less than the specified field width, then the data item will be preceded by enough **leading blanks** to fill the specified filed.
  + If the number of characters in the corresponding data item exceeds than the specified field width, additional space will be allocated to the data item, so that the entire data item will be displayed.

Example 1:

#include <stdio.h>

intmain()

{

  inti=12345;

flaot x= 345.678;

     printf("%3d %5d %8d\n", i, i,i);

printf("%3f %10f %13f\n", x,x,x);

printf("%3e %13e %16e\n", x,x,x);

    return0;

}

Output:

12345 12345 12345

345.678000 345.678000 345.678000

3. 456780e+02 3. 456780e+02 3. 456780e+02

Example 2:

#include <stdio.h>

intmain()

{

  inti=12345;

flaot x= 345.678;

     printf("%3d %5d %8d\n", i, i,i);

printf("%3g %10g %13g\n", x,x,x);

printf("%3g %13g %16g\n", x,x,x);

    return0;

}

Output:

12345 12345 12345

345.678 345.678 345.678

345.678 345.678 345.678

Note: The minimum field width conform to the specifications within control string

Example 3:

#include <stdio.h>

intmain()

{

  flaot x= 123.456;

   printf("%7f %7.3g %7.1g\n", x, x, x);

printf("%12e %12.5e %12.3e\n", x, x, x);

    return0;

}

Output:

123.456000 123.456 123.5

1.234560e+02 1.23456e+02 1.235e+02

Note:

* The first line is produced by f-type conversion. Notice the rounding that occurs in the third number because of the precision specification (one decimal place). Also, notice the leading blank space that are added to fill the specified minimum field width (seven characters)
* The first line is produced by e-type conversion, has similar characteristics. Again, we see that the third number is rounded to conform to the specified precision (three decimal places) . Also, note the leading blanks that are added to fill the specified minimum field width (12 characters).

It is possible to specify the precision without the minimum field width.

Example:

#include <stdio.h>

intmain()

{

  flaot x= 123.456;

   printf("%f %.3g %.1g\n", x, x, x);

printf("%e %.5e %.3e\n", x, x, x);

    return0;

}

Output:

134.456000 123.346 123.5

1.234560e+02 1.234560e+02 1.235e+02

The following program outlines the use of minimum field width and precision specifications in computation with strings.

#include <stdio.h>

intmain()

{

char line[12] = “hexadecimal”;

printf(“ %10s %15s %15.5s %.5s”, line, line, line, line);

}

Output:

hexadecimal hexadecimal hexadhexad

Note:

* The first string is shown entirely even though this string consists of 11 characters but the field width specification is only 10 characters.
* The second string padded with four leading blank spaces to fill out the 15- character minimum. Hence the second string is right justified within its field.
* The third string consists of only five nonblank characters because of 5 characters precision specification. However, 10 leading blanks are added to fill out the minimum field with specification, which are 15 characters.
* The last string also consists of 5 nonbalnk characters. Leading blanks are not added because there is no minimum field with specification.

The following program illustrates the use of uppercase conversion characters in the printf function.

#include <stdio.h>

intmain()

{

int a=0x80ec;

float b= 0.3e-12;

printf("%4x %10.2e \n", a, b);

printf("%4x %10.2E", a, b);

}

Output:

80ec 3.00e -13

80ec 3.00E-13

In addition to the field width, the precision and the conversion character, each character group within the control string can include a *flag*, which affects the appearance of the output. The falg must be placed immediately after the percent sign %.

The more commonly used flags are listed below.

|  |  |
| --- | --- |
| **Flag** | **Meaning** |
| - | Data item is left justified within the field (blank spaces are required to fill minimum field width will be added *after* the data item rather than *before* the data item |
| + | A sign (either + or - ) will precede each signed numerical data item. Without this flag , only negative data items are preceded by a sign |
| 0 | Causes leading zeros to appear instead of leading blanks. Applies only to data items that are right justified within a field whose minimum size is larger than the data item |
| “ | (blank space) A blank space will precede each positive signed numerical data item. This flag is overridden by the + flag if both are present. |
| # | (with o- and x-type conversion) Causes octal and hexadecimal data items to be preceded by 0 and 0x, respectively. |
| # | (with e-, f- and g-type conversion) Causes decimal point to be present in all floating-point numbers, even if the data item is a whole number. Also, prevents the truncation of trailing zeros in g-type conversion |

**Example:**

#include <stdio.h>

intmain()

{

inti=123;

float x= 12.0, y=-3.3;

printf(":%6d %7.0f %10.1e : \n",i , x, y);

printf(":%-6d %-7.0f %-10.1e : \n",i ,x, y);

printf(":%+6d %+7.0f %+10.1e : \n",i ,x, y);

printf(":%-+6d %-+7.0f %-+10.1e : \n",i ,x, y);

printf(":%7.0f %#7.0f %7g %#7g : \n",x ,x, y, y);

}

**Output:**

: 123 12 -3.3e+00:

:123 12 -3.3e+0 :

: +123 +12 -3.3e+0:

: +123 +12 -3.3e+0 :

: 12 12. -3.3 -3.30000:

**Execute the following codes segments.**

1. #include <stdio.h>

intmain()

{

inti=1234, j= 01777, k=0xa08c;

printf(":%8u %8o %8x : \n",i , j, k);

printf(":%-8u %-8o %-8x : \n",i , j, k);

printf(":%#8u %#8o %#8x : \n",i , j, k);

printf(":%08u %08o %08x : \n",i , j, k);

}

1. #include <stdio.h>

intmain()

{

char line[12] = “lower-case”;

printf(“ %15s %15.5s %.5s ”, line, line, line);

printf(“ %-15s %-15.5s %-.5s ”, line, line, line);

}

1. #include <stdio.h>

intmain()

{

float a=2.2, b=-6.3, x1=.005, x2=-12.88;

printf(“ $%4.2f %7.1f%% \n ”, a,b);

printf(“ x1=%7.3f x2=7.3f \n ”, x1,x2);

}

**scanf()**

scanf() is a predefined function in "stdio.h" header file. It can be used to read the input value from the keyword.

**Syntax**

scanf("Format specifiers",&value1,&value2,.....);

1. **Search Set in scanf statement : This method does not work with integer or float type. Using [set], you can only take character type input.**

#include <stdio.h>

int main()

{

char line[80];

scanf(“%[ ABCDEFGHIJKLMNOPQRSTUVWXYZ]”,line);

printf(“%s”, line);

}

1. #include <stdio.h>

int main()

{

char line[80];

scanf(“%[^\n]”,line);

printf(“%s”, line);

}

%[^\n] would take all characters in a single line as input. You may know that, whenever we press Enter in keyboard, then “\n” character is written in the console. Here, “^” means up-to and “\n” means new line and finally, [^\n] means take input up-to new line.

* Similarly, %[^5] would take all characters as input up-to 5. If you write “abcdefghk5k zaq” in the console, then just “abcdefghk” would be taken as input.
* %[0–9] would take just number characters as input. If you write “1234abc567” in the console, then just first numbers “1234” would be taken as input.
* Similarly, %[a–z] would take just small letter alphabets as input. If you write “abc12ijkl” in the console, then just first small letters “abc” would be taken as input.
* %[123] would just take any combination of 1, 2, 3 as input. Anything other than 1,2,3 would be ignored. If you write “1123390123” in the console then just first “11233” would be taken as input.

1. Sometimes, we need to limit the number of digit in integer or float, number of character in string. We can achieve this by adding an integer(>0) in the string format.

int a;

scanf("%5d", &a);

The code above, would take just first 5 digits as input. Any digits after 5th one would be ignored. If the integer has less than 5 digits, then it would be taken as it is.

%5d would take an integer of at most 5 digits.

%5f would take a float type number of at most 5 digits or 4 digits and one “ . ” as input.

%5s would take a string of at most 5 non white-space characters as input.

#include <stdio.h>

intmain()

{inta,b,c;

scanf(“%3d %3d %3d”, &a, &b, &c);

printf(“%d %d %d”, a, b, c);

}

Input the following input instances.

1 2 3

123 456 789

12345678

1234 5678 9

**Output:**

When the program is executed, three integer quantities will be entered from the standard input device. Suppose the input data items are entered as

**1 2 3**

Then the following assignments will result:

**a=1, b=2, c=3**

If the data had been entered as:

**123 456 789**

Then the assignments would be:

**a=123, b= 456, c=789**

**Now suppose the data had been entered as**

**123456789**

**Then the assignments would be:**

**a=123, b=456,c=789**

**Since the first three digits would be assigned to a, the next three digits to b, and the last three digits to c.**

**Finally, suppose that the data had been entered as**

**1234 5678 9**

**Then the assignments would be:**

a=123, b=4, c=567

**The remaining two digits (8 & 9) would be ignored.**

1. #include <stdio.h>

intmain()

{

inti ;

float x;

char c;

scanf (“%3d %5f %c”, &i, &x, &c);

printf(“%d %f %c”, i, x, c);

}

Input the following values:

10 256.875 T

1. #include <stdio.h>

intmain()

{

short x, iy;

long lx, ly;

double dx, dy;

scanf (“%hd %ld %lf”, &ix, &lx, &dx);

printf(“%hd %ld %lf”, ix, lx, dx);

scanf(%3ho %7lx %15le”, &iy, &ly, &dy);

printf(%3ho %7lx %15le”, iy, ly, dy);

}

**Note:**The **second scanf** function indicates that the **first data item** will have a maximum field **width of 3** characters ad it will be assigned to **short octal integer** variable, the **second data item** will have a maximum field **width of 7** characters and it will be assigned to a **long hexadecimal integer variable**,and the **third data item** will have a maximum field **width of 15** characters and it will be assigned to a **double-precision variable**.

1. #include <stdio.h>

intmain()

{

short x, iy;

long lx, ly;

double dx, dy;

scanf (“%hd %**D**  %f”, &ix, &lx, &dx);

printf(“%hd %**D**%f”, ix, lx, dx);

scanf(%3ho %7**X** %15e”, &iy, &ly, &dy);

printf(%3ho %7**X** %15e”, iy, ly, dy);

}

**Note:** the use of uppercase conversion characters (in scanf functions) to indicate long integers. The interpretation of the scanffn will be the same as the previous example.

1. It is possible to skip over a data item, without assigning to the designated variable. To do so, use \* in between % and format specifier like [**%\*d]. This feature is referred to as assignment suppression.**

#include <stdio.h>

int main()

{

char item[20];

intpartno;

float cost;

scanf(“%s %\*d %f”, item, &partno, &cost);

printf(%s%\*d%f”, item, partno,cost);

}

**Note: the asterisk in the second character group.**

**The corresponding data items are**

**Fastener 12345 0.05**

Faster will be assigned to item and 0.05 will be assigned to cost. However 12345 will not be assigned to partno because of asterisk, which is interpreted as an assignment suppression character.

* Sometimes, you may need to exclude some character or number from user input. Suppose, the input is **30/01/2018**and you want to get day, month, year separately as integer. You can achieve this, using **“ \* ”**in string format.

int day,month,year;  
 scanf("%d%\*c%d%\*c%d", &day, &month, &year);

Here, %c is character type specifier and %\*c means one character would be read from the console but it wouldn’t be assigned to any variable. Here we excluded two “/” from the input.

* + **%\*c** would exclude one character. **Remember,** **‘\n**’ and **‘\t’** are single characters.
  + **%\*d** would exclude one integer.
  + **%\*f** would exclude one float.
  + **%\*s** would exclude one word.

1. #include <stdio.h>

int main()

{

char c1, c2, c3;

scanf(“%c%c%c”, &c1, &c2, &c3);

printf((“%c%c%c”, c1, c2, c3);

}

If your input is

a b c

(with blank space between the letters), then the following assignments would result:

c1=a , c2=blank space, c3=b

if the scanf function is written as scanf(“%c%1s%1s”, &c1, &c2, &c3);

then the same input data would result in the following assignments:

c1=a , c2=b, c3=c;

Note: There are some other ways to solve this problem. We could have return the scanf function as scanf(“%c %c %c”, &c1, &c2, &c3);

With blank spaces separating the %c terms, or we could have used the original scanf function but written the input data as consecutive characters without balnks. Ie.,abc.

1. Unrecognized characters within control string are expected to be matched by the same characters in the input data.

Example:

int main()

{inti ;

float x;

scanf(“%d a %f “, &i, &x);

printf((“%d %f “, i, x);

}

Note : If the input data consist of

10 a 20.5

Then the decimal integer 10 will be read in and assigned to I, the character a will be read in but subsequently ignored, and floating-point value 20.5 will be read in and assigned to x.

On the other hand, if the input were entered simply as

10 20.5

Then the scanf function would stop executing once the expected character is not found. Therefore, I would be assigned the value 10 but x would automatically represent the value 0.

**Execute the following code segments.**

**Character format specifier : %c**

|  |
| --- |
| #include <stdio.h>  intmain()  {      charch = 'A';      printf("%c\n", ch);      return0;  } |

**Output:**

A

**Integer format specifier : %d, %i**

|  |
| --- |
| #include <stdio.h>  intmain()  {      intx = 45, y = 90;      printf("%d\n", x);      printf("%i\n", x);      return0;  } |

**Output:**

45

45

**Double format specifier : %f, %e or %E**

|  |
| --- |
| #include <stdio.h>  intmain()  {      floata = 12.67;      printf("%f\n", a);      printf("%e\n", a);      return0;  } |

**Output:**

12.670000

1.267000e+01

**Unsigned Octal number for integer : %o**

|  |
| --- |
| #include <stdio.h>  intmain()  {      inta = 67;      printf("%o\n", a);      return0;  } |

**Output:**

103

**Unsigned Hexadecimal for integer : %x, %X**

|  |
| --- |
| #include <stdio.h>  intmain()  {      inta = 15;      printf("%x\n", a);      return0;  } |

**Output:**

f

**String printing : %s**

|  |
| --- |
| #include <stdio.h>  intmain()  {      chara[] = "geeksforgeeks";      printf("%s\n", a);      return0;  } |

**Output:**

Geeksforgeeks

|  |
| --- |
| #include <stdio.h>  intmain()  {      charstr[] = "geeksforgeeks";      printf("%20s\n", str);      printf("%-20s\n", str);      printf("%20.5s\n", str);      printf("%-20.5s\n", str);      return0;  } |

**Output:**

geeksforgeeks

geeksforgeeks

geeks

geeks

**scanf(char \*format, arg1, arg2, …)**

This function take input using standard input (keyboard) and store it in variable accordingly. It returns the number of items successfully read. Formal parameter arg1, agr2, ..must be a pointer

**decimal integer : %d**

|  |
| --- |
| #include <stdio.h>  intmain()  {      inta = 0;      scanf("%d", &a); // input is 45      printf("%d\n", a);      return0;  } |

**Integer may be octal or in hexadecimal : %i**

|  |
| --- |
| #include <stdio.h>  intmain()  {      inta = 0;      scanf("%i", &a); // input is 017 (octal of 15 )      printf("%d\n", a);      scanf("%i", &a); // input is 0xf (hexadecimal of 15 )      printf("%d\n", a);      return0;  } |

**Floating data type : %f, %e(double), %lf(long double)**

|  |
| --- |
| #include <stdio.h>  intmain()  {      floata = 0.0;      scanf("%f", &a); // input is 45.65      printf("%f\n", a);      return0;  } |

**Output:**

0.000000

**String input : %s**

|  |
| --- |
| #include <stdio.h>  intmain()  {      charstr[20];      scanf("%s", str); // input is geeksforgeeks      printf("%s\n", str);      return0;  } |

**Output:**

P@

**Character input : %c**

|  |
| --- |
| #include <stdio.h>  intmain()  {      charch;      scanf("%c", &ch); // input is A      printf("%c\n", ch);      return0;  } |

Many other format specifier are also there  
1.%u for unsigned integer.  
2.%lld for long longint .  
3.%o octal integer without leading zero  
4.%x hexadecimal integer without 0x before number.

Controlling integer width with printf

The %3dspecifier is used with integers, and means a minimum width of three spaces, which, by default, will be right-justified:

|  |  |
| --- | --- |
| printf("%3d", 0); | 0 |
| printf("%3d", 123456789); | 123456789 |
| printf("%3d", -10); | -10 |
| printf("%3d", -123456789); | -123456789 |

## Left-justifying printf integer output

To left-justify integer output with printf, just add a minus sign (-) after the % symbol, like this:

|  |  |
| --- | --- |
| printf("%-3d", 0); | 0 |
| printf("%-3d", 123456789); | 123456789 |
| printf("%-3d", -10); | -10 |
| printf("%-3d", -123456789); | -123456789 |

## The printf integer zero-fill option

To zero-fill your printf integer output, just add a zero (0) after the % symbol, like this:

|  |  |
| --- | --- |
| printf("%03d", 0); | 000 |
| printf("%03d", 1); | 001 |
| printf("%03d", 123456789); | 123456789 |
| printf("%03d", -10); | -10 |
| printf("%03d", -123456789); | -123456789 |

## printf integer formatting

As a summary of printf integer formatting, here’s a little collection of integer formatting examples. Several different options are shown, including a minimum width specification, left-justified, zero-filled, and also a plus sign for positive numbers.

|  |  |  |
| --- | --- | --- |
| **Description** | **Code** | **Result** |
| At least five wide | printf("'%5d'", 10); | '   10' |
| At least five-wide, left-justified | printf("'%-5d'", 10); | '10   ' |
| At least five-wide, zero-filled | printf("'%05d'", 10); | '00010' |
| At least five-wide, with a plus sign | printf("'%+5d'", 10); | '  +10' |
| Five-wide, plus sign, left-justified | printf("'%-+5d'", 10); | '+10  ' |

## formatting floating point numbers with printf

Here are several examples showing how to format floating-point numbers with printf:

|  |  |  |
| --- | --- | --- |
| **Description** | **Code** | **Result** |
| Print one position after the decimal | printf("'%.1f'", 10.3456); | '10.3' |
| Two positions after the decimal | printf("'%.2f'", 10.3456); | '10.35' |
| Eight-wide, two positions after the decimal | printf("'%8.2f'", 10.3456); | '   10.35' |
| Eight-wide, four positions after the decimal | printf("'%8.4f'", 10.3456); | ' 10.3456' |
| Eight-wide, two positions after the decimal, zero-filled | printf("'%08.2f'", 10.3456); | '00010.35' |
| Eight-wide, two positions after the decimal, left-justified | printf("'%-8.2f'", 10.3456); | '10.35   ' |
| Printing a much larger number with that same format | printf("'%-8.2f'", 101234567.3456); | '101234567.35' |

## printf string formatting

Here are several examples that show how to format string output with printf:

|  |  |  |
| --- | --- | --- |
| **Description** | **Code** | **Result** |
| A simple string | printf("'%s'", "Hello"); | 'Hello' |
| A string with a minimum length | printf("'%10s'", "Hello"); | '     Hello' |
| Minimum length, left-justified | printf("'%-10s'", "Hello"); | 'Hello     ' |

**QUIZ Questions**

**1.What would be the output of following program?**

#include<stdio.h>

int main()

{

unsigned int x=234;

long unsigned int y=234L;

printf("%lu\n%u",x,y);

return 0;

}

**2. What would be the output of following program?**

#include<stdio.h>

int main()

{

long float x=a=456.45645656;

unsigned double y=45;

printf("%lf\n%ulf",x,y);

return 0;

}

**3. What would be the output of following program?**

#include<stdio.h>

int main()

{

inta,b;

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

return 0;

}

**4. What would be the output of following program?**

#include<stdio.h>

int main()

{

printf("%d\n%c");

return 0;

}

**5.What would be the output of following program?**

#include<stdio.h>

int main()

{

int a=10,b=20;

printf("%d",a,b);

return 0;

}

**6. What would be the output of following program?**

#include<stdio.h>

int main()

{

printf("%f");

return 0;

}

**7.What would be the output of following program?**

#include<stdio.h>

int main()

{

longnum;

num=2;

printf("%ld",num);

return 0;

}

**8. What would be the output of following program?**

#include<stdio.h>

int main()

{

int x=300;

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

return 0;

}

**9. What would the output of following program?**

#include<stdio.h>

int main()

{

inti=97;

printf("%c",i);

return 0;

}

**10. What would be the output of following program?**

#include<stdio.h>

int main()

{

float f=65.34;

printf("%c",(char)f);

return 0;

}

**11. What would be the output of following program ?**

#include<stdio.h>

int main()

{

printf("%d",sizeof('a'));

printf("\n%d",sizeof(7));

printf("\n%d",sizeof(0.7));

return 0;

}

**12. What would be the output of following program?**

#include<stdio.h>

int main()

{

double d=(double)'a';

printf("%lf",d);

return 0;

}

**13. What would be the output of following program?**

#include<stdio.h>

int main()

{

float f=45.78;

inti=(int)f;

printf("%d",i);

printf("\n%f",f);

return 0;

}

**14. What would be the output of following program?**

#include<stdio.h>

int main()

{

float f=(char)87.56;

printf("%f",f);

return 0;

}

**15.What would be the output of following program?**

#include<stdio.h>

int main()

{

inti=3;

float j=4;

printf("%d",i/j);

return 0;

}

**16. What would be the output of following program?**

#include<stdio.h>

int main()

{

charch='c';

printf("%f",(float)((int)ch));

return 0;

}

**17. What would be the output of following program?**

#include<stdio.h>

int main()

{

int p=2.999998f;

printf("%f",p);

return 0;

}

**18. What would be the output of following program?**

#include<stdio.h>

int main()

{

int a=4.5678;

int b=5.98756;

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

return 0;

}

**19. What would be the output of following program?**

#include<stdio.h>

int main()

{

float a;

a=4/2;

printf("%f\n%f",a,4/2);

return 0;

}

**20. What would be the output of following program?**

#include<stdio.h>

int main()

{

float a=75.656785;

printf("%.3f",a);

printf("\n%.0f",a);

printf("\n%6.2f",a);

printf("\n%6.2f",345.98756);

return 0;

}