* *Formatted I/O* -- refers to the coThe basic format of a **printf** function call is:
* printf (*format\_string*, *list\_of\_expressions*);

where:

* + *format\_string* is the layout of what's being printed
  + *list\_of\_expressions* is a comma-separated list of variables or expressions yielding results to be inserted into the output
* To output string literals, just use one parameter on printf, the string itself
* printf("Hello, world!\n");

printf("Greetings, Earthling\n\n");

A conversion specifier is a symbol that is used as a placeholder in a formatting string. For integer output (for example), %d is the specifier that holds the place for integers. nversion of data to and from a stream of characters, for printing (or reading) in plain text format

* All text I/O we do is considered *formatted* I/O
* The other option is reading/writing direct binary information (common with file I/O, for example)

Here are some commonly used conversion specifiers (not a comprehensive list):

%d int (signed decimal integer)

%u unsigned decimal integer

%f floating point values (fixed notation) - float, double

%e floating point values (exponential notation)

%s string

%c character

**Printing Integers**

* To output an integer, use %d in the format string, and an integer expression in the *list\_of\_expressions*.
* int numStudents = 35123;
* printf("FSU has %d students", numStudents);
* // Output:
* // FSU has 35123 students
* We can specify the field wicth (i.e. how many 'spaces' the item prints in). Defaults to right-justification. Place a number between the % and the d. In this example, field width is 10:
* printf("FSU has %10d students", numStudents);
* // Output:
* // FSU has 35123 students
* To left justify, use a negative number in the field width:
* printf("FSU has %-10d students", numStudents);
* // Output:
* // FSU has 35123 students
* If the field width is too small or left unspecified, it defaults to the minimum number of characters required to print the item:
* printf("FSU has %2d students", numStudents);
* // Output:
* // FSU has 35123 students
* Specifying the field width is most useful when printing multiple lines of output that are meant to line up in a table format

**Printing Floating-point numbers**

* Use the %f modifer to print floating point values in fixed notation:
* double cost = 123.45;
* printf("Your total is $%f today\n", cost);
* // Output:
* // Your total is $123.450000 today
* Use %e for exponential notation:
* printf("Your total is $%e today\n", cost);
* // Output:
* // Your total is $1.234500e+02 today

Note that the e+02 means "times 10 to the 2nd power" 

* You can also control the decimal precision, which is the number of places after the decimal. Output will round to the appropriate number of decimal places, if necessary:
* printf("Your total is $%.2f today\n", cost);
* // Output:
* // Your total is $123.45 today
* Field width can also be controlled, as with integers:
* printf("Your total is $%9.2f today\n", cost);
* // Output:
* // Your total is $ 123.45 today

In the conversion specifier, the number before the decimal is field width, and the number after is the precision. (In this example, 9 and 2).

* + %-9.2 would left-justify in a field width of 9, as with integers

**Printing characters and strings**

* Use the formatting specifier %c for characters. Default field size is 1 character:
* char letter = 'Q';
* printf("%c%c%c\n", '\*', letter, '\*');
* // Output is: \*Q\*
* Use %s for printing strings. Field widths work just like with integers:
* printf("%s%10s%-10sEND\n", "Hello", "Alice", "Bob");
* // Output:
* // Hello AliceBob END

[output.c](https://www.cs.fsu.edu/~myers/c++/examples/io/output.c) -- contains all of the above sample outputs. Try running it yourself 

**scanf**

**Basics**

* To read data in from standard input (keyboard), we call the **scanf** function. The basic form of a call to scanf is:
* scanf(*format\_string*, *list\_of\_variable\_addresses*);
  + The format string is like that of printf
  + But instead of expressions, we need space to store incoming data, hence the list of variable addresses
* If **x** is a variable, then the expression **&x** means "address of x"
* scanf example:
* int month, day;
* printf("Please enter your birth month, followed by the day: ");
* scanf("%d %d", &month, &day);
* **Conversion Specifiers**
  + Mostly the same as for output. Some small differences
  + Use %f for type float, but use %lf for types double and long double
* The data type read, the conversion specifier, and the variable used need to match in type
* White space is skipped by default in consecutive *numeric* reads. But it is *not* skipped for character/string inputs.

**Example**

[input1.c](https://www.cs.fsu.edu/~myers/c++/examples/io/input1.c) -- linked here

#include <stdio.h>

int main()

{

int i;

float f;

char c;

printf("Enter an integer and a float, then Y or N\n> ");

scanf("%d%f%c", &i, &f, &c);

printf("You entered:\n");

printf("i = %d, f = %f, c = %c\n", i, f, c);

}

**Sample run #1**

User input underlined, to distinguish it from program output

Enter an integer and a float, then Y or N

> 34 45.6Y

You entered:

i = 34, f = 45.600, c = Y

**Sample Run #2**

Enter an integer and a float, then Y or N

> 12 34.5678 N

You entered:

i = 12, f = 34.568, c =

Note that in this sample run, the character that was read was **NOT** the letter 'N'. It was the space. (Remember, white space not skipped on character reads).

This can be accounted for. Consider if the scanf line looked like this:

scanf("%d%f %c", &i, &f, &c);

There's a space betwen the %f and the %c in the format string. This allows the user to type a space. Suppose this is the typed input:

12 34.5678 N

Then the character variable c will now contain the 'N'.

[input2.c](https://www.cs.fsu.edu/~myers/c++/examples/io/input2.c) -- a version of the example with this change is linked here 

**Interactive Input**

You can make input more interactive by prompting the user more carefully. This can be tedious in some places, but in many occasions, it makes programs more user-friendly. Example:

int age;

double gpa;

char answer;

printf("Please enter your age: ");

scanf("%d", &age);

printf("Please enter your gpa: ");

scanf("%lf", %gpa);

printf("Do you like pie (Y/N)? ");

scanf("%c", %answer);

A good way to learn more about scanf is to try various inputs in various combinations, and type in test cases -- see what happens!

**printf/scanf with C-strings**

An entire C-style string can be easily printed, by using the %s formatting symbol, along with the name of the char array storing the string (as the argument filling in that position):

char greeting[] = "Hello";

printf("%s", greeting); // prints the word "Hello"

Be careful to **only** use this on char arrays that are being used as C-style strings. (This means, only if the null character is present as a terminator).

Similarly, you can read a string into a char array with scanf. The following call allows the entry of a word (up to 19 characters and a terminating null character) from the keyboard, which is stored in the array word1:

char word1[20];

scanf("%s", word1);

Characters are read from the keyboard until the first "white space" (space, tab, newline) character is encountered.  The input is stored in the character array and the null character is automatically appended.

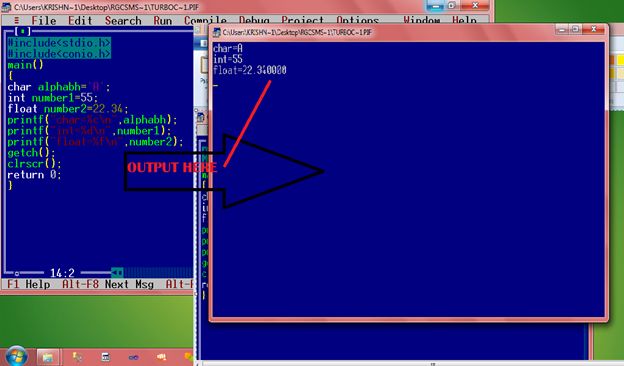
C provides standard functions scanf() and printf(), for performing formatted input and output .These functions accept, as parameters, a format specification string and a list of variables. The format specification string is a character string that specifies the data type of each variable to be input or output and the size or width of the input and output.  
  
Now to discuss formatted output in functions.  
  
**Formatted Output**The function printf() is used for formatted output to standard output based on a format specification. The format specification string, along with the data to be output, are the parameters to the printf() function.  
 **Syntax:**  
printf (format, data1, data2,……..);  
  
In this syntax format is the format specification string. This string contains, for each variable to be output, a specification beginning with the symbol % followed by a character called the conversion character.  
  
**Example:**printf (“%c”, data1);  
  
The character specified after % is called a conversion character because it allows one data type to be converted to another type and printed.  
  
See the following table conversion character and their meanings.

|  |  |
| --- | --- |
| Conversion Character | Meaning |
| d | The data is converted to decimal (integer) |
| c | The data is taken as a character. |
| s | The data is a string and character from the string , are printed until a NULL, character is reached. |
| f | The data is output as float or double with a default Precision 6. |

|  |  |
| --- | --- |
| Symbols | Meaning |
| \n | For new line (linefeed return) |
| \t | For tab space (equivalent of 8 spaces) |

**Example**  
printf (“%c\n”,data1);  
  
The format specification string may also have text.  
  
**Example**  
printf (“Character is:”%c\n”, data1);  
  
The text "Character is:" is printed out along with the value of data1.

1. #include<stdio.h>
2. #include<conio.h>
3. Main()
4. {
5. Char alphabh="A";
6. **int** number1= 55;
7. **float** number2=22.34;
8. printf(“**char**= %c\n”,alphabh);
9. printf(“**int**= %d\n”,number1);
10. printf(“**float**= %f\n”,number2);
11. getch();
12. clrscr();
13. retrun 0;
14. }
15. Output Here…
16. **char** =A
17. **int**= 55
18. flaot=22.340000

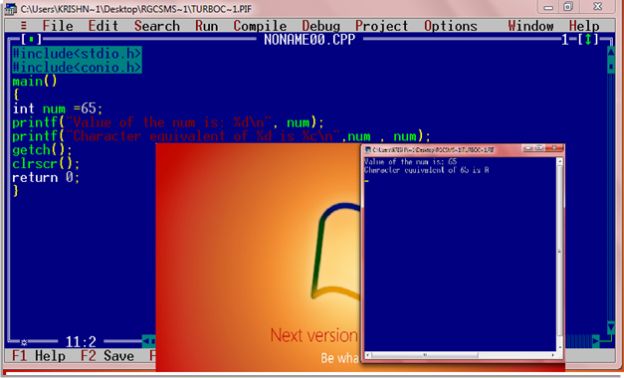
See the following image:  
  
  
  
**What is the output of the statement?**printf(“Integer is: %d; Alphabet is:%c\n”,number1, alpha);  
  
Where number1 contains 44 and alpha contains "Krishna Singh".   
  
Give the answer below.  
  
Between the character % and the conversion character, there may be:

* A minus sign: Denoting left adjustment of the data.
* A digit: Specifying the minimum width in which the data is to be output, if the data has a larger number of characters then the specified width occupied by the output is larger. If the data consists of fewer characters then the specified width, it is padded to the right or to the left (if minus sign is not specified) with blanks. If the digit is prefixed with a zero, the padding is done with zeros instead of blanks.
* A period: Separating the width from the next digit.
* A digit following the period: specifying the precision (number of decimal places for numeric data) or the maximum number of characters to be output.
* Letter 1: To indicate that the data item is a long integer and not an int.

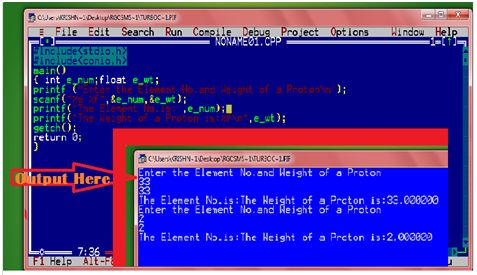
|  |  |  |
| --- | --- | --- |
| Format specification string | Data | Output |
| |%2d| | 9 | |9| |
| |%2d| | 123 | |123| |
| |%03d| | 9 | |009| |
| |%-2d| | 7 | |7| |
| |%5.3d| | 2 | |002| |
| |%3.1d| | 15 | |15| |
| |%3.5d| | 15 | |0015| |
| |%5s| | “Output sting” | |Output string| |
| |%15s| | “Output sting” | |Output string| |
| |%-15s| | “Output sting” | |Output string| |
| |%15.5s| | “Output sting” | |Output string| |
| |%.5s| | “Output sting” | |Output| |
| |%15.5s| | “Output sting” | |Output| |
| |%f| | 87.65 | |87.650000| |
| |%.4.1s| | 87.65 | |87.71| |

Example based on the conversion character:

1. #include<stdio.h>
2. #include<conio.h>
3. main()
4. {
5. Int num=65;
6. printf(“Value of num **is** : %d\n:, num);
7. printf(“Character equivalent of %d **is** %c\n”, num , num);
8. getch();
9. clrscr();
10. rerurn o;
11. }

Output Here…  
char =A  
int= 55  
flaot=22.340000  
  
  
  
**Formatted Input**  
The function scanf() is used for formatted input from standard input and provides many of the conversion facilities of the function printf().  
  
**Syntax**scanf (format, num1, num2,……);  
The function scnaf() reads and converts characters from the standards input depending on the format specification string and stores the input in memory locations represented by the other arguments (num1, num2,….).  
  
For Example:  
  
scanf(“ %c %d”,&Name, &Roll No);  
  
**Note**: the data names are listed as &Name and &Roll No instead of Name and Roll No respectively. This is how data names are specified in a scnaf() function. In case of string type data names, the data name is not preceded by the character &.  
 **Example with program**  
Write a function to accept and display the element number and the weight of a proton. The element number is an integer and weight is fractional.  
  
Solve here:

1. #include<stdio.h>
2. #include<conio.h>
3. main()
4. {
5. Int e\_num;
6. Float e\_wt;
7. printf (“Enter the Element No. and Weight of a Proton\n”);
8. scanf (“%d %f”,&e\_num, &e\_wt);
9. printf (“The Element No.**is**:”,e\_num);
10. printf (“The Weight of a Proton **is**: %f\n”, e\_wt);
11. getch();
12. **return** 0;
13. }

 We can classify I/O function into two categories:

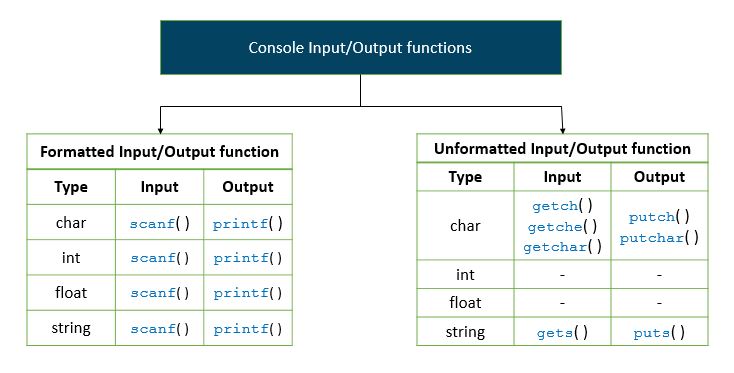
1. **Console I/O function**
2. **File I/O function**

Console Input-Output functions

Console simply means screen and keyboard. There are two types of a console I/O functions:

* **Formatted input-output function**
* **Unformatted input-output function**

The major difference is that formatted function allows us to format the input from the keyboard and the output to be displayed on the screen.

 printf( ) is the standard library function that is used for precise output formatting.

syntax of printf( ) function:

 printf( format-control-string, other-arguments );

Format control string in printf( ) function describes the output format which consists of conversion specifiers, precisions, flags, field widths and literal characters.

Each conversion specifier starts with % sign and ends with a conversion specifier.

We can perform following formatting options with the help of printf( ):

* Right and left justification.
* Rounding floating-point values.
* Inserting literal characters.
* Displaying all types of data with appropriate size and precisions.

**How printf( ) funtion works?**

#include <stdio.h>

int main ()

{

int num = 22;

printf("\nNumber: %d", num);

return 0;

}

**Output**

Number: 22

**Explanation**

First of all, printf( ) scans the format control string from left to right. It shows characters onto the screen until it encounters % or \.

In the above program, first, it encounters \n and places the cursor at the beginning of the new line.

Now, Number: is displayed on the screen.

When it encounters conversion specification in the format control string it picks up the first variable from other arguments such as num in the above example.

This process continues until the end of format control string.

**Integer conversion specifiers**

An integer is a whole number that can be displayed in several formats.

|  |  |
| --- | --- |
| **Conversion Specifier** | **Description** |
| d | Signed decimal Integer |
| i | Signed decimal integer |
| o | An unsigned octal integer |
| u | An unsigned decimal integer |
| x or X | An unsigned hexadecimal integer |
| h, I or II | When placed before conversion specifier indicates short, long or long long integer. These are called **length modifiers.** |

**Floating-point conversion specifier**

A floating-point value contains decimal point.

|  |  |
| --- | --- |
| **Conversion Specifier** | **Description** |
| e or E | Floating-point values in exponential notation |
| f or F | Floating-point values in fixed notation |
| g or G | Floating-point value in either floating-point form or exponential form based on length of the value |
| L | Indicates a long double floating-point value if placed before floating-point conversion specifier |

**Conversion Specifier for strings and character**

|  |  |
| --- | --- |
| **Conversion Specifier** | **Description** |
| c | Print individual character |
| s | Print string until terminating null character \0 is encountered and requires a pointer to char as an argument |

**Printing with field width and precision**

With the help of field width, we can change the position of data being displayed on the screen.

If the field width is greater than data being displayed, then the data is right justified.

#include <stdio.h>

int main ()

{

printf("%4d\n", 1);

printf("%4d\n", 12);

printf("%4d\n", 123);

printf("%4d\n", 1234);

printf("%4d\n", 123456);

}

**Output**

1

12

123

1234

123456

Precision has a different meaning when used with different data types.

When used with the integer, it indicates the minimum number of digits to be printed.

Thus, if the value to be printed contains fewer digits than the precision value, zeros are prefixed until the total number of the digit is equivalent to the precision.

When used with the floating-point, it indicates the number of digits to appear after the decimal point.

**Example: Use of precision for integer, floating-point, and strings**

#include <stdio.h>

int main ()

{

int num1 = 123;

double num2 = 123.45678;

char s[] = "Hello World!!!";

puts("Use of precision for integer");

printf("\t%.4d\n\t%.8d\n\n", num1, num1);

puts("Use of precision for floating-point");

printf("\t%.2f\n\t%.2e\n\n", num2, num2);

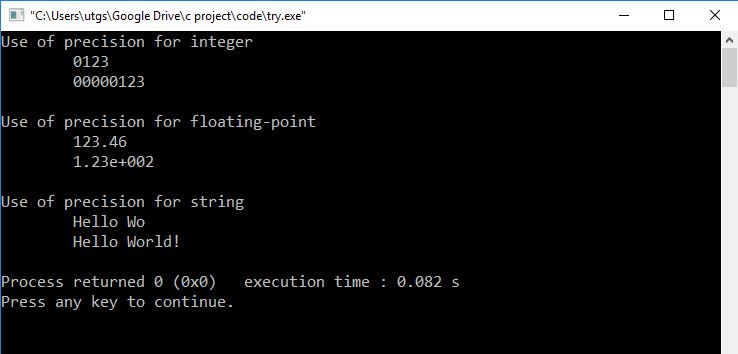
puts("Use of precision for string");

printf("\t%.8s\n\t%.12s\n", s, s);

return 0;

}

**Output**



## C programming I/O function: scanf( )

scanf( ) is the standard library function that is used for precise input formatting.

### syntax of scanf( ) function:

 scanf( format-control-string, other-arguments );

Format control string in scanf( ) function describes the input format and other argument are pointers to list of variables.

Each conversion specifier starts with % sign and ends with a conversion specifier.

We can perform the following formatting using scanf( ) function

* Inputting certain characters from input.
* Skipping certain characters from input.
* Inputting all types of data.

Conversion specifiers for scanf( ) function are same as printf( ).

Let’s discuss some important features of scanf( ) function.

**Reading character using scan set**

We can only read certain characters from the input stream using **scan set**, set of characters enclosed in the square bracket [ ] that is preceded by a percent % sign.

The characters from the input stream that matched the character in scan set are stored in the array, otherwise, stops inputting characters when a character that is not stored in scan set is encountered.

On the other hand, if we want to omit certain characters from input stream we should place a care (^) sign before the scan characters. This is called an **inverted scan set**.

Inverted scan set is opposite of scan set.

**Demonstration of scan set in scanf( )**

#include <stdio.h>

int main ()

{

char s[ 9 ];

printf("Enter string: ");

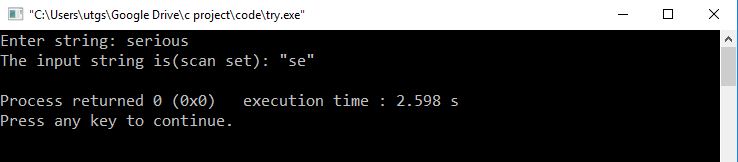
scanf("%9[selz]", s);

printf("The input string is: \"%s\"\n", s);

return 0;

}

**Output**



**Demonstration of Inverted scan set in**scanf**( )**

#include <stdio.h>

int main ()

{

char s[ 10 ];

printf("Enter string: ");

scanf("%9[^selz]", s);

printf("The input string is(Inverted scan set): \"%s\"\n", s);

return 0;

}

**Output**

