C Programming Lecture 2

Tuesday, 4 June 2024 7:03 PM

printf() and scanf() in C

The printf() and scanf() functions are used for input and output in C language. Both functions are inbuilt library functions, defined in stdio.h (header file).

printf() function

The **printf() function** is used for output. It prints the given statement to the console.
The syntax of printf() function is given below:

```
printf("format string",argument_list);
```

scanf() function

The **scanf() function** is used for input. It reads the input data from the console.

```
scanf("format string",argument_list);
```

```
scanf("%d...", ...);
printf("%d...", ...);
```

Format Specifiers in C

The format specifier in C is used to tell the compiler about the type of data to be printed or scanned in input and output operations.

They always start with a **%** symbol and are used in the formatted string in functions like printf(), scanf(), etc.

The C language provides a number of format specifiers that are associated with the different data types such as %d for int, %c for char, etc.

Format Specifier	Description
%c	For character type.
%d	For signed integer type.
%e or %E	For scientific notation of floats.
%f	For float type.
%g or %G	For float type with the current precision.
%i	Integer
%ld or %li	Long
%lf	Double
%Lf	Long double
%lu	Unsigned int or unsigned long
%lli or %lld	Long long
%llu	Unsigned long long
%o	Octal representation
%p	Pointer
%s	String
%u	Unsigned int
%x or %X	Hexadecimal representation
%n	Prints nothing
%%	Prints % character

Escape Sequence in C

Programming languages like C have ***escape sequences*** as a standard feature.

They enable the inclusion of ***special characters*** and ***control patterns*** in strings and character constants that are difficult to represent or ***write directly***.
An ***escape sequence*** in the C programming language consists of a ***backslash ()*** and a character that stands in for a ***special character*** or ***control sequence***

It is composed of two or more characters starting with backslash \. For example: \n represents new line.

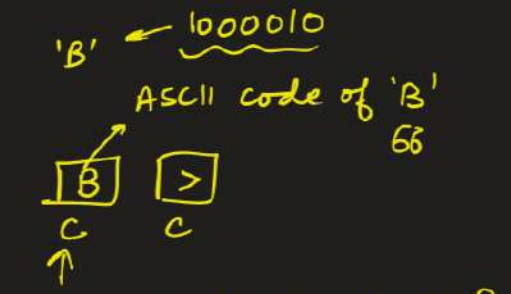
Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark
\nnn	octal number
\xhh	hexadecimal number
\0	Null

1. Character Format Specifier – %c in C

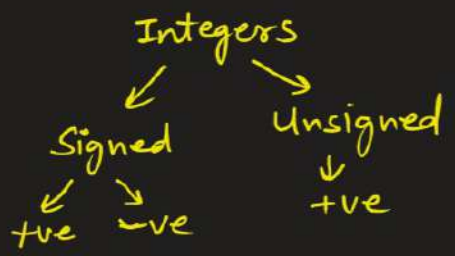
```c
// C Program to illustrate the %c format specifier.
#include <stdio.h>

int main() {

    char c;
    // using %c for character input
    scanf("%c", &c);

    // using %c for character output
    printf("The entered character: %c", c);
    return 0;
}
```

Declaration of a variable

Format specifier



2. Integer Format Specifier (signed) – %d in C




```c
// C Program to demonstrate the use of %d and %i
#include <stdio.h>

// Driver code
int main() {

    int x; // Variable declaration
    // taking integer input
    scanf("%d", &x);

    // printing integer output
    printf("Printed using %%d: %d\n", x);
    printf("Printed using %%i: %3i\n", x);
    return 0;
}
```

66 (x) 1000010 → 66

Printed using %d :_ 66
Printed using %i :__ 66
(extra space)

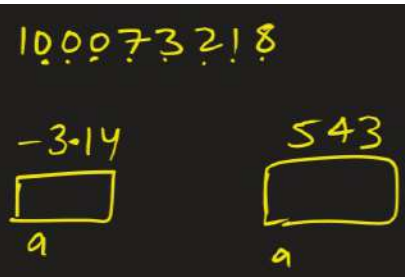
6666 (x) %2i ?

3. Unsigned Integer Format Specifier – %u in C

```c
// C Program to illustrate the how to use %u
#include <stdio.h>

// driver code
int main() {

    unsigned int var;

    scanf("%u", &var);

    printf("Entered Unsigned Integer: %u", var);

    return 0;
}
```

4. Floating-point format specifier – %f in C

```c
// C program to demonstrate the use of %f, %e  and %E
#include <stdio.h>

// driver code
int main() {

    float a;

    scanf("%f", &a);

    printf("Using %%f: %f\n", a);
    printf("Using %%e: %e\n", a);
    printf("Using %%E, %E", a);
    return 0;
}
```



%.f → -3.14
%.e → 5.43e2
%.E → 5.43E2

1.0007e8

5. String Format Specifier – %s in C

```c
// C program to illustrate the use of %s in C
#include <stdio.h>

int main() {

    char a[] = "Hi Students";
    printf("%s\n", a);
    return 0;
}
```

Input and Output Formatting

C language provides some tools using which we can format the input and output. They are generally inserted between the % sign and the format specifier symbol Some of them are as follows:

1. A minus(-) sign tells left alignment.
2. A number after **%** specifies the minimum field width to be printed if the characters are less than the size of the width the remaining space is filled with space and if it is greater then it is printed as it is without truncation.
3. A period(.) symbol separates field width with precision. Precision tells the minimum number of digits in an integer, the maximum number of characters in a string, and the number of digits after the decimal part in a floating value.

```c
// C Program to demonstrate the
// formatting methods.

#include <stdio.h>
int main() {
    char str[] = "Hello World!";
    printf("%20s\n", str);
    printf("%-20s\n", str);
    printf("%20.5s\n", str);
    printf("%-20.5s\n", str);
    return 0;
}
```

```
            Hello World!
Hello World!
```

```c
// C Program to demonstrate the
// formatting methods.

#include <stdio.h>
int main() {
    int a=345, b=2, c=56;
    int d=6, e=777, f=678;
    char n1[] = "Subham";
    char n2[] = "Shyam";
    char n3[] = "Gita";
    printf("%10s | %10s | %10s \n", n1, n2, n3);
    printf("%10d | %10d | %10d \n", a, b, c);
    printf("%10d | %10d | %10d", d, e, f);
    return 0;
}
```

Variables in C

A variable is the name of the memory location. It is used to store information. Its value can be altered and reused several times. It is a way to represent memory location through symbols so that it can be easily identified.

Variables are key building elements of the C programming language used to store and modify data in computer programs.

A variable is a designated memory region that stores a specified data type value. Each variable has a ***unique identifier***, its ***name***, and a ***data type*** describing the type of data it may hold.

```c
int a;
float b;
char c;
int a=20, b=30; //declaring 2 variable of integer type
float f=20.9;
char c='A';
```

Syntax:

The syntax for defining a variable in C is as follows:

```
data_type variable_name;
```

int a;
float ff;

Here,

- **data_type:** It represents the type of data the variable can hold. Examples of data types in C include ***int (integer), float (a floating-point number), char (character), double (a double-precision floating-point number),***

- **variable_name:** It is the identifier for the variable, i.e., the name you give to the variable to access its ***value*** later in the program. The variable name must follow specific rules, like starting with a ***letter*** or ***underscore*** and consisting of ***letters***, digits, and underscores.

Rules for defining variables

In C, ***Variable names*** must follow a few rules to be valid. The following are the rules for naming variables in C:

- **Allowed Characters:** Variable names include letters (uppercase and lowercase), digits, and underscores. They must start with a letter (uppercase or lowercase) or an underscore.
- **Case Sensitivity:** C is a ***case-sensitive*** programming language. It means that ***uppercase*** and ***lowercase letters*** are considered distinct. For example, ***myVar, MyVar,*** and ***myvar*** are all considered different variable names.
- **Keywords:** ***Variable names*** cannot be the same as ***C keywords (reserved words)***, as they have special meanings in the language. For example, you cannot use ***int, float, char, for, while***, etc., as ***variable names***.
- **Length Limitation:** There is no standard limit for the length of variable names in C, but it's best to keep them reasonably short and descriptive. Some compilers may impose a maximum length for variable names. (31 characters)
- **Spaces and Special Characters:** ***Variable names*** cannot contain ***spaces*** or special characters (such as ***!, @, #, $, %, ^, &, *, (,), -, +, =, [,], {, }, |, \, /, <, >,., ?;, ', or "***). ***Underscores*** are the only allowed special characters in variable names.
- **Reserved Identifiers:** While not strictly a rule, it is advisable to avoid ***specific patterns*** or ***identifiers*** common in libraries or standard usage. For example, ***variables*** starting with ***__ (double underscores)*** are typically reserved for system or ***compiler-specific*** usage.

Suggestion

stNumber1

Valid examples of variable names:

```
int age;
float salary;
char _status;
double average_score;
int studentCount;
```

Invalid examples of variable names:

```
int 1stNumber; // Starts with a digit
float my-salary; // Contains a hyphen (-)
char int; // Same as a C keyword
int double; // Same as a C keyword
float my
$var; // Contains an unsupported special character
```

Q:

Which symbol can be used in variable name?

A) * (asterisk)

B) | (pipeline)

C) – (hyphen)

D) _ (underscore)

Q:

Consider the following C code snippet:

```
#include <stdio.h>

int main() {
    int value1 = 10;
    int _Value2 = 20;
    int Value_3 = 30;
    int Value$4 = 40;

    printf("%d %d %d %d\n", value1, _Value2, Value_3, Value$4);
    return 0;
}
```

$ is not allowed when naming variables

Which of the following statements is true about the given code?

A. The code will compile and run successfully, printing 10 20 30 40.

B. The code will compile but produce a warning due to the variable name Value$4.

C. The code will not compile because Value$4 is not a valid variable name.

D. The code will not compile because variable names cannot contain both uppercase and lowercase letters.

The Three components of declaring a variable

Let us explain the ***three aspects*** of defining a variable: ***variable declaration, variable definition, and variable initialization***, along with examples.

1. Variable Declaration:

The process of telling the ***compiler*** about a variable's existence and data type is known as ***variable declaration***.
It notifies the compiler that a variable with a specific name and data type will be used in the program.
Still, no memory for the variable is allocated at this moment. It is usually seen at the start of a function or block before the variable is utilized.

int a;

The general syntax for variable declaration is

```
data_type variable_name;
```

Example of variable declaration:

```c
#include <stdio.h>

int main() {
    // Variable declaration
    int age;
    float salary;
    char initial;

    return 0;
}
```

2. Variable Definition:

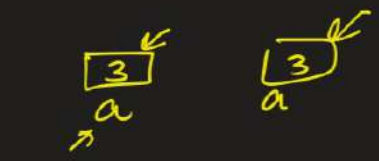
The process of ***reserving memory space*** for the variable to keep its contents during program execution is known as a ***variable definition***. It is based on the data type and connects the variable name with a particular memory address of sufficient size.

A variable in C can be declared and defined in the same statement, although they can also be separated if necessary.

Example of variable definition:

```c
#include <stdio.h>

int main() {
    // Variable definition
    int age = 25;
    float salary = 2500.5;
    char initial = 'J';

    return 0;
}
```

int a = 3;



Declare
Initialize
Define = Delare + Initialize

3. Variable Initialization:

Variable declaration is the act of informing the compiler about the existence and ***data type*** of a variable. It informs the compiler that a variable with a specific name and data type will be used in the program, but that memory for the variable still needs to be allocated.

Not explicitly initialized variables will contain ***garbage/random data*** that may result in unexpected program behavior.

Example of variable initialization:

```c
#include <stdio.h>
int main() {

    // Variable definition and initialization
    int age = 25;
    float salary = 2500.5;
    char initial = 'J';

    // Later in the program, you can change the value of the variable
    age = 30;
    salary = 3000.0;

    return 0;
}
```