Variables, Data Types and Operators

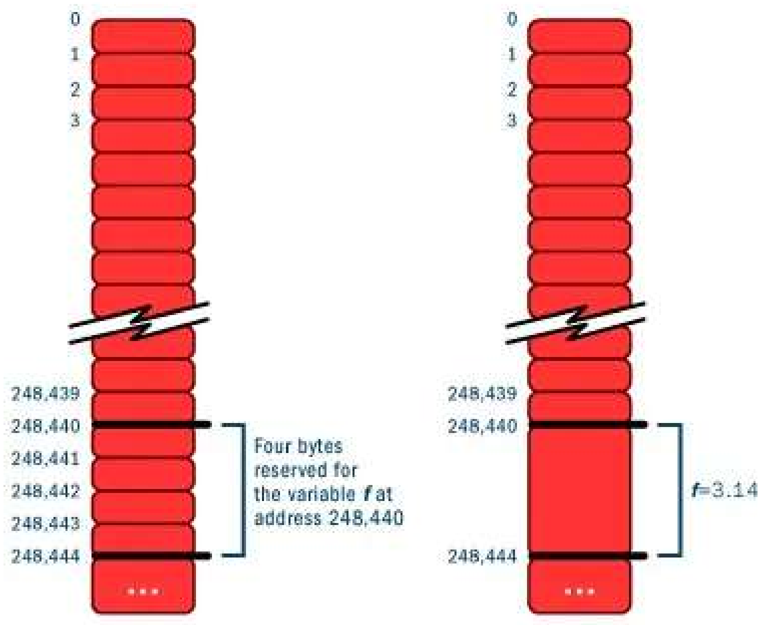
Variables and Constants

Remember those questions in maths whose solutions started with the famous “Let’s assume the answer to be x ”. You used to dutifully solve tedious equations to find the value of x and this value came out different for different questions. X here was a variable and the value we found for it was a constant. Variables and constants in programming are very similar to this.

They act like “containers” that help store, retrieve, and sometimes, modify data. Variables hold values that can be modified when a program is executed. Constants, on the other hand, hold values that remain unchanged throughout the execution of a program. Behind the scenes, variables and constants define areas in memory in which values(data) are stored. Let's have a brief look at how through an example.

The image represents memory in a computer. Each cell has a capacity of 1 byte and an address indicated on the left. Say you want to store the number 3.14 in memory. 3.14 requires 4 bytes of memory. But why? We'll discuss this in the very next topic . For now, take it as is. So first, the computer allocates 4 bytes of memory at some random address, 248,440 to 248,444 in this case. Then the value 3.14 is stored at that address in binary form. Once the value is stored, what if we want to retrieve it from its random address. This is the role of variables and constants, they act as "identifiers" for these blocks of memory, and provide the user with access to this memory and its data. So instead of having to remember these memory locations, the user can simply refer to the memory by a set name.

This is all well and good, but how do actually put this all into effect in code?



This is technically called Declaration of a variable, and is done in three parts:-

Data Type

These are special words in every language used to define how much memory the variable requires. In the example above, our variable has type “int” which tells the computer that it requires 4 bytes of memory. We will discuss more these data types, in just the next topic.

Variable name

The next part is the variable name. This is that user-defined name, we referred to earlier. This is the identifier by which the memory block will be referred. Can we use “int” as a variable name? Won’t that be confusing? No, “int” cannot be used as a variable name. There are some special predefined words in every language that cannot be used as identifiers. These are called keywords. These keywords are language-specific, i.e different languages have different keywords.

To know more about these keywords, visit this link.(complete this link)

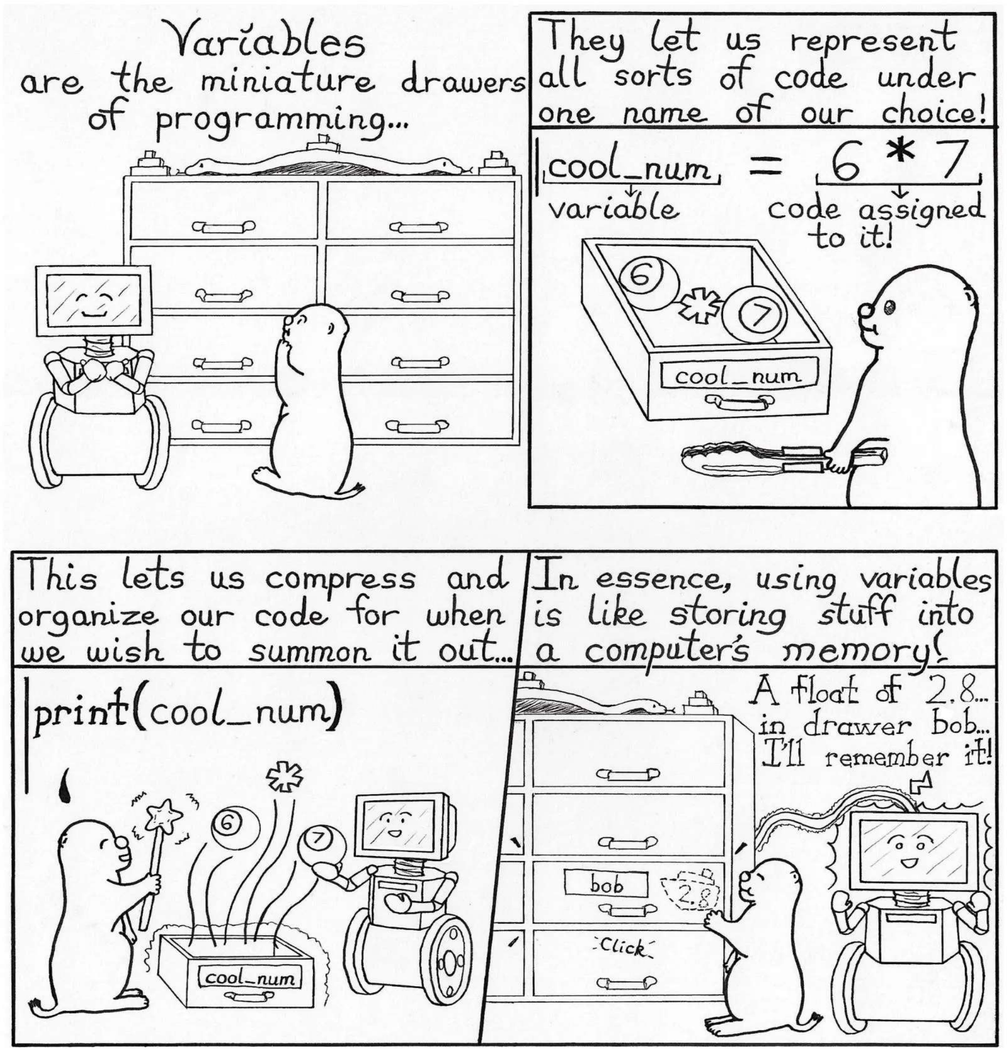
What about “$money”? Can we name our variable “$money”? No, other than not using keywords, there are some more rules that need to be followed while naming variables. To know more, visit the link.

Value

The last part is the value. It is the actual data to be stored in memory.

Note while discussing the parts we used only “variables” and not “variables and constants”. Were these parts only applicable to variables? No, constants also have a very similar declaration, the only difference being that usually some extra special word is appended at the beginning such as “const”.

Let's summarize,


Variables are the miniature drawers of programming...
They let us represent all sorts of code under one name of our choice!
cool_num = 6 * 7
variable
code assigned to it!
cool_num
This lets us compress and organize our code for when we wish to summon it out...
print(cool_num)
cool_num
In essence, using variables is like storing stuff into a computer's memory!
A float of 2.8... in drawer bob... I'll remember it!
bob
2.8
Click

```
INITIATING.....WAIT...WAIT A BIT MORE...ALMOST
//////////////////////////////////////
.......FUNK SUBROUTINE V2.09.........
WELCOME TO FUNKTOWN, BROTHER
DEFINE CRUNK PARAMETERS: SUPER CRUNKY
WORKING......................................
CRUNKYNESS SET TO SETTING: SUPER(SUPER CRUNKY)
C:/LOVE_THE_WAY_YOU_MOVE.MP3 -LOAD TO EXT. DEVICE
WRITTEN SUCCESSFULLY TO EXT. DEVICE
C:/DROP_DA_BEAT.EXE -P(LOVE_THE_WAY_YOU_MOVE.MP3)
BEAT DROPPING................................
BEAT SUCCESSFULLY DROPPED. IS IT FRESH? Y/N
Y
TIGHT. INITIATE FLY SOVIET DANCE KICKS TESTING
PHASE? Y/N
Y
INITIATING....PLEASE WAIT.....................
.............................................
SUCCESSFULLY INITIATED.
C:/BASH
BASH 3.2$ LS -LRT
FUNK      DROPS      MAD_BEATS      FLY_KICKS
DANCE     POPS       LOCKS          CRUNK
BASH 3.2$ CD DANCE
```

Data Types

4 + 5 = ? Ajay - 1 = ?

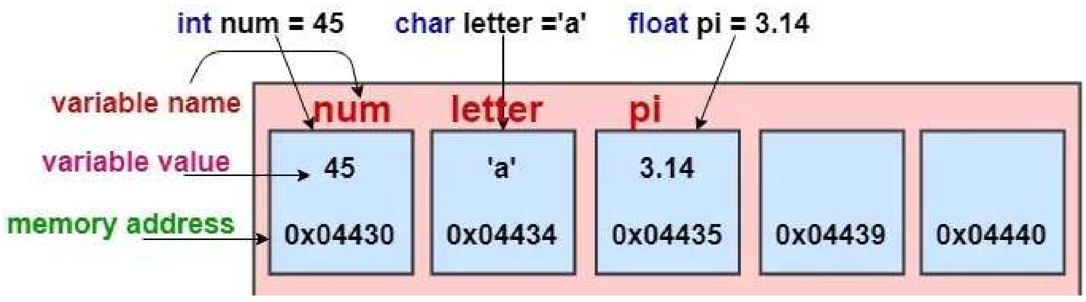
We have briefly discussed data types. Let’s elaborate on that in this section.

Suppose we want our program to do simple operations like addition and subtraction. So, I gave the above instructions to the computer through my program. What do you think the output will be?

Of course, 4+5=9. But Ajay-1 will give an error because we can add and subtract integers but not an integer and a word. How does the computer decide this? This is the role of data types. All programming languages have predefined data types that indicate how much memory some data requires(we touched on this in variables and constants) and specify the operations that are allowed on that data.

All programming languages support data in the form of integers, floating-point numbers, and characters, and use different keywords to specify them. Let’s see some basic data types of the C programming language:

- char: It stores a single character and requires a single byte of memory in almost all compilers.
- int: As the name suggests, an int variable is used to store an integer. It requires four bytes of memory.
- float: It is used to store decimal numbers (numbers with floating-point value) with 7 digits of precision. It also requires four bytes of memory. This is the reason why in the previous section we said 3.14 requires 4 bytes of memory.



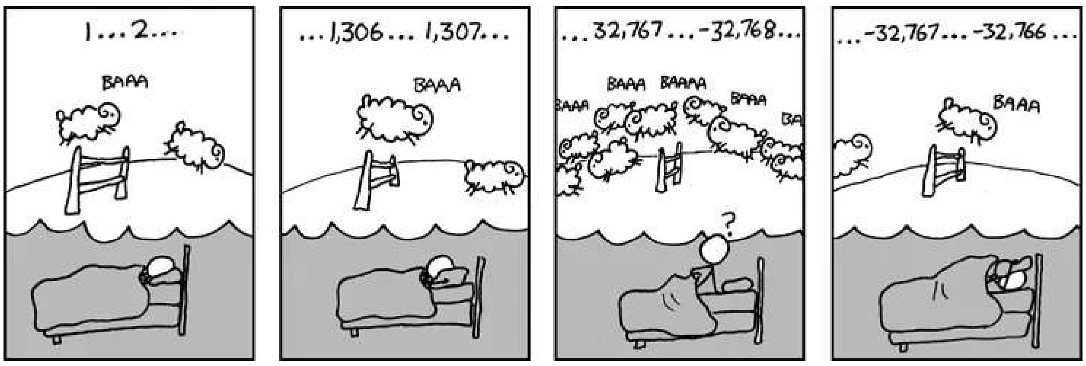
Note that these are not all, there are many more data types but those are smaller or larger memory versions of the same three data types. To learn about these visit this link.(Complete this link)

We now know how much memory each data type requires. But that is not very useful for us as a user. We are more concerned with the range of values. Let’s learn how to find the range of values for a four-byte integer.

We know that each byte contains 8 bits and so 4 bytes will contain 32 bits. And each bit can be either zero or one. So the number of possible combinations will be 2^{32}(Simple P and C, right?). So, if we were to have only non-negative numbers, we could have numbers from 0 to 2^{32}-1. However, generally, when we use the integer data type it is convenient to have both positive and negative numbers. Hence, the range is taken as -2^{31} to 2^{31}-1. Note that the total number still remains 2^{32} (2^{31} negative + zero + 2^{31}-1 positive). We have found out the range for the int data type. It can be similarly applied to any data type for integer values. Now for a small assignment.
Try finding out how you can determine the range of float and char data types.

The range of an integer is all well and good now. But what if we try putting a value out of the range inside a variable. This leads to something called an overflow. So as values go above the highest value, they start again from the lowest value.

Something like,



A very important point to be kept in mind is that different data types cannot be compared or operated together by the computer. That is why “Ajay - 5” failed.

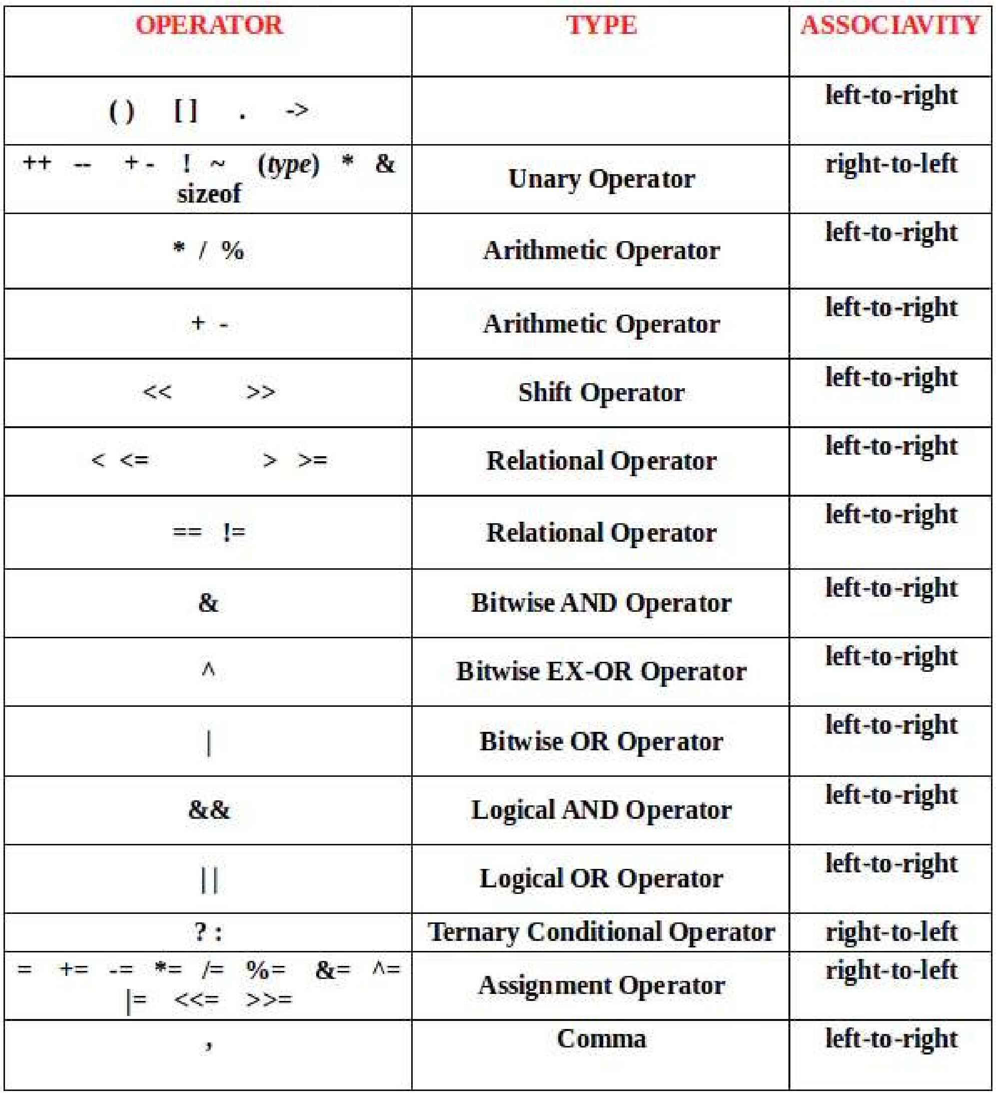
For eg :- Int + float (not acceptable)
In certain cases, however, it is possible to override this, by something called casting. To learn more about casting, click here. (complete this link)

Operators

What does the computer say, when asked “What do we tell death?”. It says “!Today”. “!” is actually an operator that signifies “not”. We have already seen some operators - “+”, “-”; now let's have a look at some more operators commonly used in programming. But first a proper definition.

Operators are the special symbols used in programming languages that have a specific meaning. They may vary across languages but they are a core concept in them all. Arithmetic, relational and logical operators are the most common types of operators. You are recommended to visit this link(complete this link) and have a look at various types of operators in the C language.

You all must have used BODMAS in your school times to solve simple mathematical expressions, Similarly computer also follows some set of rules to evaluate its expressions, let us read them Precedence determines which operator is performed first in an expression with several operators with different precedences. Associativity is used when two operators of the same precedence appear in an expression. Associativity can either be from Left to Right or from Right to Left. For example: ‘*’ and ‘/’ have the same precedence and their associativity is Left to Right, so the expression ”100 / 10 * 10” is treated as “(100 / 10) * 10”. The precedence and associativity chart of operators is as follows:-



OPERATOR	TYPE	ASSOCIAVITY
( ) [ ] . ->		left-to-right
++ -- + - ! ~ (*type*) * & sizeof	Unary Operator	right-to-left
* / %	Arithmetic Operator	left-to-right
+ -	Arithmetic Operator	left-to-right
<< >>	Shift Operator	left-to-right
< <= > >=	Relational Operator	left-to-right
== !=	Relational Operator	left-to-right
&	Bitwise AND Operator	left-to-right
^	Bitwise EX-OR Operator	left-to-right
|	Bitwise OR Operator	left-to-right
&&	Logical AND Operator	left-to-right
||	Logical OR Operator	left-to-right
? :	Ternary Conditional Operator	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operator	right-to-left
,	Comma	left-to-right

This is what is followed by the computer.
Let’s solve a basic maths problem:
100 + 3 * 200 / 10 - 10

```
(100 + 3 * 200 / 10 - 10)     //   * and / has higher precedence than + and -
```

```
2.    (100 + 600 / 10 - 100)        // associativity of
* and / is left to right
3.    (100 + 60 - 10)               // associativity of
+ and - is again left to right
4.    (160 - 10)
5.    150
```

Q: You are given various operators and operands in a distorted manner. Arrange them and form an equation keeping in mind operator precedence and associativity such that the output comes out to be 8. The operators are

```
+ 5 / 5 4 + 2 + 4 4 / * 8 3 - 2 /
```

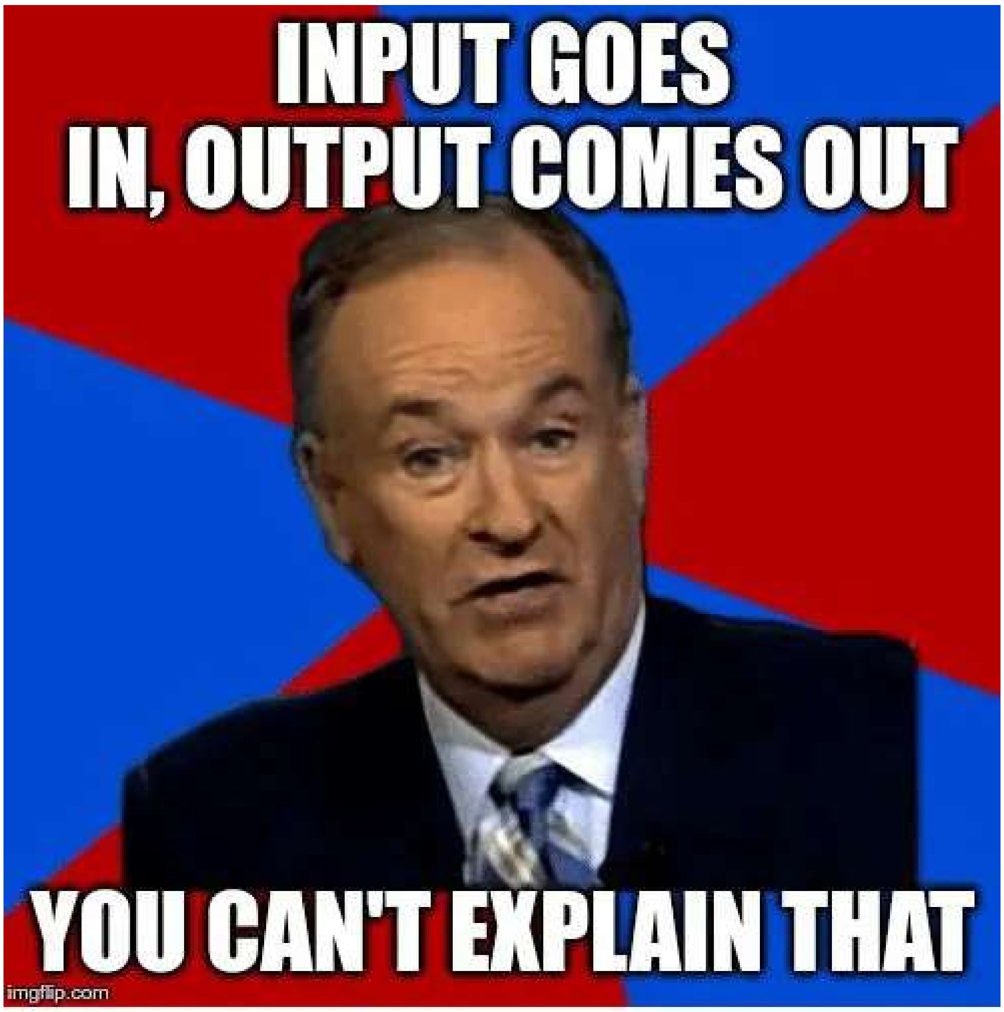
Go to this link to learn more about operators and operands to solve the next question.

```
int a = 5;
int b = 5;
int c = a || --b;
int d = a-- && --b;
int e = a++ + ++a;
```

What will be the values of a, b, c, d and e?

Input Output

Before moving on to programming constructs, it is imperative to discuss the basics of input-output in some common languages, so that our code for them does not leave you confused.



We won't bore you with the definitions of input and output. However, we are going to be taking a look at how we can take user input and output in the C language. But before that, a slight hint of functions. Functions in programming are similar to their counterparts in math.

These take some values and perform operations on these values. They are also denoted similarly as a function(). This will do for now. We have a dedicated topic later.

-C

For C, input and output is done using the scanf() and printf() functions . Let's see how they work.

So suppose you want to take a number input to your program and store it in an integer variable num. You will do this as

```
scanf("%d, &num);
```

Whoa! too many things at once. Let’s break this down. First, everything written in () brackets are the values passed to the scanf() function.

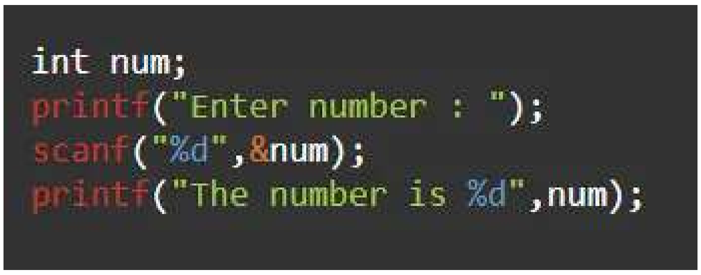
Now, what is this “%d”? This is a format specifier; it denotes the data type of value being taken as input. “%d” for example denotes that an integer is being taken as input. We will take a look at some more format specifiers in the next subtopic.

The, “,” separates the two values being passed to the function. ”num” is the variable where the value is to be stored, but what is this weird “&” before it. Is it bitwise AND we saw earlier? No, this is the address operator. Note that bitwise AND operates on two numbers, and the address operator works on variables. As the name suggests, it gets us the address in memory for our variable. This is required to instruct the computer where to store the input value.

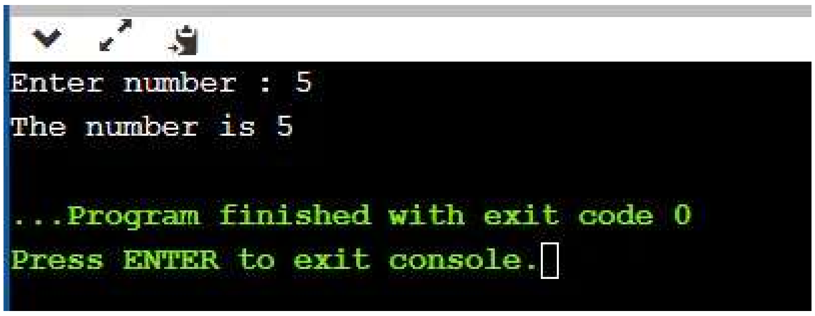
Now, for the printf() function. So say you want to output the value of an integer variable num on the screen. You will do so as

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

Things seem a little familiar this time. “%d” is again the same format specifier, this time denoting the data type of the value to output.Have a look at an example.



```
int num;
printf("Enter number : ");
scanf("%d",&num);
printf("The number is %d",num);
```



```
Enter number : 5
The number is 5

...Program finished with exit code 0
Press ENTER to exit console.
```

Carefully notice how we can also print text as well as combination of text and integers using the printf function.

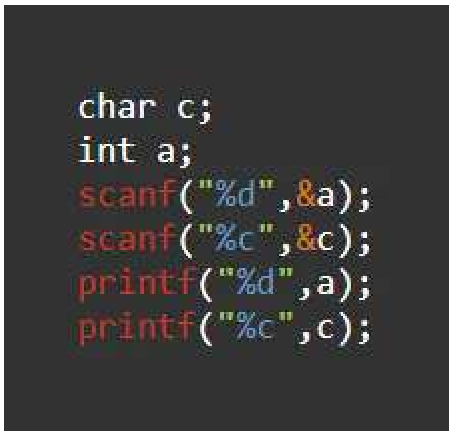
Now for the revisit of format specifiers as promised.

Format Specifiers

Format specifiers are an important part of input and output in the C language. They help the computer determine the data type of the input and output. The format starts with the '%' character. We have already discussed the format specifier for integer. We should also keep in mind the specifiers for the other data types we discussed.

```
int     %d
char    %c
float   %f
```

To know more about format specifiers click here.
Now for a very simple question

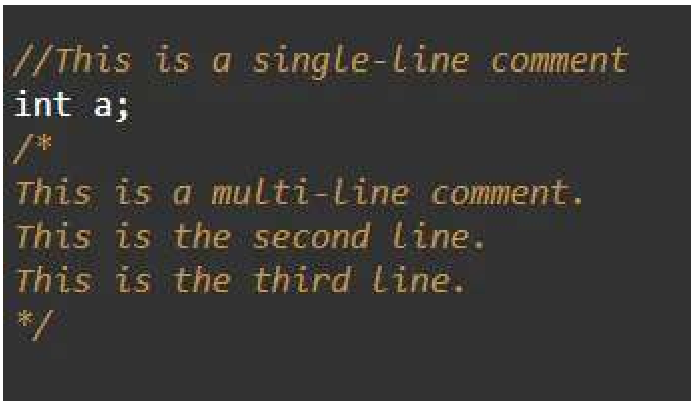


```
char c;
int a;
scanf("%d",&a);
scanf("%c",&c);
printf("%d",a);
printf("%c",c);
```

What should be the output for this code snippet if the user inputs 1 and “a”?

Comments

A lot of times when programmers revisit complex code this is their situation. To avoid this, we use comments ! Comments are text notes added to the program to provide information about the code and to remind programmers of the logic they used while writing the code.These are ignored while running the program. Let's have a look at how to add comments in our code.



```
//This is a single-line comment
int a;
/*
This is a multi-line comment.
This is the second line.
This is the third line.
*/
```

The first one is a single line comment. To write a multiple line comment we enclose it in /*, */

It is now time to move to the first major programming construct of this blog - Conditional Statements.