C PROGRAMMING LECTURE 1 TO 20

CODE WITH HARRY

What is C:

C is one of the fastest programming language as it is close to low-level languages such as Assembly language. If you are already familiar with programming, then you must know the demand out there for a C programmer. Because C comes in handy in case of all the applications that require fast processing, such as games, or major parts of different operating systems.

Benefits of C:

C was developed in early **1970's** but it is still in much demand and it doesn't seem it is going anywhere, there are a lot of reasons for that such as:

- C takes only significant CPU time for interpretation. That is why a lot Python libraries such as **NumPy**, **pandas**, **scikit-learn**, etc. are built using C.
- Being close to Machine language, some of its functions including direct access to machine level hardware APIs
- It is a highly portable language
- Data structure and algorithm can be better implemented in C language, as you can see in my Data Structures and Algorithms Course Playlist.
- C is being used to support other programming languages to enhance their efficiency.

Scope:

Let's talk about the scope of C, so you may know that you are not wasting for time with this Playlist. C has a wide range of scope

- Windows, Linux, Mac all are mostly written in C. **Linux is written 97% using C**.
- C has also been used widely while creating iOS and Android kernels.
- MySQL database is written using C.
- Ruby and Pearl are mostly written using C.
- Most part of Apache and NGINX is written using C.
- Embedded Systems are created using C

Control:

C gives most of the control in the hand of users. Things like memory allocation and manipulation are totally in the hands of the programmer. Being a flexible language, it provides more access to the programmer because of which it is more efficient.

All of these reasons make C one of the widely used Programming language having special respect in the heart of programmers.

Summary:

C is not a hard language to learn and along with that it stills has a scope in the future. It is a low-level programming language that makes it more efficient and swifter. So many devices and applications are build completely or partially using C, so there is a wide range of options related to field you want to get into.

LECTURE 02

What is coding?

If we look at the definition i.e. " coding is the ability to write computer programs". The definition explains nothing at all, so I am going to explain you in simple words. Coding is a way of communication with the computer. As we know that computer is a dumb machine and it can not work properly until it is provided with instructions. Instructions to the computer are provided using different languages in order for it to do specific tasks.

We have different options i.e. languages through which we can communicate through computer, some of which are low level, that are closer to computer's understanding i.e. Machine language or C and some are high level that required more processing time in order to transfer the commands or instructions to computer hardware i.e. Python.

Programming or coding is an act of telling the computer, what to do? Without it computer is useless. Programming Languages makes it easier for us to communicate to computer because they work as an intermediary language that both computers and humans can understand.

In 2nd phase we are briefly going to cover the history of C.

History:

C language was developed by **Dennis Ritchie in 1972**. At that time, he was working at bell laboratories of AT&T. AT&T is an American multinational conglomerate holding company. It wasn't the first programming language as before it B, BCPL were being used but there we a lot of problems related to those language and they weren't that much easier to use. After the introduction of C in the tech industry, it soon got fame without any promotion or advertisement of any sort and till now it is being used widely.

The third thing we will be discussing in this tutorial is our Course and its designed.

Course Content:

This is a unique course as this is specially designed keeping in mind the welfare of the viewer. This course is going to help you the most if you see it in sequence as all of the videos are designed in a way that will start you off from a beginner's level and take you to an expert level.

The starting topics are easy and simple such as data types, operators, if else statements, etc. but yet well explained as they are the basis of the language. Then we move on to the intermediate topics such as Strings, Arrays or pointers and at the end we will move towards the most advance topics such as File I/O or Function Pointers, etc.

There are also a lot of exercise included in these tutorials, so you might test your skills or learnings and can also evaluate your performance.

Summary:

This course is going to make you an expert level C programmer if you take it seriously and it is also going to help you with other languages as C++, which is very much like C. If you have learned C than you have learned about 60-70% C++. It will also help you with Python as moving towards a simpler language is more easier.

LECTURE 03

What is an IDE?

IDE stands for **Integrated development environment**. It is nothing more than an enhanced version of a text editor that helps you write more efficient and nicer code. It helps to differentiate different parts of your codes with different colors and notifying you if you are missing some semicolon or bracket at some place by highlighting that area. A lot of IDEs are available such as **DEVC++** or **Code Blocks** but we are going to use **VS Code** for this tutorial series.

Compiler:

A compiler is used to run the program of a certain language by converting the code into the language that our computer could understand. Without a compiler we can not run our code. Every programming language is required a different compiler for its functioning because the syntax of every language is different from the other. There are a lot of compilers available, but we are going to use **MinGW** for this course because it will fulfill all of our requirements and also it is recommended by Microsoft itself.

LECTURE 04

In this tutorial, our focus will be towards the structure of our written program. As a beginner the concept might be difficult for you to understand but do not worry as it happens to everyone, but with time you will develop a good grasp on it. We will be discussing some of the structural parts of our code, such as:

- Pre-processor commands
- Functions
- Variables
- Statements
- Expressions
- Comments

Now we will divide our written program into a few lines of code and one by one we will go over the meaning of each and every keyword in the specific line.

So, let's begin with **Pre-processor commands**.

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

Copy

This is the first line of our code. Any line starting with # represents a preprocessing command. It tells our program that before its execution, it must include the stdio.h named file in it because we are using some of the commands or codes from this file.

For example, if our programs needs mathematical operations of high level them we must include:

```
#include <math.h>
```

Copy

It helps us to use the code from math.h file for the calculations in our programs.

```
int main()
```

Copy

this is the 2nd line. main() is **function** here and we will see the detail about the function in later tutorials. Here int is the **return type** of function and the return type is according to the functions activity i.e. if it is giving an integer variable as a result then return type should be int.

```
int a, b;
```

Copy

here we are **initializing** two variables as integers. Initializing with integer means that we can store integer values in it. If we would have initialized them with char then we could have stored character values in it such as a, b, c, d, etc.

```
printf("Enter number a\n");
```

Copy

This is simply a **print statement**. Whatever we write in the brackets will be directly printed onto the screen. /n is the **new line character** here.

```
scanf("%d", &a);
```
Copy
 scanf is used to take **inputs from the user**. Here &a gives the address of variable “a” to store the user's given value. %d notifies that the value should be of integer type.

```
printf("The sum is %d\n", a+b);
```
Copy
Here a+b is simply a **mathematical addition** and print statement is printing the result onto the screen.

```
return 0;
```
Copy
we need a **return value** for a function. The function we created was of int type so it is returning 0. Return 0 means that the function is working perfectly.

Note: Return statement and function type should be same.

```
//This is a comment
```
Copy
We write **comments** by using the double slash sign (//). Comments are to notify other programmers the working of the code at specific intervals or we write them for our-self. They do not have any effect on the written program.

Now we are going to write a command in our power terminal window, down below:

```
  gcc-Wall-save-temps file_name.c -o new_file
```
Copy
Note: Here the **file_name** is the name of the file we created to write code and **new_file** is the name we want our executable file to have.

Now I will tell you the benefits of this command. By the help of this command five files will be created namely:

- file_name.i
- file_name.s
- file_name.o
- file_name.c
- new_file.exe

file_name.i will create a **preprocessing file** where comments are removed, macros are expended and all the code from # files are copied into the file_name.i file with our respective code at the end.

file_name.s will have our code converted in **assembly** level code

file_name.o will have all the code in **machine level language** in binary form.

file_name.c will contain our **executable C language code**.

new_file.exe is the linker that **links** all the file_name.o sort of files at on place.

A C program is made up of different **tokens** combined together. These tokens include:

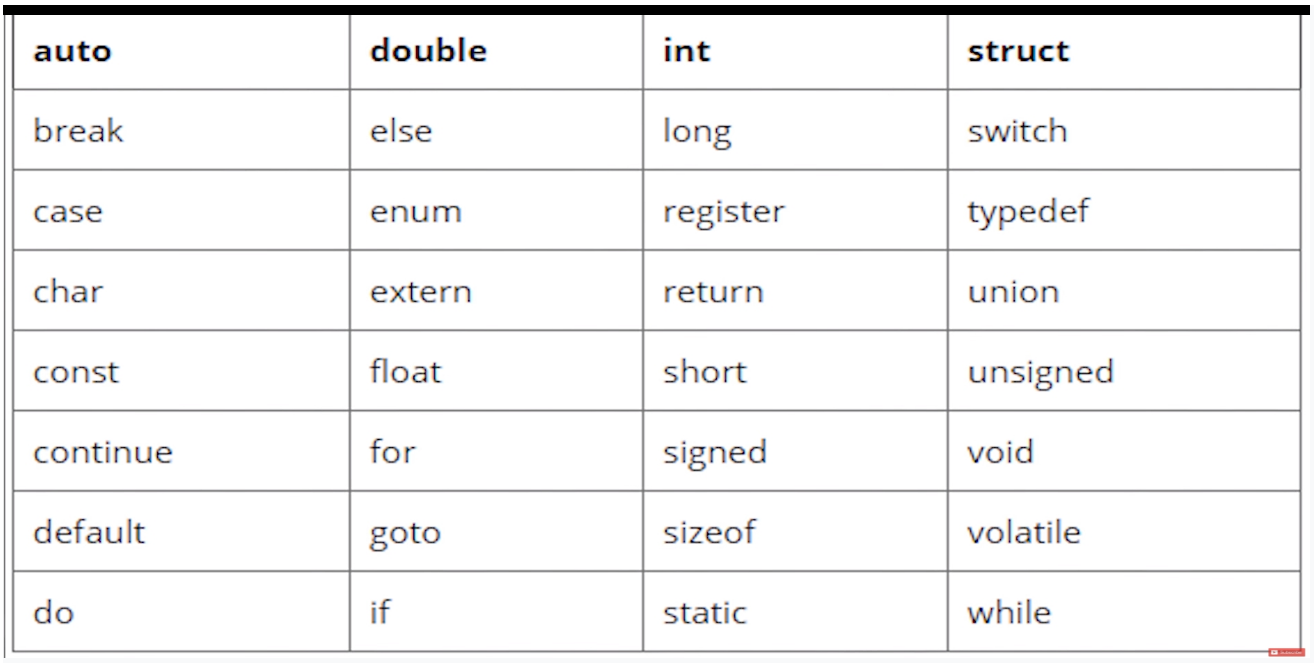
- Keywords
- Identifiers
- Constants
- String Literal
- Symbols

```
int a;
printf("Enter number a\n");
scanf("%d", &a)
return 0;
```

Copy
I have written a four-line code above so I can explain the tokens in a better way by using the references from the code above.

Keywords:

Keywords are reserved words that can not be used elsewhere in the program for naming a variable or function, instead they have a specific function or tasks and they are solely used for that. In the above given code, the return statement in the third line is a keyword.



auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Image: List of all the Keywords of C

Identifiers:

Identifiers are names given to variables or functions in order to differentiate them from one another. They are solely based on our choice but there are few rules that we have to follow while naming identifiers. According to the rules the name can not contain special symbols such as @, - , *, < , etc. In the above given code the "a" integer is an identifier.

Note: C is a case sensitive language so an identifier containing a capital letter and another one containing a small letter at the same place will be different. For example the three words: Code, code and cOde can be used as three different identifiers.

Constant:

Constant are very similar to variable and their values can be of any data type. The only difference between constant and variable is that a constant's value never changes. We will see constants in more detail in the upcoming tutorials. In the above given code the "0" in the last line is a constant.

String literal:

String literal or string constant is a line of characters enclosed by double quotes. In the above given code "Enter number a" is a string literal. printf is being used there to print string literal onto the screen.

Symbol:

Symbols are special characters reserved to perform certain actions. They are used to notify the compiler so they can perform specific tasks on the given data. In the above example code & is being used as a symbol.

Let's talk a little about **white space**. White space or blank space does not create any difference while using C. Unlike Python where we have to press enter to go to new line, in C we use semi-colon (;) to end a line of code. So until a semi colon arrives, the compiler will treat the code as a single liner so no matter how many lines we consume the code will run accurately if written correctly.

There are two code snippets given below. You can notice that they differ a lot regarding while space but their execution wills how the same output onto the screen i.e. **"Hello World"**.

Code1:

```
#include <stdio.h>


int main()
{
   printf("Hello World\n");
   return 0;
}
```

Copy

Code2:

```
#include <stdio.h>
int main(){
   Printf("Hello World\n")
;
   return 0;}
```

Variables & Data Types In C: C Tutorial In Hindi #6

So guys in this tutorial, we are going to learn about variables and go over various data types. This tutorial is mostly going to be theoretical, and we will only touch the code for the purpose of understanding, except for that we will not be performing any coding related work, as theory is what makes your basis strong and firm foundation can help you grasp the coding part more efficiently.

As we have already discussed in the previous tutorial while going through identifiers that variables are nothing more than simple names given to a specific space in memory for reservation. I will get into more detail about it with the help of an example but first, let us cover some basics.

Declaration:

We cannot declare a **variable** without specifying its **data type**. The data type of a variable depends on what we want to store in the variable and how much space we want it to hold. The syntax for declaring a variable is simple:

```
data_type  variable_name;
```

Copy
or

```
data_type  variable_name = value;
```

Copy
the data type can be int, float, char, depending on what kind of value we want to store.

Naming a Variable:

A variable name can be of anything we want to call out variable. Yet there are specific rules we must follow while naming a variable:

- A variable name can contain **alphabets**, **digits**, and **underscore** (-) only.

- The starting letter can not be a **digit**.
- **White spaces** cannot be used.
- The name should not be **reserved keyword or special character**.

We can declare and assign value to a variable in two ways.

1st way:

```
int a = 12;
```

Copy

2nd way:

```
int a;
a= 12;
```

Copy

Both of these have exactly the same working.

A variable as it names define can be altered, or its value can be changed, but same is not true for its type. If a variable is of integer type, then it will only store an integer value, which means that we cannot assign a character type value to an integer variable. We can not even store a decimal value into an integer variable.

Let's see this with an example:

Example 1:

```
#include <stdio.h>
int main(){
   int a = 12.2221;
   printf("Output = %d" , a);
   return 0;}
```

Copy

We are sending 12.2221 as a value in a, but since it is an integer type variable, the output will be only 12.

```
Output = 12
```

Copy

Example 2:

```
#include <stdio.h>
int main(){
   float a = 12.2221;
   printf("Output = %f" , a);
```

```
    return 0;}
```

Copy

Here we are using float as a data type. In this case, you can see the output below is 12.222100

```
Output = 12.222100
```

Copy

Note that we used %f instead of %d in case of float.

The reason is that int can store only **2 bytes** worth data as its storage capacity is 2 bytes while float storage capacity is **4 bytes**.

DATA TYPE	MEMORY (BYTES)	RANGE
Char	1	-128 to 127
signed char	1	-128 to 127
unsigned char	1	0 to 255
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 65,535
int	2	--32,768 to 32,767
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295
float	4	
double	8	
long double	10	

Code as described/written in the video

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


    printf("%lu",sizeof(int));
    return 0;}
```

Operators In C: C Tutorial In Hindi #7

Today we are going to learn about operators. I will teach you guys the theory related to operators as well as showing you the code as examples. So we will be using VS Code to write a few lines of codes for better understanding of the topic. Let’s start with the definition:

“Special symbols that are used to perform actions or operations are known as operators.”

For example, the symbol plus (+) is used to perform addition so it is an operator.

We will discuss all sorts of operator here. Let’s start with the simpler one’s i.e. Arithmetic.

Arithmetic operators:

Arithmetic operators are used to perform mathematical operations such as addition, subtraction etc. Few of the simple arithmetic operators are :

Operator	Description
+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Modulus

We all know their purpose and how they are used in simple mathematics. Their purpose and functionality are the same, let’s see their implementation in C.

```
int a = 2;
int b = 3;
printf("a + b = %d\n", a+b);
```

Copy

The output will be:

```
a + b = 5
```

Copy

Relational Operators:

Relational operators are used for the comparison between two or more numbers. Same as Java, C also has six relational operators and their return value is in Boolean i.e. either **True or False** (1 or 0).

Operator	Description
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Is equal to
!=	Is not equal to

Let's go to VS Code now:

```
int a = 2;
int b = 2;
printf("a == b = %d\n", a==b);
```

Copy

The output is 1 i.e. True.

If we change the value of a or b the value will be false or 0.

```
int a = 1;
int b = 2;
printf("a == b = %d\n", a==b);
```

Copy

The output is 0 i.e. False.

Logical Operators:

There are three logical operators i.e. AND, OR and NOT. They can be used to compare Boolean values but are mostly used to compare conditions to see whether they are satisfying or not.

AND: it returns true when both operators are true or 1.

OR: it returns true when either operator is true or 1.

Not: it is used to reverse the logical state of the operand.

Symbol	Operator
&&	AND operator
||	OR Operator
!	NOT Operator

Example:

```
int a = 1;

int b = 0;

printf("a or b = %d\n", a||b);
```
Copy

Here the output is:

```
a or b = 1
```
Copy

Let’s see what happens if both the values are zero

```
int a = 0;

int b = 0;

printf("a or b = %d\n", a||b);
```
Copy

the output is:

```
a or b = 0
```
Copy

Bitwise Operators:

To perform bit level operations, bitwise operators are used. They convert the values we provide to them in binary format and then compare them to provide us the results.

Symbols	Operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise complement
<<	Shift left
>>	Shift right

Assignment Operators:

Assignment operators are used to assign values. They are going to be used in each and every one of our program.

```
int a = 0;

int b = 1;
```
Copy

Equal to (=) is the assignment operator here, assigning 0 to a and 1 to b.

Operator	Description

=	Assigns values from right side operands to left side operand
+=	It adds the right operand to the left operand and assign the result to the left operand.
-=	It subtracts the right operand from the left operand and assigns the result to the left operand.
*=	It multiplies the right operand with the left operand and assigns the result to the left operand.
/=	It divides the left operand with the right operand and assigns the result to the left operand.

Conclusion:

These are few of the important operators that you should know about before starting actual programming. There are also many other operators such as &, % or *(pointer). I will let you know their details when working with them but the few defined above will be used frequently so knowledge about them is important. You do not have to remember them all as you can open the Tutorial any time again when required.

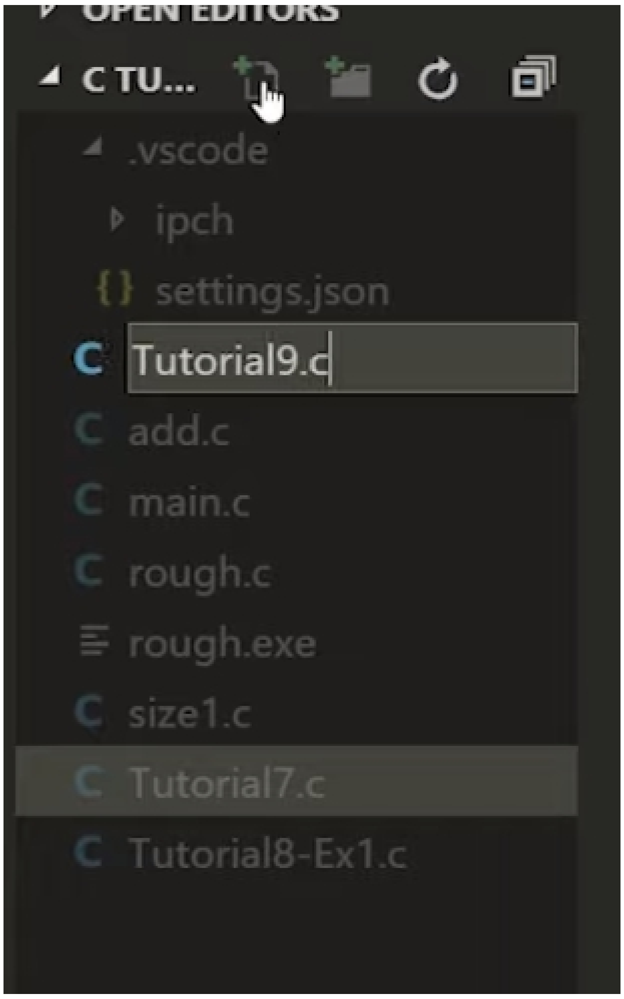
Code cTuts7.c as described/written in the video

```c
#include
int main(){
   /* code */
   int a, b;
   a = 2;
   b = 3;

   printf("a & b = %d\n", a&b);
   printf("a - b = %d\n", a-b);
   printf("a * b = %d\n", a*b);
   printf("a / b = %d\n", a/b);

   return 0;}
```

Format Specifiers and Escape Sequences With Examples : C Tutorial In Hindi #9

To run a **C program**, we need an IDE's like Visual Studio Code or Code blocks. For this series, we are using Virtual Studio Code (VS Code). **Visual Studio Code** is a fast source code editor and provides the tools that a developer needs for efficient programming. To download VS Code, click on Download Visual Studio Code . For guidance, check the tutorial Install & Configure VS Code

Now open VS Code and create a file with name **"tutorial9.c"**



In today's tutorial, we will learn about the format specifiers and escape characters as it is one of the most important concepts in C programming. Whenever we write a code in C, we have to use the format specifiers to define the variable type in input and output, and escape characters to format the output. So before exploring more C language concepts, first it will be better to have a strong grip on this concepts.

Format specifier in C:-

The format specifier in C programming is used for input and output purposes. Through this, we tell the compiler what type of variable we are using for input using scanf() or printing using printf(). Some examples are %d, %c, %f, etc.

The %c and %d used in the printf( ) are called format specifiers. **Format specifier** tells printf( ) to print the value, for %c, a character will printed, and for %d, a decimal will be printed.Here is a list of format specifiers.

Format Specifier	Type
%c	Used to print the character
%d	Used to print the signed integer
%f	Used to print the float values
%i	Used to print the unsigned integer
%l	Used to long integer
%lf	Used to print the double integer
%lu	Used to print the unsigned int or unsigned long integer
%s	Used to print the String
%u	Used to print the unsigned integer

Following are the few examples of format specifier:

- To print the integer value:

```
printf("\n The value of integer is %d", d);
```

Copy

- To print the float value:

```
printf("The value of float is %f", f);
```

Copy

- To print the character:

```
printf("\n The value of character is %c", c);
```

Copy

- To print the string:

```
printf("\n The value of string is %s", s);
```

Copy

What is Escape Sequence in C?

Many programming languages supports the concept of Escape Sequence. An escape sequence is a sequence of characters which are used in formatting the output. They are not displayed on the screen while printing. Each character has its own specific function like \t is used to insert a tab and \n is used to add newline.

Types of Escape Sequence in C

Escape Sequence	Description
\t	Inserts a tab
\b	Inserts a backspace
\n	Inserts a newline.
\r	Inserts a carriage return.
\f	Inserts a form feed.
\’	Inserts a single quote character.
\”	Inserts a double quote character.
\\	Inserts a backslash character.

Following are the some examples of escape sequence:

- Print character backslash(\) using printf function

```
printf("\n C programming \m/ ");
```
Copy

- Prints a newline before and after the text

```
printf("\n This is my C program\n"); ·
```
Copy

- Use \" to print double quote and \' for single quote

```
printf("\n Welcome to \"The C Programming tutorial\"");
```
Copy
```
printf("\n Welcome to \'C programming series \'") ;
```
Copy

- To provide tab space between two words

```
printf("Hello \t Viewers");
```
Copy

- To add vertical tab character.

```
printf("Hello Viewers");
```

Copy

```
printf("\v Welcome to C Programming");
```

Copy

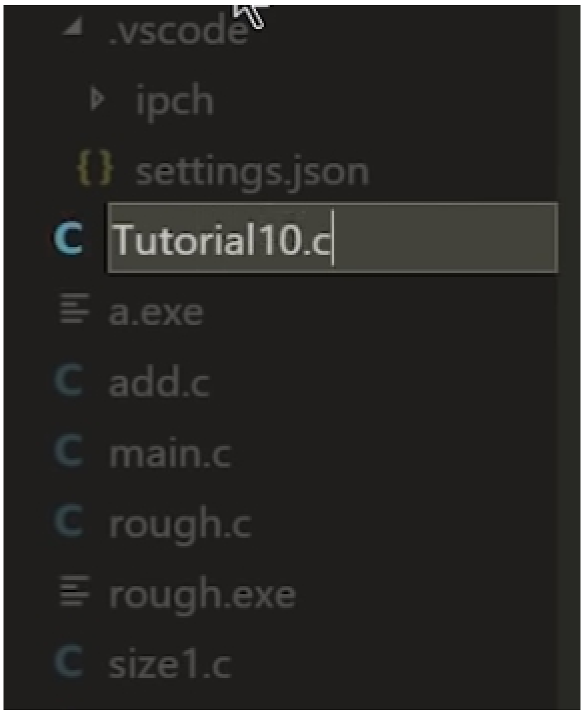
Code as described/written in the video

```
#include <stdio.h>#define PI 3.14/* this is a multiline comment



this is ignored by my compiler

*/int main(){
    int a = 8;
    const float b = 7.333;
    // PI = 7.33; //cannot do this since PI is a constant
    printf(" tab character \t\t my backslash  %f", PI);
    // b = 7.22; //cannot do this since b is a constant
    // printf("Hello World\n");
    // printf("The value of a is %d and the value of b is %2.4f\n", a, b);
    // printf("%18.4f this",b);

    return 0;}
```

If Else Control Statements In C: C Tutorial In Hindi #10

To run a **C program**, we need an IDE's like Visual Studio Code or Code blocks. For this series, we are using Virtual Studio Code (VS Code). **Visual Studio Code** is a fast source code editor and provides the tools that a developer needs for efficient programming. To download VS Code, click on Download Visual Studio Code . For guidance, check the tutorial Install & Configure VS Code

Now open VS Code and create a file with name **"tutorial10.c"**



Sometimes we want a set of instructions to be executed if certain condition is satisfied and an entirely different set of instructions to be executed if the condition does not fulfil. This kind of situation is dealt in C language using a **decision control instruction.** In today's tutorial, we will discuss about if-else statements in C programming.

Like other programming languages, C also uses the **if keyword** to implement the decision control instruction. The condition for the if statement is always enclosed within a pair of parentheses. If the condition is true, then the set of statements will execute. If the condition is not true then the statement will not execute, instead the program skips that part.

We express a condition for if statements using relational operators. The relational operators allow us to compare two values to see whether they are equal, unequal, greater than or less than.

Conditions	Meaning
a == b	a is equal to b
a != b	a is not equal to b
a < b	a is less than b
a> b	a is greater than b
a <= b	a is less than or equal to b
a >= b	a is greater than or equal to b

The if-else Statement:

The statement written in if block will execute when the expression following if evaluates to true. But when the if is written with else block, then when the condition written in if block turns to be false, then the set of statements in the else block will execute.

Following is the syntax of if-else statements:

```
if ( condition ){
statements;}
 else {
statements;}
```
Copy

Example:

Here is a program, which demonstrates the use of if-else statement.

```
#include <stdio.h>int main( ) { int num ;printf ( "Enter a number less than 10 " ); scanf ( "%d", &num );
 if ( num <= 10 ){  printf ( "Number is less than 10");}else{printf("Number is greater than 10");}return 0; }
```
Copy
On execution of this program, if you type a number less than or equal to 10, program will show a message on the screen through printf( ).

Nested If-Else Statements:-

We can write an entire if-else statement within either the body of the if statement or the body of an else statement. This is called 'nesting' of ifs. The Example of nested if-else statements is given below

```
main( ) { int   a;printf ( "Enter either 0 or 1 " ) ; scanf ( "%d", &a ) ;
 if ( a == 1 ){
 printf ( "Number 1 is entered!" ) ;  }else  { if ( a == 0 ){   printf ( "Number 0 is entered" ) ;}  else {  printf
( "Wrong Input" ) ;  }}return 0;}
```

Copy

Summary

In today’s tutorial, we learned about the if-else statements in C language. If-else statement is used when a program needs to take a certain decision. An if block does not always need to be associated with an else block. However, an else block will always be associated with an if statement. If the expression in if statement is evaluated to true, statements inside the body of if are executed. And when the test expression is evaluated to false, statements inside the body of if are not executed. If the else is associated with if block, then the statements written in else block will execute.

Code as described/written in the video

```
#include <stdio.h>
int main(){
   int age;
   printf("Enter your age\n");

   scanf("%d", &age);
   printf("You have entered %d as your age\n", age);
   if (age>=18) {
      printf("You can vote!");
   }

   else if(age>=10)
   {
      printf("You are between 10 to 18 and you can vote for kids");
   }

   else if(age>=3)
   {
      printf("You are between 3 to 10 and you can vote for babies");
   }

   else{
      printf("You cannot vote!");
   }



   return 0;}
```

```
// maths and science - 45// science - 15// maths - 15
// print the type of gift you are giving to them
```
Copy

Switch Case Control Statements In C: C Tutorial In Hindi #11

In this tutorial we are going to learn about control statements. C language provides us a special control statement that allows us to handle different cases effectively instead of using a series of if statements. This control instruction is the main focus of this tutorial.

The control statement that allows us to make a decision effectively from the number of choices is called a **switch**, or a **switch case-default** since these three keywords go together to make up the control statement. The expression in switch returns an integral value, which is then compared with the different cases. Switch executes that block of code, which matches the case value. If the value does not match with any of the cases, then **the default** block is executed. The syntax of the switch statement is:

```
switch ( integer expression ){  case value 1 :   do this ;
  case value 2 :
 do this ;
case value 3 :   do this ;
default :   do this ;
}
```
Copy

Explanation of the general form of Switch Statement:-

The expression following the switch can be an integer expression or a character expression. The case value 1, 2 are case labels that are used to identify each case individually. Remember that case labels should be different. If it is the same, it may create a problem while executing a program. At the end of the Case labels, we always use a colon (:). Each case is associated with a block. A **block** contains multiple statements that are grouped for a particular case.

Whenever the switch is executed, the value of test-expression is compared with all the cases present in switch statements. When the case is found, the block of statements associated with that particular case will execute. The break keyword indicates the end of a particular case. If we do not put the break in each case, then even though the specific case is executed, C's switch will continue to execute all the cases until the end is reached. The default case is optional. Whenever the expression's value is not matched with any of the cases inside the switch, then the default will be executed.

Example of Switch in C :-

```
#include <stdio.h>

 int main() {
   int i = 9;

   switch (i) {
     case 5:
     printf("Value is 7");
       break;

     case 0:
     printf("Value is 8");
       break;

     case 9:
     printf("Value is 9");
       break;

     default:
     printf("Value is not present");
       break;
   }
   return 0;
 }
```

Copy

Note -: It is not necessary to use break keyword after every case. When we do not want to terminate our case at that time, we will not use the break keyword.

Nested Switch in C:

We can also use nested switch statements i.e., switch inside another switch. Also, the case constants of the inner and outer switch may have common values without any conflicts.

the syntax of the nested switch is:

```
switch(expression 1)
{
 Case 1:
   printf("Switch Statement 1");
switch(expression 2){
 Case 1:
 printf("Switch Statement 2");
 Break;

 Case 2:
 Do this;}break;

 Case 2:
 Do this;}
```
Copy

Why do we need a Switch case?

There is one problem with the if statement: the program's complexity increases whenever the number of if statements increases. If we use multiple if-else statements in the program, the code might become difficult to read and comprehend. Sometimes it also even confuses the developer whom himself wrote the program. Using the switch statement is the solution of this problem.

Rules for Switch Statement -:

1. **The test expression of** Switch must be an int or char.
2. **The value of the** case should be an integer or character.
3. Cases should be inside the switch statement.
4. Using the break keyword in the switch statement is not necessary.
5. The case label values inside the switch should be unique.

Difference between a switch and if:

- Switch statements cannot evaluate float conditions, and the test expression can be an integer or a character, whereas if statements can evaluate float conditions.
- The switch statement cannot evaluate relational operators i.e., they are not allowed in switch statements, whereas if statements can evaluate relational operators. switch
- Cases in the switch can never have variable expressions; for example, we cannot write case a +3 :

Code as described/written in the video

```
#include <stdio.h>
int main(){
    int age, marks;
    printf("Enter your age\n");
    scanf("%d", &age);

    printf("Enter your marks\n");
    scanf("%d", &marks);

    switch (age)
    {
        case 3:
            printf("The age is 3\n");
            switch (marks)
            {
                case 45:
                    printf("Your marks are 45");
                    break;

                default:
                    printf("your marks are not 45");
            }
            break;

        case 13:
            printf("The age is 13\n");
            break;

        case 23:
            printf("The age is 23\n");
            break;
    default:
        printf("Age is not 3, 13 or 23\n");

  }

  return 0;}
```

Copy

← PreviousNext →

Loops In C: C Tutorial In Hindi #12

So guys in this tutorial, I would recommend you to focus mostly on the theoratical concept, as you can learn or copy the syntax from any where, but the usage of right loop at the right place is crucial. In programming, we frequently need to perform an action, of again and again, with variations in the details each time. The mechanism, which meets this need, is the 'loop,' and loops' concept is the tutorial's main focus. The versatility of the computer lies in its ability to perform the set of instructions repeatedly. This involves repeating some code in the program, either a specified number of times or until a particular condition is satisfied. Loop control instructions are used to perform this repetitive operation.

Following are three types of loop in C programming.

- For loop
- While loop

- do-while loop

There are two kinds of loops:

1. Entry Controlled loops: In entry controlled loops, the test condition is evaluated before entering the loop body. The For Loop and While Loop are an example of entry controlled loops.
2. Exit Controlled Loops: In exit controlled loops, the test condition is tested at the end of the loop. The loop body will execute at least once, whether the test condition is true or false. The do-while loop is an example of an exit controlled loop.

For Loop:-

A for loop is a repetition control structure that allows us to efficiently write a loop that will execute a specific number of times. The for loop working is as follows:

- The initialization statement is executed only once; in this statement, we initialize a variable to some value.
- In the second step, the test expression is evaluated. Suppose the test expression is evaluated to true. In that case, the for loop keeps running, and the test expression is re-evaluated, but if the test expression is evaluated to false, then the for loop terminates.

The loop keeps executing until the test expression is false. When the test expression is false, then the loop terminates.

while loop:-

While loop is called a pre-tested loop. The while loop allows code to be executed multiple times, depending upon a boolean condition that is given as a test expression. While studying for loop, we have seen that the number of iterations is known, whereas while loops are used in situations where we do not know the exact number of iterations of the loop. The while loop execution is terminated on the basis of the Boolean (true or false) test condition.

do-while loop:-

In do-while loops, the execution is terminated on the basis of the test condition. The main difference between the do-while loop and while loop is that, in the do-while loop, the condition is tested at the end of the loop body, whereas the other two loops are entry controlled loops.

Note: In do-while loop, the loop body will execute at least once irrespective of the test condition.

What about an Infinite Loop?

An infinite loop also known as an endless loop occurs when a condition always evaluates to true. Usually, this is considered an error.

Sometimes, while executing a loop, it becomes necessary to jump out of the loop. For this, we will use the break statement or continue statement.

- break statement

When a break statement is encountered inside a loop whether it is a for loop or a while loop, the loop is terminated and the program continues with the statement immediately following the loop.

- continue statement

Using a continue statement in the loop will cause the control to go directly to the test-condition and then it will continue the loop process.

Code as described/written in the video

Do While Loop In C: C Tutorial In Hindi #13

In the previous tutorial, we learned the basic concept of the loops in C. In today's tutorial, we will see the do-while loop in detail, along with an example. A do-while loop executes the statements inside the body of the do-while loop before checking the condition. So if a condition is false in the first place, then they do while would run once. A do-while loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

Unlike **for** and **while** loops, which test the loop condition first, then execute the code written inside the body of the loop, the **do-while** loop checks its condition at the end of the loop. Following is the syntax of the do-while loop in C programming.

```
do {
  statements );} while( test condition );
```

Copy

If the test condition returns true, the flow of control jumps back up to do, and the set of statements in the loop executes again. This process repeats until the given test condition becomes false.

How does the do-while loop work?

- First, the body of the do-while loop is executed once. Only then, the test condition is evaluated.
- If the test condition returns true, the set of instructions inside the body of the loop is executed again, and the test condition is evaluated.
- This looping process goes on until the test condition becomes false.
- If the test condition returns false, then the loop terminates.

Example:-

The following Program will add numbers until the user enters zero

```
#include <stdio.h>int main(){
   int n, sum = 0;
   do
   {  printf("Enter a number: ");scanf("%i", &n);
      sum += n;
   }
   while(n != 0);
   printf("Sum is = %d",sum);

   return 0;}
```

Copy

Output:-

```
Enter a number: 3
Enter a number: 4
Enter a number: 8
Enter a number: 0
Sum is = 15
```

Copy

}

As we have mentioned at the beginning of this tutorial, that do-while runs at least once even if the test condition returns false, because the test condition is evaluated after the execution of the instructions in the body of the loop.

WHAT IS THE DIFFERENCE BETWEEN WHILE AND DO-WHILE LOOPS IN C?

While loop is executed when given test condition return true, whereas, do-while loop is executed for the first time irrespective the test condition is true or false, because the test condition is checked after executing while loop for the first time.

This difference between while and do-while will be more clear by the following program.

```
main( ) {
 while ( 2 < 1 )   printf ( "Hello World \n") ;}
```

Copy

Here, since the condition fails the first time itself, the printf( ) statement will not get executed. Let's now write the same code using a do-while loop.

```
main( ) {  do {   printf ( "Hello World\n") ;  } while ( 2 < 1 ) ;}
```

Copy

In this program, the printf( ) statement would be executed once, since first the body of the loop is executed, and then the test condition is evaluated

Code as described/written in the video

```
#include
int main(){
   int num, index = 0;
   printf("Enter a number\n");
   scanf("%d", &num);
   do
   {
      printf("%d\n", index + 1);
      index = index + 1;
   } while (index < num);

   return 0;}
```

Copy

While Loop In C: C Tutorial In Hindi #14

Previously we learned the basic concept of loop and the working of do-while loop in detail. In this tutorial we are going to cover While loop in detail. A **Loop** executes the sequence of statements many times until the condition becomes false. In the previous tutorial, we learned the do-while loop. In this tutorial, we will learn while loop in C programming.

While loop is also called as a **pre-tested loop**. A while loop allows code to be executed multiple times, depending upon a given Boolean(true or false) condition. The while loop is mostly used in the case where the number of iterations is not known. If the number of iterations is known, then we use for loop.

The Syntax of while loop is:

```
while (condition test){// Set of statements}
```

Copy
The body of while loop can contain a single statement or a block of statements.
The **condition** may be any expression, and true is any nonzero value. The loop iterates while the test condition evaluates to true.

When the condition becomes false, the program control passes to the line immediately following the loop.

Example:-

```
#include<stdio.h>
 intmain(){
 inti=0;
 while(i<=5){
printf("%d \n",i);
i++;
```

7. }
8. return0;
9. }

Explanation of the above program:-

1. We have **initialized** a variable i with value 0. This code will print from 0 to 4; hence the variable is initialized with value 0.
2. In a while loop, we have provided a **condition** (i<=5), which means the loop will execute the body until the value of i becomes 5. After that, the loop will be terminated.
3. In the body of a loop, we have a print function to print our number and an **increment operation** (i++) to increment the value per execution of a loop. This process will continue until the value becomes 5 and then it will print the number and then terminate the loop.

Properties of while loop:

Following are some properties of while loop.

- A conditional expression written in the brackets of while is used to check the condition. The Set of statements defined inside the while loop will execute until the given condition returns **false**.
- The condition will return **0** if it is **true**. The condition will be false if it returns any nonzero number.
- In the while loop, we cannot execute the loop until we do not specify the condition expression.
- It is possible to execute a while loop without any statements. This will give no **error**.
- We can have multiple conditional expressions in a while loop.
- Braces are optional if the loop body contains only one statement.

Code as described/written in the video

```
#include<stdio.h>
int main(){
  int i = 0;
  while (i<54)
  {
    printf("%d\n", i);
    i = i+1;
  }

  return 0;}
```

Copy

For Loop In C: C Tutorial In Hindi #15

Loop is one of the most important concepts in all programming languages as it simplifies complex problems and makes it easier to read and understand the code. Imagine a situation where you would have to print numbers from 1 to 1000. What would you do? Will you type in the printf() statement a thousand times? Using a **for loop**, we can perform this action in three statements. So in today's tutorial, we are going to study about for-loop in detail

The **"For" Loop** is used to repeat a specific code until a specific condition is satisfied. The for-loop statement is very specialized. We use for a loop when we know the number of iterations we want, whereas when we do not know about the number of iterations, we use while loop. Here is the syntax of the for loop in C programming.

The syntax of the for loop is:

```
for ( initialize counter ; test counter ; increment/decrement counter)
{
//set of statements
}
```
Copy

- **initialize counter**: It will initialize the loop counter value, i.e., i=0.
- **test counter**: It verifies whether the condition is true.
- **Increment/decrement counter**: Incrementing or decrementing the counter.
- **Set of statements**: Execute the set of statements.

Example:-

```
#include <stdio.h>
int main(){
   int num = 10;
   int i;
   for(i = 0; i < num; i++) {
   printf("%d ",i);}
   return 0;}
```
Copy

Output:-

```
0 1 2 3 4 5 6 7 8 9
```
Copy

Explanation of the above code:-

First, the initialization expression will initialize loop variables. The expression **i=0** executes once when the loop starts. Then the condition **i < num** is checked if it is **true**, then the statements inside the body of the loop are executed. After executing the statements inside the body, the control of the program is transferred to increment the variable by **1 (i++)**. The expression **i++** modifies the loop variables. Then the condition **i<num** is evaluated again. If the condition is still true, the body of the loop will execute once more. The for loop terminates when **i < num** becomes **false**.

Just as if statement, we can have for loop inside another for a loop. This is known as **nested for loop**. Similarly, while loop and do while loop can also be nested.

```
for ( initialization; test condition; increment ) {
   for ( initialization; test condition; increment ) {
    // set of statements
   }
   // set of statements}
```
Copy

Note: there is no rule that a loop must be nested inside its own type. For loop can be nested inside the while loop and vice versa.

Why we prefer For Loops?

It is clear to a developer exactly how many times the loop will execute. So, if the developer has to dry run the code, it will become easier. Also, the Syntax of the for loop is almost the same as other programming languages like C++ and Java.

Code as described/written in the video

```
#include <stdio.h>
int main(){
    printf("Hello World\n");
    int i, j=0;
    for(i=0 ;  ; )
    {
      printf("%d %d\n", i, j);
      i++;j++;
    }


    return 0;}
```

Copy

Break and Continue Statements In C: C Tutorial In Hindi #16

Break Statement :

- Break statement is used to break the loop or switch case statements execution and brings the control to the next block of code after loop or switch case.
- Break statements are used to bring the program control out of the loop.
- The break statement is used inside loops or switch statement in C Language.

Syntax for Break statement :

```
#include<stdio.h> int main() {
          int i,age;
          for(i = 0 ; i < 5 ; i++) \
          {


                    printf("Iteration time = %d\nEnter Your Age : ",i );
                    scanf("%d",&age);
```

```
                if (age>10)
                {
                        break; // Checking Break Statement
                }
                // if(age<10)
                // { continue; }
                // printf("Hey Guys\n");
                // printf("This code is printed coz if condition is not satisfied. \n");
                // printf("Checking Continue Statement\n\n"); // Checking Continue Statement
        } return 0; }
```
Copy

Continue Statement :

- The **continue statement** is used inside loops in C Language. When a continue statement is encountered inside the loop, control jumps to the beginning of the loop for next iteration, skipping the execution of statements inside the body of loop after continue statement.
- It is used to bring the control to the next iteration of the loop.
- The continue statement skips some code inside the loop and continues with the next iteration.
- It is mainly used for a condition so that we can skip some lines of code for a particular condition.
- It forces next iteration in loop i.e. as break terminates the loop but continue forces the next iteration of the loop.

Syntax for Continue statement :

```
#include<stdio.h> int main() {
        int i,age;
        for(i = 0 ; <  ++)
        {
                printf("Iteration time %d\nEnter Your Age : ",i );
                scan“"%d",& );
                // if (age>10)
                // {
                // break; // Checking Break Statement
                // }
                if(age<10)
                { continue; }
                printf("Hey Guys\n");
```

```
                        printf("This code is printed coz if condition is not satisfied. \n");
                        printf("Checking Continue Statement\n\n"); // Checking Continue Statement
            }
            return 0; }
```
Copy

That's all about Break and Continue Statements in C Language.

Code as described/written in the video

```
#include <stdio.h>
int main(){
   printf("Hello World\n");
   int i, age;
   for (i=0; i<10; i++){
      printf("%d\nEnter your age\n", i);
      scanf("%d", &age);
      // if (age>10)
      // {
      //     break;
      // }
      if (age>10)
      {
         continue;
      }
      printf("we have not come accross any continue statements");
      printf("we have not come accross any continue statements");
      printf("we have not come accross any continue statements");
      printf("we have not come accross any continue statements");
      printf("Harry is a good boy");


   }


   return 0;}
```
Copy

Goto Statement In C: C Tutorial In Hindi #17

- A goto statement in C programming language provides an unconditional jump from the ‘goto’ to a labeled statement in the same function.

NOTE – Use of **goto** statements is highly discouraged or avoided in any programming language because it makes difficult to trace the control flow of a program to fellow programmers, making the program hard to understand and hard to modify or manipulate. Any program which uses goto can be modified to avoid goto statements.

- These are also called ‘Jump Statement’.
- It is used to transfer the control to a predefined label.
- It’s use is avoided since it causes confusion for the fellow programmers in understanding code.
- goto statement is preferable when we need to break multiple loops using a single statement at the same time.

Syntax for goto statement :

```
#include <stdio.h> int main() {
        int x;
        for (int i = 0;i <5; i++)
        {
                printf("\nHey Guys\n\n");
                for (int j = ;j < 3; j++)
                {
                        printf("Type any No. & To Exit : Press 1\n");
                        scanf("%d", &x);
                        if (x == 1)
                        {
                                goto end; // This goto will transfer the control to end: i.e. out of both
loop
                        }
                }
        }
        end:
        printf("\'For\' loops are skipped as you pressed 1");
        return 0; }
```
Copy

That's all about goto statement in C Language.

Code as described/written in the video

```
#include <stdio.h>
int main(){
  // label:
  //     printf("we are inside label");
  //     goto end;
  // printf("Hello World\n");
  // goto label;
  // end:
  //     printf("we are at end");
  int num;
  for(int i = 0; i < 8; i++)
  {
    printf("%d\n", i);
    for(int j = 0; j < 8; j++)
    {
      printf("Enter the number. enter 0 to exit\n");
      scanf("%d", &num);
      if(num==0){
        goto end;
      }
    }

  }
  end:


  return 0;}
```
Copy

Typecasting In C: C Tutorial In Hindi #18

Typecasting is a way to convert one data type into another one. It is also known as data conversion or type conversion.

C Language Provides two methods of type casting :

1. Implicit type casting
2. Explicit type casting

Implicit type casting : Implicit type casting means conversion of data types from one to another without losing its original meaning or sense.

- In this type casting program automatically converts the variable from one data type to other. It follows strict rules in converting data type of variables as it always converts lower data types to higher ones.

```
#include<stdio.h> int main() {
        short a=10; //initializing variable of short data type
        int b; //declaring int variable 'b'.
        b=a; //Implicit type casting
        printf("%d\n",a);
        printf("%d\n",b); }
```
Copy

Explicit type casting : Explicit type casting means conversion of data type from one to another forcefully by programmer. It is user defined conversion.

```
#include<stdio.h> int main() {
        int a;
        float b;
        char c;
        printf("Enter the value of a\n");
        scanf("%d",&a);
        printf("A is %d\n", a);
        printf("Enter the value of b\n");
        scanf("%f",&b );
        printf("B is %d\n", (int) b);
        printf("Type any character : \n");
        scanf(" %c",&c );
        printf("Character is %d",(int) c);
        return 0; }
```
Copy

That's all about TypeCasting in C Language.

Code as described/written in the video

```
#include <stdio.h>// Typecasting Syntax// (type) value;int main(){


        int a = 3;
        float n= (float)54/5;
        printf("The Value of a is %f\n", b );
        return 0;
}
```
Copy

Functions In C: C Tutorial In Hindi #19

Functions :

- Functions are used to divide a large C program into smaller pieces.
- Function can be called multiple or several times to provide reusability and modularity to the C program.
- Functions are also called as procedure or subroutines.
- It is a piece of code to accomplish certain operation.

Now let’s understand in simple terms what function is?

Function is nothing but a group of code put together and given a name and it can be called anytime without writing the whole code again and again in a program.

I know its syntax is bit difficult to understand but don’t worry after reading this whole information about Functions you will know each and every term or thing related to Functions.

Advantages of Functions :

- We can avoid rewriting same logic or code through functions.
- We can divide the work among programmers using functions.
- We can easily debug or can find bugs in any program using functions.

Function Aspects :

There are 3 aspects of function :-

1. Declaration
2. Definition
3. Call

- A function is declared to tell the compiler about its existence.
- A function is defined to get some task done. (It means when we define function we write whole code of that function. In this actual implementation of function is done.)
- A function is called in order to be used.

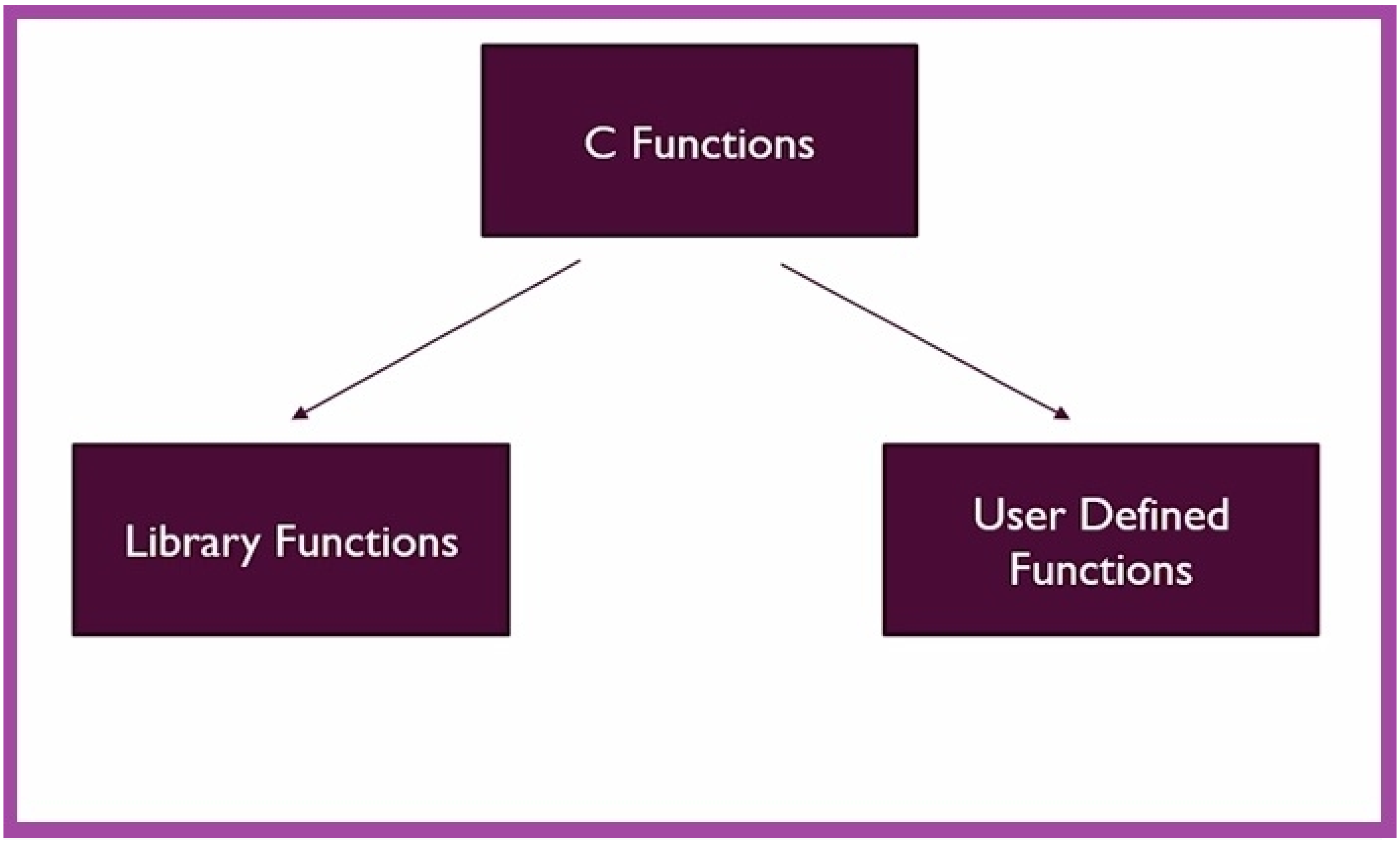
Types of Functions :

- Library Functions – These are pre-defined functions in C Language. These are the functions which are included in C header files.

E.g. printf(), scanf() etc.

- User defined Functions – Functions created by programmer to reduce complexity of a program i.e. these are the functions which are created by user or programmer.

E.g. Any function created by programmer.



NOTE -: Every C Program must contain one function i.e. main() function as execution of every C program starts from main() function.

Ways to define a Function :

There are 4 ways in which we can define any C Function and these are :

- Without arguments and without return value.
- Without arguments and with return value.
- With arguments and without return value.
- With arguments and with return value.

1.Without arguments and without return value : In this function we don't have to give any value to the function (argument/parameters) and even don't have to take any value from it in return.

```
//No Argument and No Return Value void Star_pattern() {
        int a;
        printf("Enter how many stars(*) you want : \n"); /* In this both declaration and definition of
function are done together.*/
        scanf("%d", &a );
        for (int i = 0; i<a; i++)
        {
                printf("*");
        } }
```

Copy

As you can see in above example the function with name ‘Star_pattern’ is not taking any argument (Since it’s parentheses are empty) and even it is not returning any value as we haven’t even used return keyword for returning something and in return_type we have written ‘void’ which means nothing.

So, with this example you can understand about functions which do not take any argument or value and even don’t return anything.

2.Without arguments and with return value : In these type of functions we don’t have to give any value or argument to function but in return we get something from it i.e. some value.

```
// Without arguments and with return value :int Sum(); /* Declaration of Function */

 /*Other Code*/ int Sum() {

        int x,y,z;
        printf("Enter no. 1 : \t");
        scanf("%d",&x);
        printf("\nEnter no. 2 : \t");
        scanf("%d",&y );
        z=x+y;
        return z; }
```

Copy

In above example you can see that our function with name ‘Sum’ is not taking any value or argument (Since parentheses are empty) but it is returning an integer value. So, that’s how we can use functions which don’t take any argument but returns something.

3.With arguments and without return value : In these type of function we give arguments or parameters to function but we don't get any value from it in return.

```
// With arguments and without return value :
 void Product(int x ,int y ); /* Declaration of Function */

                                       // Other Code Product(5,4) /* Calling Product Function in main() */

void Product(int a,int b) {
            int Multiplication;
            Multiplication = a*b ; /* Definition of Function */
            printf("The Product is %d\n\n",Multiplication); }
```
Copy

In this example as you can see we have declared function globally i.e. this function can be used anywhere in our program not just only in main() function. After that we have defined it to tell what this function will do and in this function as it will take some value or argument, it will process that data and will give some output but it will not return anything.

When we pass arguments in function at the time of calling it at that time only the value of variables get copied to variable of that function means only values are passed. It means any modification which is done on passed values do not affect the actual values of the variables in main() function. And this call is known as **Call by Value**.

4.With arguments and with return value : In these functions we give arguments to function and in return we also get some value from it.

```
// With arguments and with return value : float Percentage(int x,int y); // Declaration of Function /*Code*/
int x;
x = Percentage(80,95); // Calling Function
 /*Code*/ float Percentage(int x,int y) {
            float perct;
            perct = ((x+y)/200.0)*100.0; // Definition of Function
            return perct; }
```
Copy

So, this is the widely used method to define function in this way we give arguments to function and also get some value in return from it.

So, that's all about Functions in C Language.

Code as described/written in the video

```
#include <stdio.h>int sum(int a, int b);void printstar(int n){
   for(int i = 0; i < n; i++)
   {
       printf("%c", '*');
   } }

int takenumber(){
   int i;
   printf("Enter a number");
   scanf("%d", &i);
   return i;}int main(){
   int a, b, c;
   a = 9;
   b = 87;
   // c = sum(a, b);
   // printstar(7);
   c = takenumber();
   // printf("The sum is %d \n", c);
   printf("The number entered is %d \n", c);
   return 0;}

int sum(int a, int b){
   return a + b;}
```

Copy

Question : Print Multiplication table of any number entered by user

Solution :

There are two methods or ways to complete this task :

1. Simple printf command method
2. Loops method

We can simply use print commands and can print the table easily. And the other way is that we can use loops i.e. repetition of block of code to accomplish our task.

Method 1 : By Using Simple print commands :

```
#include<stdio.h>int main(){
	//method 1: Using Print Commands

	int table;
	printf("\nEnter the number you want multiplication table of : ");
	scanf("%d", &table);

	printf("\nMultiplication table of %d is :", table);

	printf("\n\n%d*1 = %d\n, table, table*1");
	printf("%d*2 = %d\n, table, table*2");
	printf("%d*3 = %d\n, table, table*3");
	printf("%d*4 = %d\n, table, table*4");
	printf("%d*5 = %d\n, table, table*5");
	printf("%d*6 = %d\n, table, table*6");
	printf("%d*7 = %d\n, table, table*7");
	printf("%d*8 = %d\n, table, table*8");
	printf("%d*9 = %d\n, table, table*9");
	printf("%d*10 = %d\n, table, table*10");

	return 0;}
```
Copy

Method 2 : By using loops :

```
#include<stdio.h>int main(){
	//method 2: Using Loops

	int table;
```

```
        printf("\nEnter the number you want multiplication table of : ");
        scanf("%d", &table);

        printf("Multiplication table of %d is as follows:\n\n", table);

        for (int i =1; i<=10, i++)
        {
                printf("%d*%d = %d\n", table, i, table*i );
        }



        return 0;}
```

Copy

This is how we can solve this problem.

Code as described/written in the video

```
#include <stdio.h>
int main(){
  int num;
  printf("Enter the number you want multiplication table of:\n");
  scanf("%d", &num);

  printf("Multiplication table of %d is as follows:\n", num);

  // printf("%d X 1 = %d\n", num, num*1);
  // printf("%d X 2 = %d\n", num, num*2);
  // printf("%d X 3 = %d\n", num, num*3);
  // printf("%d X 4 = %d\n", num, num*4);
  // printf("%d X 5 = %d\n", num, num*5);
  // printf("%d X 6 = %d\n", num, num*6);
  for (int i = 1; i < 11; i++)
  {
    printf("%d X %d = %d\n", num, i, num * i);
  }

  return 0;}
```