**Basic Structure of C Program #4**

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.

**Code add.c as described/written in the video**

#include <stdio.h>

int main()

{

int a, b;

printf("Enter number a\n");

scanf("%d", &a);

printf("Enter number b\n");

scanf("%d", &b);

printf("The sum is %d\n", a+b);

return 0;

}

Copy

**Code main.c as described/written in the video**

#include <stdio.h>

int main()

{

printf("Hello World\n");

return 0;

}

**Basic Syntax Of a C Program #5**

In this tutorial we are going to understand the basic syntax of a C program. We will be using the same program we have been using for previous two programs, so you may develop a better understanding about the syntax before proceeding further. So, let’s get started with our tutorial.

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)

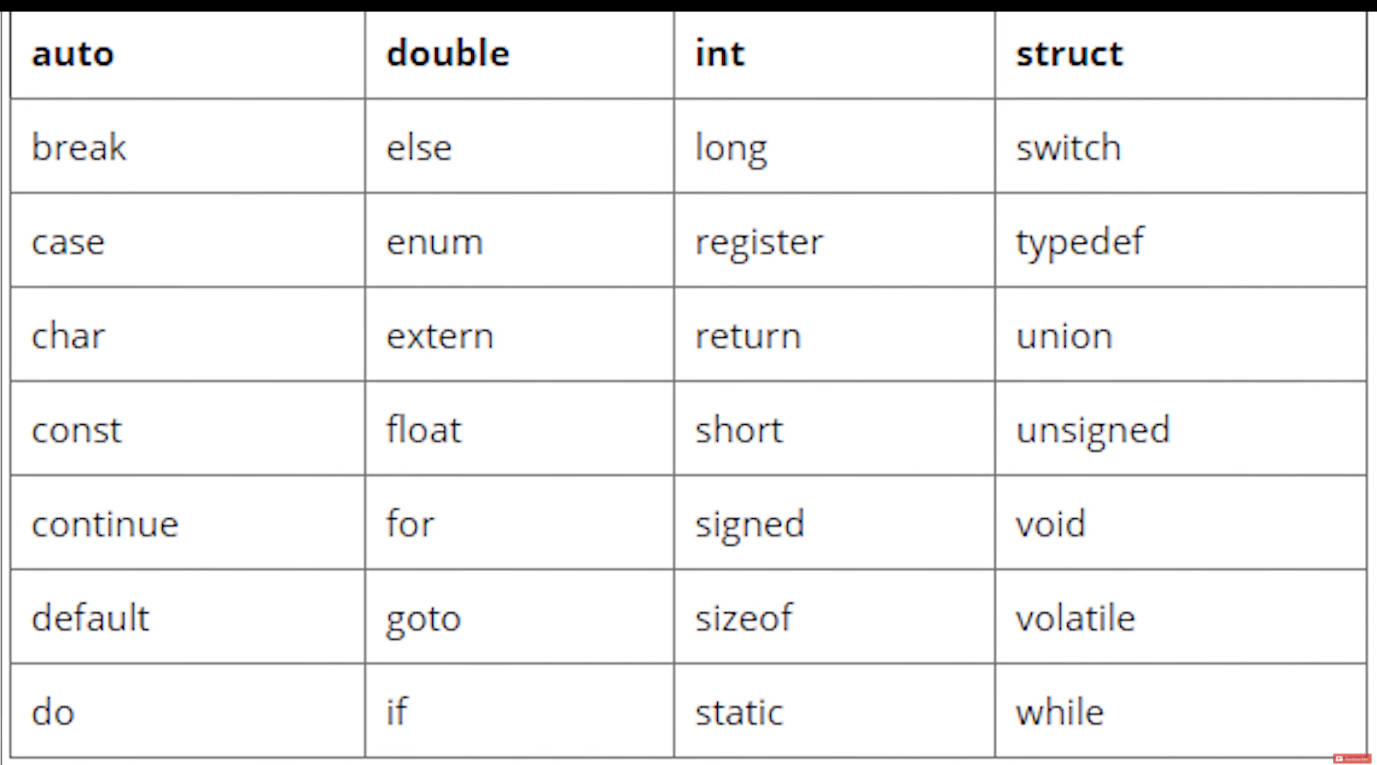
return 0;

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.

******

***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 #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 #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.

**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 |

**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 |

**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.

|  |  |
| --- | --- |
| **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**

#include <stdio.h>

int main()

{

    /\* code \*/

    int a, b, d, e;

    float c;

    a = 34;

    b = 6;

    c = 6.34;

    d = 1;

    e = 1;

    // int input

    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\n", a/b);

    // float

    printf("a + c = %f\n", a+c);

    printf("a - c = %f\n", a-c);

    printf("a \* c = %f\n", a\*c);

    printf("a / c = %f\n\n", a/c);

    // Relational operators

        // output=1-true, output=0-false

    printf("a == b = %d\n", a==b);

    printf("a != b = %d\n", a==b);

    printf("a >= b = %d\n\n", a==b);

    // Logical operators

        // output=1-true, output=0-false

    printf("d &&(and) e = %d\n", d && e);

    printf("d ||(or) e = %d\n", d || e);

    printf(" !(not) e = %d\n\n",  !e);

    return 0;

}

Output:

a + b = 40

a - b = 28

a \* b = 204

a / b = 5

a + c = 40.340000

a - c = 27.660000

a \* c = 215.560005

a / c = 5.362776

a == b = 0

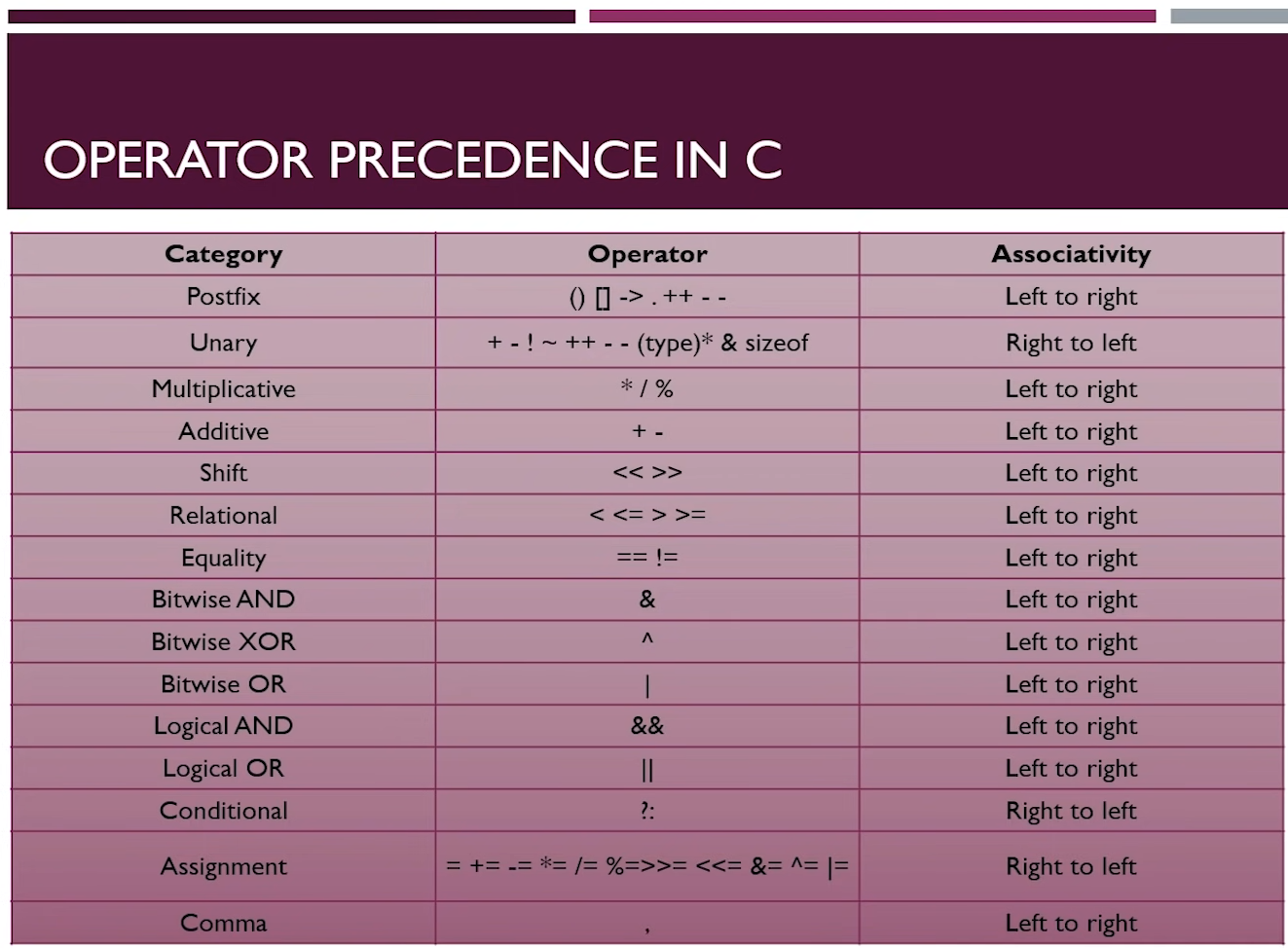
a != b = 0

a >= b = 0

d &&(and) e = 1

d ||(or) e = 1

!(not) e = 0



# Exercise 1 - Multiplication Tables #8

This is the first exercise of C programming tutorials. This C programming series contains many exercises that will help you to become a great problem solver. So, your first task is to **“Print Multiplication Table in C”.** All of the concept we have studied till now will use in this task, so by solving this exercise you can examine yourself that how much effort you need to put in learning C language.

#### Instructions:-

Take an input (in integer form) from the user and print its multiplication table on the screen. Following is the example of the output.

#### Input:

Enter the number you want multiplication table of: 6

#### Output:

Table of 6.

6\*1 = 6

6\*2 = 12

6\*3 = 18

6\*4 = 24

6\*5 = 30

6\*6 = 36

6\*7 = 42

6\*8 = 48

6\*9 = 54

6\*10 = 60

Code:

int main()

{

    int a;

    printf("Enter number a : ");

    scanf("%d", &a);

    printf("%d X 1 = %d\n", a , a\*1);

    printf("%d X 2 = %d\n", a , a\*2);

    printf("%d X 3 = %d\n", a , a\*3);

    printf("%d X 4 = %d\n", a , a\*4);

    printf("%d X 5 = %d\n", a , a\*5);

    printf("%d X 6 = %d\n", a , a\*6);

    printf("%d X 7 = %d\n", a , a\*7);

    printf("%d X 8 = %d\n", a , a\*8);

    printf("%d X 9 = %d\n", a , a\*9);

    printf("%d X 10 = %d\n", a , a\*10);

    return 0;

}

**C Format Specifiers and Escape Sequences With Examples #9**

**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);

* To print the float value:

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

* To print the character:

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

* To print the string:

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

**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. |

#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\n", 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;

}