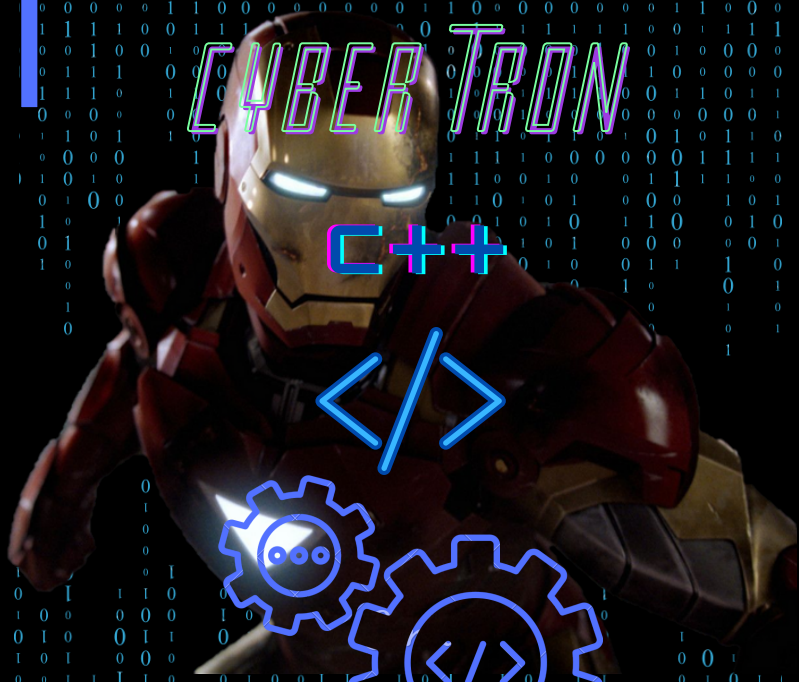
2023

C++

BASIC

CYBER TRON

C++



(FIRST PART)

T.I.M. HAMIM

Index:

1. Introduction to C++

What is C++ :

- C++ is known to be a very powerful computer programming language.
- C++ is a general purpose, case-sensitive, object oriented programming language.

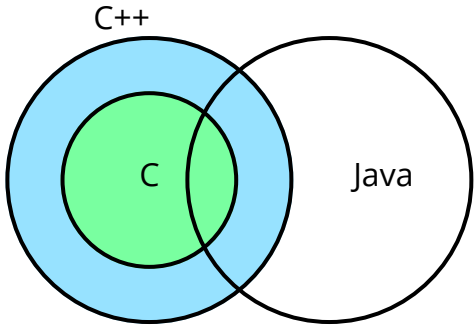
Usage of C++ :

- It is used to develop game engines, games and desktop apps, art applications, music players etc.
- C++ is being highly used to write device drivers and other software.

History of C++ :

- C++ programming language was developed in 1980 by Bjarne Stroustrup at bell laboratories of AT&T.
- Bjarne Stroustrup is known as the founder of C++ language.
- C++ was derived from C and is largely based on it.

Overlap of C, C++ and Java



Features of C++ :

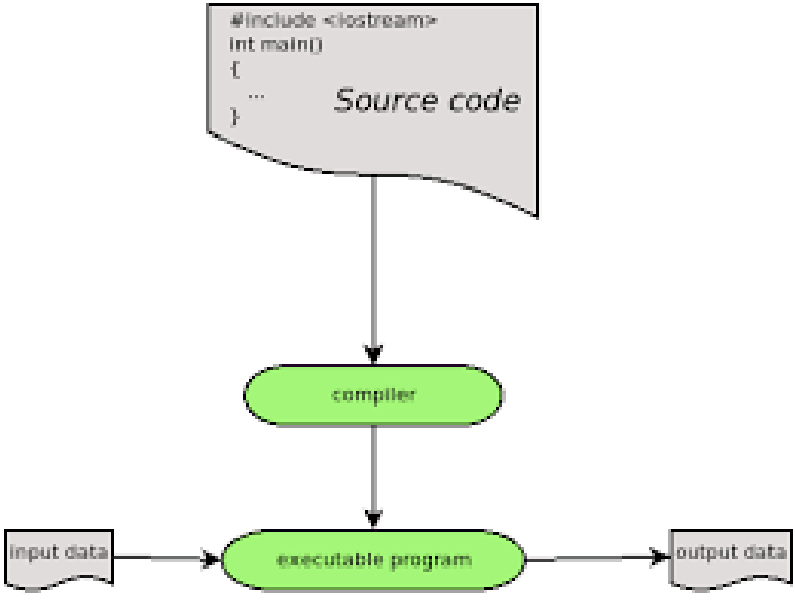
- Simple
- Mid-level programming language
- Rich Library
- Memory Management
- Fast Speed
- Pointers
- Recursion
- Object Oriented
- Compiler based

Compiler vs Interpreter :

Compiler and Interpreter are two different ways to translate a program from programming or scripting language to machine language.

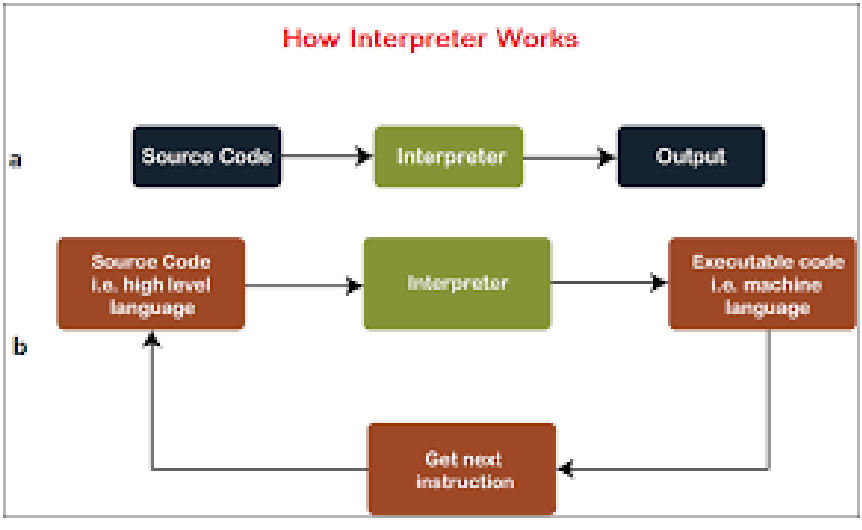
Compiler :

A compiler takes entire program and converts it into object code which is typically stored in a file. The object code is also referred as binary code and can be directly executed by the machine after linking. Examples of compiled programming languages are C and C++.



Interpreter :

An Interpreter directly executes instructions written in a programming or scripting language without previously converting them to an object code or machine code. Examples of interpreted languages are Perl, Python and Matlab.



Differences between Interpreter and Compiler:

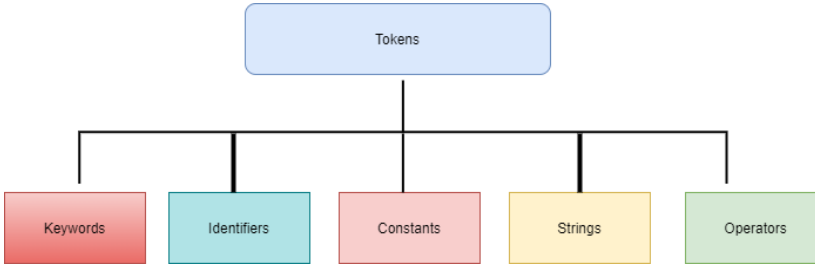
• Interpreter translates just one statement of the program at a time into machine code.	• Compiler scans the entire program and translates the whole of it into machine code at once.
• An interpreter takes very less time to analyze the source code. However, the overall time to execute the process is much slower.	• A compiler takes a lot of time to analyze the source code. However, the overall time taken to execute the process is much faster.
• An interpreter does not generate an intermediary code. Hence, an interpreter is highly efficient in terms of its memory.	• A compiler always generates an intermediary object code. It will need further linking. Hence more memory is needed.
• Keeps translating the program continuously till the first error is confronted. If any error is spotted, it stops working and hence debugging becomes easy.	• A compiler generates the error message only after it scans the complete program and hence debugging is relatively harder while working with a compiler.
• Interpreters are used by programming languages like Ruby and Python for example.	• Compliers are used by programming languages like C and C++ for example.

C++ is a powerful language. In C++, we can write structured programs and object-oriented programs also. C++ is a superset of C and therefore most constructs of C are legal in C++ with their meaning unchanged. However, there are some exceptions and additions.

Token :

When the compiler is processing the source code of a C++ program, each group of characters separated by white space is called a token. Tokens are the smallest individual units in a program. A C++ program is written using tokens. It has the following tokens:

- Keywords
- Identifiers
- Constants
- Strings
- Operators



Keywords :

Keywords(also known as reserved words) have special meanings to the C++ compiler and are always written or typed in short(lower) cases. Keywords are words that the language uses for a special purpose, such as void, int, public, etc. It can't be used for a variable name or function name or any other identifiers. Below is the table for some commonly used C++ keywords.

C++ Keyword			
asm	double	new	switch
auto	else	operator	template
break	enum	private	this
case	extern	protected	throw
catch	float	public	try
char	for	register	typedef
class	friend	return	union
const	goto	short	unsigned
continue	if	signed	virtual
default	inline	sizeof	void
delete	int	static	volatile
do	long	struct	while

Escape Sequences :

In C++ programming language, there are 256 numbers of characters in character set. The entire character set is divided into 2 parts i.e. the ASCII characters set and the extended ASCII characters set. But apart from that, some other characters are also there which are not the part of any characters set, known as ESCAPE characters.

It is a specilal character followed by backslash.

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form feed
\n	New line
\t	Tab (Horizontal)
\v	Tab (Vertical)
\r	Carriage return
\'	Single quote
\"	Double quote
****	Backslash
\?	Question Mark
\ooo	Octal number
\xhh	Hexadecimal number

2. Variables in C++

Local & Global variable:

Local variable:

A variable that is declared and used inside the function or block is called a local variable. It is scope is limited to function or block. It cannot be used outside the block. Local variables need to be initialized before use.

Global variable:

A variable that is declared outside the function or block is called a global variable. It is declared at the start of the program. It is available for all functions.

- **What will be the output of the program bellowed?**

```
#include <iostream>
using namespace std;
int x; //global variable
int main()
{
   int y; // local variable
   printf("x = %d, y = %d\n",x,y);
   return 0;
}
```

Output:
x = 0, y = 4194432

Here,
x is a global variable and y is a local variable.
at the time of declaring global variable if we do not assign any value, it will automatically assign 0.
But the local variable can't do this. That's why local variable assign garbage value.

- **What will be the output of the program bellowed?**

```
#include <iostream>
using namespace std;
int x = 1; //global variable
void myfnc(int y){
    y=y*2;
    x=x+10;
    printf("myfnc, x = %d, y = %d\n",x,y);
}
int main()
{
    int y = 5; // local variable
    x = 10;
    myfnc(y);
    printf("main, x = %d, y = %d\n",x,y);
    return 0;
}
```

Output:
myfnc, x = 20, y = 10
main, x = 20, y = 5

Here,
If any program has a global variable and a function of a local variable that is the same as the global variable then the function will evaluate the local variable.

- **What will be the output of the program bellowed?**

```
#include <iostream>
using namespace std;
int x = 1; //global variable
int y = 5; //glocal variable
void myfnc(){
   y=y*2;
   x=x+10;
   printf("myfnc, x = %d, y = %d\n",x,y);
}
int main()
{
   myfnc();
   printf("main, x = %d, y = %d\n",x,y);
   return 0;
}
```

Output:
myfnc, x = 11, y = 10
main, x = 11, y = 10

Static variable:

Static variables have a property of preserving their value even after they are out of their scope! Hence, static variables preserve their previous value in their previous scope and are not initialized again in the new scope.
We can write static variable in two ways :
In global scope(out of function) and function scope(into the function).

- **What will be the output of the program bellowed?**

```
#include <iostream>
using namespace std;
int a;
static int b;
void func()
{
   a=a+1;
   b=b+1;
}
int main()
{
   func();
   printf("a=%d\n",a);
   printf("b=%d\n",b);
   return 0;
}
```

Output:

```
a=1
b=1
```

- **What will be the output of the program bellowed?**

```
#include <iostream>
using namespace std;
void func()
{
   int a=10;
   static int s=10;
   a=a+2;
   s=s+2;
   printf("a = %d, s = %d\n",a,s);
}
int main()
{
   func();
   func();
   func();
   return 0;
}
```

Output:
a = 12, s = 12
a = 12, s = 14
a = 12, s = 16

Here,
Note that we declared two integers a and s in the func function. One is an ordinary integer variable, the other is a static integer variable. I assigned the value 10 to both. Then I added 2 to the two variables and printed it. Now call func

function three times from main function. a variable did 12 leaps every time but variable s printed 12, 14 and 16 respectively. This is the property of static variables. The first time the function (the function in which the static variable is declared) is crawled, the value assigned to that variable is not deleted after the function is exited. It remains in memory as long as the program is running.

The interesting thing is that since the variable is declared inside a function, it cannot be accessed in any other function outside of that function, it can only be accessed if that function is called again. So the second time we call the func function, the value of the s variable remains 12 and adds 2 to it, printing 14.

3. cout<< and printf()

cout<< :

The cout object in C++ is an object of class iostream. It is defined in iostream header file. It is used to display the output to the standard output device i.e. monitor. It is associated with the standard C output stream stdout.

The data needed to be displayed on the screen is inserted in the standard output stream (cout) using the insertion operator(<<).

- **Make a program that will print/show your name.**

```
#include<iostream>
using namespace std;
int main()
{
   cout << "I am Hamim";
   return 0;
}
```

or,

```
#include<iostream>
using namespace std;
int main()
{
   cout << "I " << "am "<< "Hamim";
   return 0;
}
```

Output:
I am Hamim

Here,
iostream = input output stream Header file.
With the help of #include it is included with the program.

int = integer is a keyword.
For compilation and running all program starts from main() function.
All functions have a first bracket sign (()) after the function name.

return is keyword.
return 0 means a successful termination.

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
   cout<<"Hamim Talukder\nJim\nRahim\n";
   cout<<"Tangail\n";
   cout<<"01731578498";
   return 0;
}
```

Output:
Hamim Talukder
Jim
Rahim
Tangail
01731578498

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
   cout<<"\"Hamim Talukder\"\n";
   cout<<"\tTangail\n";
   cout<<"01731578498";
   return 0;
}
```

Output:
"Hamim Talukder"
	Tangail
01731578498

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
   cout<<"Hamim Talukder"<<endl;
   cout<<"Tangail\n"<<endl;
   cout<<"01731578498";
   return 0;
}
```

Output:

Hamim Talukder
Tangail

01731578498

Here,
endl makes a new line in the output of the program. It works as like '\n' escape sequence do.

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
   cout << "Hamim Talukder"<< endl <<"Tangail"
<< endl << "01731546387";
   return 0;
}
```

Output:
Hamim Talukder
Tangail
01731546387

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
   //I am Hamim. I love Allah infinity.
   /*I love my country and my countries
   people. I want to do something for them.*/
   cout << "Hamim Talukder"<< endl ;
   return 0;
}
```

Here,
// and /* */ are comment of coding . Using these we can write anything in program but these will not effect in program because compiler do not take it as a part of a program. // this sign use to write one line of comment and /* */ this sign use to determine many lines of comment and it start from /* and end at */.

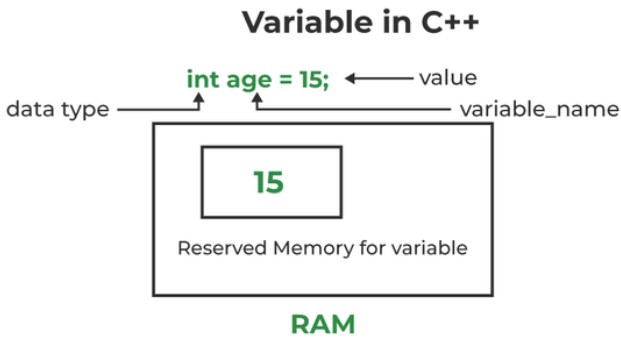
C++ Variables :

Variables in C++ is a name given to a memory location. It is the basic unit of storage in a program.

- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
- In C++, all the variables must be declared before use.

Rules For Declaring Variable :

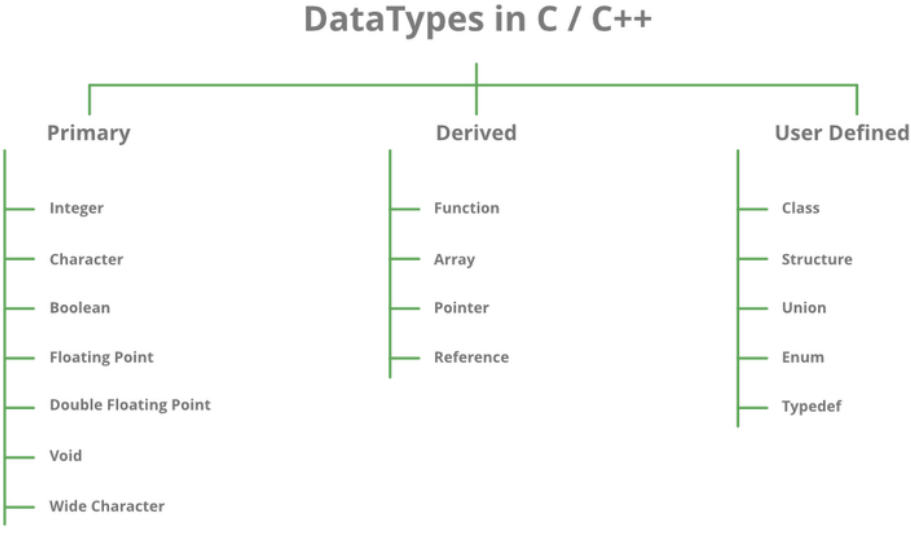
1. The name of the variable contains letters, digits, and underscores.
2. The name of the variable is case sensitive (ex Arr and arr both are different variables).
3. The name of the variable does not contain any whitespace and special characters (ex #,$,%,*, etc).
4. All the variable names must begin with a letter of the alphabet or an underscore(_).
5. We cannot used C++ keyword(ex float,double,class)as a variable name.



The variable's character can be (A to Z), (a to z), (0 to 9), (_), and ($) sign. And remember one thing that we can't use digit before variables.

Example:
Hamim_Talukder, hamim$, hamim1

Data types in C++:



- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;

int main()
{
 //Variable declaration
 int num1;
 float num2;
 double num3;
 char ch;

 //Variable initialozation
 num1=12;
 num2=15.5;
 num3=10.7;
 ch='H';
 //Printing values
 cout<<"num1 = "<<num1<<"\n";
 cout<<"num2 = "<<num2<<endl;
 printf("num3 = %.1lf\n",num3);
 printf("ch = %c\n",ch);
 return 0;
}
```

Output:

```
num1 = 12
num2 = 15.5
num3 = 10.7
ch = H
```

- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;

int main()
{
   double num1 = 10.5, num2=20.50;
   char ch;
   ch = 'a';
   cout<< "num1 = " << num1<<endl<<"num2 = "
<<num2<<endl;
   cout<<"ch="<<ch;
   return 0;
}
```

Output:

```
num1 = 10.5
num2 = 20.5
ch=a
```

- **What is the output of the following C++ program fragment:**

```
// C++ program to show difference between
// definition and declaration of a
// variable
#include <iostream>
using namespace std;

int main()
{
    // this is declaration of variable a
    int a;

    // this is initialisation of a
    a = 10;
    cout<<"a="<<a<<endl;

    // this is definition = declaration + initialisation
    int b = 20;
    cout<<"b="<<b<<endl;

    // declaration and definition
    // of variable 'a123'
    char a123 = 'a';
    cout<<"a123="<<a123<<"\n";
```

```
    // This is also both declaration and definition
    // as 'c' is allocated memory and
    // assigned some garbage value.
    float c=12.3;
    cout<<"C="<<c<<endl;

    // multiple declarations and definitions
    int _c, _d45, e;

    // Let us print a variable
    cout << a123 << endl;

    return 0;
}
```

Output:

```
a=10
b=20
a123=a
C=12.3
a
```

- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;

int main()
{
   int num1, num2;
   num1=10;
   num2=20;
   int sum = num1 + num2;
   cout<<"Sum is : "<<sum;
   return 0;
}
```

or,

```
#include <iostream>
using namespace std;

int main()
{
   int num1, num2;
   num1=10;
   num2=20;
   cout<<"Sum is : "<<num1+num2;
   return 0;
}
```

Output:

Sum is : 30

- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;

int main()
{
   char name[16] = "Hamim Talukder";
   cout<<name;
   return 0;
}
```

Output:
Hamim Talukder

- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;

int main()
{
   char name[16] = "Hamim Talukder";
   cout<<"My name is "<<name;
   return 0;
}
```

Output:
My name is Hamim Talukder

Formatting output of C++ :

- **showpoint :**
  showpoint function is used to show number after decimal including integer number. decimal.
- **noshowpoint :**
  noshowpoint function is used not to show numbers after decimal.
- **setprecision(n) :**
  setprecision(n) function is used to show numbers according to n parameter after decimal including integer number.
- **fixed**
- **set()**

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
  float num1,num2;
  cout<<"Enter 2 numbers : ";
  cin>>num1>>num2;
  float sum = num1+num2;
  cout<<"Sum is : "<<sum;
  cout<<endl;
  float sub = num1-num2;
  cout<<"Subtraction is : "<<sub;
  cout<<endl;
  float mul=num1*num2;
  cout<<"Multiplication is : " <<mul<<endl;
  double div=(float) num1/num2;
  cout<<"Division is : "<<div;
  cout<<endl;
  return 0;
}
```

Output:

```
Enter 2 numbers : 10 3
Sum is : 13
Subtraction is : 7
Multiplication is : 30
Division is : 3.33333
```

- **showpoint :**

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
  float num1,num2;
  cout<<"Enter 2 numbers : ";
  cin>>num1>>num2;

  cout<<showpoint;

  float sum = num1+num2;
  cout<<"Sum is : "<<sum;
  cout<<endl;
  float sub = num1-num2;
  cout<<"Subtraction is : "<<sub;
  cout<<endl;
  float mul=num1*num2;
  cout<<"Multiplication is : " <<mul<<endl;
  double div=(float) num1/num2;
  cout<<"Division is : "<<div;
  cout<<endl;
  return 0;
}
```

Output:

```
Enter 2 numbers : 10 3
Sum is : 13.0000
Subtraction is : 7.00000
Multiplication is : 30.0000
Division is : 3.33333
```

Here,
We have used showpoint function to show numbers after decimal.

- **noshowpoint :**

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
  float num1,num2;
  cout<<"Enter 2 numbers : ";
  cin>>num1>>num2;

  cout<<showpoint;

  float sum = num1+num2;
  cout<<"Sum is : "<<sum;
  cout<<endl;

  cout<<noshowpoint;    // noshowpoint

  float sub = num1-num2;
  cout<<"Subtraction is : "<<sub;
  cout<<endl;
  float mul=num1*num2;
  cout<<"Multiplication is : " <<mul<<endl;
  double div=(float) num1/num2;
  cout<<"Division is : "<<div;
  cout<<endl;
  return 0;
}
```

Output:

```
Enter 2 numbers : 10 3
Sum is : 13.0000
Subtraction is : 7
Multiplication is : 30
Division is : 3.33333
```

Here,
We have used noshowpoint function not to show numbers after decimal.

- **setprecision(n) :**

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
#include<iomanip>
using namespace std;
int main()
 {
 float num1,num2;
 cout<<"Enter 2 numbers : ";
 cin>>num1>>num2;

 cout<<showpoint;
 cout<<setprecision(10);

 float sum = num1+num2;
 cout<<"Sum is : "<<sum;
 cout<<endl;
 float sub = num1-num2;
 cout<<"Subtraction is : "<<sub;
 cout<<endl;

  float mul=num1*num2;
  cout<<"Multiplication is : " <<mul<<endl;
  double div=(float) num1/num2;
  cout<<"Division is : "<<div;
  cout<<endl;
  return 0;
}
```

Output:

```
Enter 2 numbers : 10 3
Sum is : 13.00000000
Subtraction is : 7.000000000
Multiplication is : 30.00000000
Division is : 3.333333333
```

Here,
We have used setprecision(n) function to show number after decimal and this function is in <iomanip> = input output manipulation header file.

- **fixed :**

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
#include<iomanip>
using namespace std;
int main()
{
 float num1,num2;
 cout<<"Enter 2 numbers : ";
 cin>>num1>>num2;

 cout<<showpoint;
 cout<<fixed;
 cout<<setprecision(10);

 float sum = num1+num2;
 cout<<"Sum is : "<<sum;
 cout<<endl;
 float sub = num1-num2;
 cout<<"Subtraction is : "<<sub;
 cout<<endl;
 float mul=num1*num2;
 cout<<"Multiplication is : " <<mul<<endl;
 double div=(float) num1/num2;
 cout<<"Division is : "<<div;
 cout<<endl;
 return 0;
}
```

Output:

```
Enter 2 numbers : 10 3
Sum is : 13.0000000000
Subtraction is : 7.0000000000
Multiplication is : 30.0000000000
Division is : 3.3333333333
```

- **setw(n) :**

- **What is the output of the following C++ program fragment:**

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
  int n, m;
  cout << "Enter row : ";
  cin >> n;
  cout << "Enter column : ";
  cin >> m;
  int arr1[n][m], arr2[n][m];

  cout << "Enter elements for 1st matrix : \n";
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < m; j++)
    {
      cout << "arr1[" << i << "]"
         << "[" << j << "] = ";
      cin >> arr1[i][j];
    }
    cout << endl;
  }
```

```
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < m; j++)
    {
      cout << setw(3) << arr1[i][j];
    }
    cout << endl;
  }
}
```

Output :

```
Enter row : 3
Enter column : 4
Enter elements for 1st matrix :
arr1[0][0] = 23
arr1[0][1] = 7
arr1[0][2] = 56
arr1[0][3] = 9

arr1[1][0] = 89
arr1[1][1] = 0
arr1[1][2] = 34
arr1[1][3] = 12

arr1[2][0] = 3
arr1[2][1] = 67
arr1[2][2] = 89
arr1[2][3] = 4

  23   7  56   9
  89   0  34  12
   3  67  89   4
```

- **What is the output of the following C++ program fragment:**

```
 #include<iostream>
#include<iomanip>
using namespace std;
int main()
{
 float num1,num2;
 cout<<"Enter 2 numbers : ";
 cin>>num1>>num2;

 cout<<showpoint;
 cout<<fixed;
 cout<<setprecision(10);
 float sum = num1+num2;
 cout<<setw(20)<<"Sum is : "<<sum;
 cout<<endl;
 float sub = num1-num2;

  cout<<setw(20)<<"Subtraction is : "<<sub;
  cout<<endl;
  float mul=num1*num2;
  cout<<setw(20)<<"Multiplication is : "
<<mul<<endl;

  double div=(float) num1/num2;
  cout<<"Division is : "<<div;
  cout<<endl;
  return 0;
}
```

Output:

```
Enter 2 numbers : 10 3
        Sum is : 13.0000000000
  Subtraction is : 7.0000000000
Multiplication is : 30.0000000000
Division is : 3.3333333333
```

- **Make a program that can calculate area of triangle.**

```
#include<iostream>
using namespace std;
int main()
{
  double base,height,area;
  cout<<"Enter base : ";
  cin>>base;
  cout<<"Enter Height : ";
  cin>>height;
  area = (double)1/2*base*height;
  cout<<"Area of triangle = "<<area;
  return 0;
}
```

Output:

Enter base : 21.3
Enter Height : 56.8
Area of triangle = 604.92

printf() :

printf() is a function which work is to print/show the output of the program.

Hole	Description	Example	Result
%d, %i	Print any integer	printf "%d" 5	5
%x, %o	Print any integer in Hex or Octal format	printf "%x" 255	ff
%s	Print any string	printf "%s" "ABC"	ABC
%f	Print any floating-point number	printf "%f" 1.1M	1.100000
%c	Print any character	printf "%c" '\097'	a
%b	Print any Boolean	printf "%b" false	False
%O	Print any object	printf "%0" (1, 2)	(1, 2)
%A	Print anything	printf "%A" (1, [])	(1, [])

- **Make a program that will print your name.**

```
#include<stdio.h>
int main()
{
  printf("I am Hamim");
  return 0;
}
```

Here,
Statement = printf("I am Hamim"),
Library function = #include<stdio.h>
Header file = stdio.h,
Extention = .h

- **Make a program that will show sum of 50 and 60.**

```
#include<stdio.h>
int main()
{
  int a;
  int b;
  int sum;
  a = 50;
  b = 60;
  sum = a+b;
  printf("Sum is %d",sum);
  return 0;
}
```

Here,
Variable = a,b,sum,
Data type = int (for integer number) = %d,
float (for fraction number) = %f,
double (for real number) = %lf,

or,

```
#include<stdio.h>
int main()
{
  int a=50;
  int b=60;
  int sum;
  sum = a+b;
  printf("Sum is %d",sum);
  return 0;
}
```

or,

```
#include<stdio.h>
int main()
{
  int a=50,b=60,sum;
  sum = a+b;
  printf("Sum is %d",sum);
  return 0;
}
```

or,

```
#include<stdio.h>
int main()
{
 int a=50,b=60;
 printf("Sum is %d",a+b);
 return 0;
}
```

or,

```
#include<stdio.h>
int main()
{
  int a=50,b=60;
  printf("%d+%d=%d",a,b,a+b);
  return 0;
}
```

- **What will be the output of the program given bellow?**

```
#include<stdio.h>
int main()
{
  int x,y;
  x=1;
  y=x;
  x=2;
  printf("%d",y);
  return 0;
}
```

Here,
The output will be 1. Now think about why it happend ? The reason is y=x is not a equation in program and here we are just assigning the value of y variable.

- **Make a program that will show sum of 50.45 and 60.**

```
#include<stdio.h>
int main()
{
  int a=50.45,b=60;
  printf("%d+%d=%d",a,b,a+b);
  return 0;
}
```

Here,
The output will be 110. Now think about why it happend ? Because a is an integer number and we have assigned 50.45. But C compiler take it as 50 using type cast.

- **What will be the output of the program given bellow.**

```
#include <stdio.h>
int main()
{
  int a=277;
  printf("a = %d\n",a);
  printf("a = %5d\n",a);
  printf("a = %05d\n\n",a);

  float b=277.1, c=277, d=277.10, e=277.0;
  printf("b = %f\n",b);
  printf("b = %09.3f\n",b);
  printf("b = %9.3f\n",b);
  printf("b = %9f\n",b);
  printf("c = %f\n",c);
  printf("c = %09.3f\n",c);
  printf("c = %9f\n",c);
  printf("d = %09.3f\n",d);
  printf("e = %9f\n",e);

  return 0;
}
```

Output:

```
a = 277
a =   277
a = 00277

b = 277.100006
b = 00277.100
b =   277.100
b = 277.100006
c = 277.000000
c = 00277.000
c = 277.000000
d = 00277.100
e = 277.000000
```

- **Range of data type :**

Type	size	Range	Format
unsigned char	1 byte	0 to 255	%c
signed char or char	1 byte	-128 to +127	%c
unsigned int	2 byte	0 to 65535	%u
signed int or int	2 byte	-32,768 to +32767	%d/%i
unsigned short int	2 byte	0 to 65535	%u
signed short int or short int	2 byte	-32,768 to +32767	%d/%i
unsigned long int	4 byte	0 to +4,294,967,295	
signed long int or long int	4 byte	-2,147,483,648 to +2,147,483,647	
long double	10 byte	3.4E-4932 to 1.1E+4932	%ld
double	8 byte	1.7E-308 to 1.7E+308	%lf
float	4 byte	3.4E-38 to 3.4E+38	%f

Here,
1 byte = 8 bit
1 bit = 0 and 1
2 byte = 16 bit = 2^16
4 byte = 32 bit = 2^32
signed = +value/-value
unsigned = +value

- **Make a program that will show sum of two real number 4.9 and 7.3.**

```
# include<stdio.h>
int main()
{
  double a=4.9, b=7.3, sum;
  sum=a+b;
  printf("Sum is %lf\n",sum);
  printf("Sum is %.1f\n",sum);
  printf("Sum is %.2lf\n",sum);
  printf("Sum is %.3lf\n",sum);
  printf("Sum is %.4lf\n",sum);
  printf("Sum is %.5lf\n",sum);
  printf("Sum is %.9lf\n",sum);
  return 0;
}
```

Here,

Think about why I have written so many printf() statement.

- **Make a double number into integer number using type cast.**

```
#include<stdio.h>
int main()
{
  double x=10.5;
  int n;
  n=(int) x;
  printf("Value of n is %d\n",n);
  printf("Value of x is %lf\n",x);
  return 0;
}
```

4. cin>> and scanf()

cin>> :

The cin object in C++ is an object of class iostream. It is used to accept the input from the standard input device i.e. keyboard.

It is associated with the standard C input stream stdin. The extraction operator(>>) is used along with the object cin for reading inputs. The extraction operator extracts the data from the object cin which is entered using the keyboard.

- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;

int main()
{
   int num;
   cout<<"Enter an integer number : ";
   cin>>num;
   cout<<"Entered number is : "<<num;
   return 0;
}
```

Output:

```
Enter an integer number : 12
Entered number is : 12
```

- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;

int main()
{
   int num1;
   double num2;
   float num3;
   char ch;
   cout<<"Enter an integer number : ";
   cin>>num1;
   cout<<"Entered integer number is : "
<<num1<<endl;
   cout<<"Enter a double number : ";
   cin>>num2;
   cout<<"Entered double number is : "
<<num2<<endl;
   cout<<"Enter a float number : ";
   cin>>num3;
   cout<<"Entered float number is : "
<<num3<<endl;
   cout<<"Enter a character number : ";
   cin>>ch;
   cout<<"Entered character is : "<<ch<<endl;
   return 0;
}
```

Output:

```
Enter an integer number : 12
Entered integer number is : 12
Enter a double number : 13.4
Entered double number is : 13.4
Enter a float number : 16.70
Entered float number is : 16.7
Enter a character number : a
Entered character is : a
```

- **Make a program to take two integer numbers and print/show their sum.**

```
#include <iostream>
using namespace std;

int main()
{
   int num1,num2,sum;
   cout<<"Enter 2 numbers : ";
   cin>>num1>>num2;
   sum = num1 + num2;
   cout<<"Sum = "<<num1<<" + "<<num2<<" = "<<sum;
   return 0;
}
```

or,

```
#include <iostream>
using namespace std;

int main()
{
   int num1,num2;
   cout<<"Enter 2 numbers : ";
   cin>>num1>>num2;
   cout<<"Sum = "<<num1<<" + "<<num2<<" = "
<<num1+num2;
   return 0;
}
```

Output:

```
Enter 2 numbers : 10 20
Sum = 10 + 20 = 30
```

scanf() :

In the C programming language, scanf is a function that reads formatted data from stdin (i.e, the standard input stream, which is usually the keyboard, unless redirected) and then writes the results into the arguments given.
This function belongs to a family of functions that have the same functionality but differ only in their source of data. For example, fscanf gets its input from a file stream, whereas sscanf gets its input from a string.

- **Make a program that can show sum of any two integer numbers.**

```
# include<stdio.h>
int main()
{
  int a,b,sum;
  scanf("%d%d",&a,&b);
  sum=a+b;
  printf("Sum is %d",sum);
  return 0;
}
```

Output:
10 12
Sum is 22

- **Make a program that can show sum of any two numbers.**

```
# include<stdio.h>
int main()
{
  double a,b,sum;
  scanf("%lf",&a);
  scanf("%lf",&b);
  sum=a+b;
  printf("Sum is %.3lf",sum);
  return 0;
}
```

- **Make a program that will convert decimal number into octal and hexadecimal number.**

```
#include<stdio.h>
int main()
{
    int number;
    printf("Decimal number = ");
    scanf("%d",&number);

    printf("Octal number = %o\n",number);
    printf("Hexadecimal number = %x\n",number);
    return 0;
}
```

- **Make a program that will show the first letter of your name.**

```
# include<stdio.h>
int main()
{
  char ch;
  printf("Enter the first letter of your name :");
  scanf("%c",&ch);
  printf("The first letter of your name is : %c\n",ch);
  return 0;
}
```

Here,
Data type = char (for any character) = %c,
It can be any single English letter or sign.
or,

```
# include<stdio.h>
int main()
{
 char ch;
 printf("Enter the first letter of your name :");
 ch = getchar();
 printf("The first letter of your name is : %c\n",ch);
 return 0;
}
```

Here,
Getchar is also a function as like char data type .It reads a character then assign into variable as like scanf function do.

- **Make a program that can summation, subtraction, multiplication and division.**

```
# include<stdio.h>
int main()
{
 int num1, num2;
 printf("Enter a number :");
 scanf("%d",&num1);
 printf("Enter another number :");
 scanf("%d",&num2);
 printf("%d+%d=%d\n",num1,num2,num1+num2);
 printf("%d-%d=%d\n",num1,num2,num1-num2);
 printf("%d*%d=%d\n",num1,num2,num1*num2);
 printf("%d/%d=%d\n",num1,num2,num1/num2);
 return 0;
}
```

or,

```
# include<stdio.h>
int main()
{
 double num1, num2;
 printf("Enter a number :");
 scanf("%lf",&num1);
 printf("Enter another number :");
 scanf("%lf",&num2);
 printf("%lf+%lf=%.3lf\n",num1,num2,num1+num2);
 printf("%lf-%lf=%.3lf\n",num1,num2,num1-num2);
 printf("%lf*%lf=%.3lf\n",num1,num2,num1*num2);
 printf("%lf/%lf=%.3lf\n",num1,num2,num1/num2);
 return 0;
}
```

or,

```
# include<stdio.h>
int main()
{
  //I am Hamim. I love Allah infinity.
  /*I love my country and my countries
  people. I want to do something for them.*/
  double a,b,value;
  char sign;
  printf("Enter a number :");
  scanf("%lf",&a);
  printf("Enter another number :");
  scanf("%lf",&b);
  value=a+b;
  sign='+';
  printf("%lf%c%lf=%.2lf\n",a,sign,b,value);
  value=a-b;
  sign='-';
  printf("%lf%c%lf=%.2lf\n",a,sign,b,value);
  value=a*b;
  sign='*';
  printf("%lf%c%lf=%.2lf\n",a,sign,b,value);
  value=a/b;
  sign='/';
  printf("%lf%c%lf=%.2lf\n",a,sign,b,value);
  return 0;
}
```

Here,
// and /* */ are comment of coding . Using these we can write anything in program but these will not effect in program because compiler do not take it as a part of a program. // this sign use to write one line of comment and /* */ this sign use to determine many lines of comment and it start from /* and end at */.

```
#include <stdio.h>
int main()
{
  // test program - comment 1
  printf("Hello ");
  /* We have printed Hello,
  now we shall print World.
  Note that this is a multi-line comment */
  printf("World"); // printed world
  return 0;
}
```

5. Operator And Expression :

- Consider the expression **A + B * 5** , where,

+ , *	are **operators**,
A, B	are **variables**,
5	is **constant**,
A, B and 5	are called **operand**, and
A + B * 5	is an **expression**.

`(a+b)*c`	Operator is `*`, operands are `(a+b)` and `c`
`(a+b)`	Operator is `()`, operand is `a+b`
`a+b`	Operator is `+`, operands are `a` and `b`

Why we are learning these things:

15 * 10 ⟶ Arithmetic operator

300000000 >= 500000000 ⟶ Relational operator

if ( && ) ⟶ Logical operator

Types of operators in C:

NAME OF OPERATORS	OPERATORS
ARITHMETIC OPERATORS	**+ , - , * , / , %**
INCREMENT/DECREMENT OPERATORS	**++ , --**
RELATIONAL OPERATORS	**== , != , <= , >= , < , >**
LOGICAL OPERATORS	**&& , || , !**
BITWISE OPERATORS	**& , ^ , | , ~ , >> , <<**
ASSIGNMENT OPERATORS	**= , += , -= , *= , /= , %= , <<= , >>= , &= , ^= , |=**
OTHER OPERATORS	?: , & , * , SIZEOF() , ,

ARITHMETIC OPERATORS

+ , - , * , / , %

addition Subtraction Multiplication Division Modulus

All are binary operators → means two operands are required to perform the operation.

For example: A + B

Op1 Op2

Operator Precedence and Associativity:

PRECEDENCE ↓ HIGHEST	OPERATORS	ASSOCIATIVITY
	* , / , %	LEFT TO RIGHT
LOWEST	+ , -	LEFT TO RIGHT

Note: Associativity is used only when two or more operators are of same precedence.

For example: + , -

Same precedence therefore we use associativity

+ , - | left to right

Coding Example:

```
#include<iostream>
using namespace std;
int main()
{
 int a=2,b=3,c=4,d=5;
 printf("a*b/c=%d\n",a*b/c);
 printf("a+b-c=%d\n",a+b-c);
 printf("a+b*d-c%%a=%d",a+b*d/b-c%a);
 return 0;
}
```

Output:
a*b/c=1
a+b-c=1
a+b*d-c%a=7

Here:
a+b*d/b-c%a
= a+((b*d)/b)-(c%a)
= 2+((3*5)/3)-(4%2)
= 2+15/3-0
= 2+5-0
= 7-0
= 7

Increment and Decrement operators:

Increment operator is used to increment the value of a variable by one. Similarly, decrement operator is used to decrement the value of a variable by one.

Increment	Decrement
int a=5	int a=5;
a++;	a-- ;
a=6	a=4

a++; is same as a = a + 1;
a-- ; is same as a = a - 1;

Both are unary operators.
- because they are applied on single operand.

a++; ✓ a ++ a; ✗

Pre-increment operator
++a;

post-increment operator
a++;

Pre-decrement operator
--a;

post-decrement operator
a--;

(a+b)++; error!
++(a+b); error!

error: lvalue required as increment operand

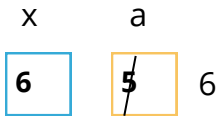
Compiler is expecting a variable as an increment operand but we are providing an expression (a+b) which dose not have the capability to store data.

Question: What is the difference between pre-increment and post-increment operator or pre-decrement and post-decrement operator?

Pre - means first increment/decrement then assign it to another variable.

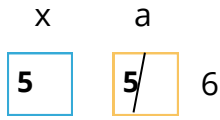
Post - means first assign it to another variable then increment/decrement.

x = ++a;



x = 6

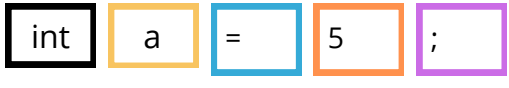
x = a++;



x = 5

Token Generation:

- Lexical analysis is the first phase in the compilation process.
- Lexical analyzer(scanner) scans the whole source program and when it finds the meaningful sequence of characters(lexemes) then it converts it into a token.
- Token: Lexemes mapped into token-name and attribute-value.
  Example: int → <keyword, int>
- It always matches the longest character sequence.

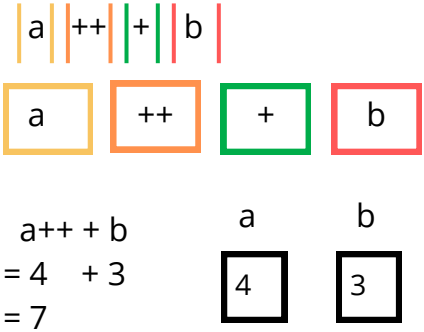


- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
 int a=4,b=3;
 printf("%d\n",a+++b);
 return 0;
}
```

Output:
7

Here,



Post-increment/ decrement in context of equation:
First use the value in the equation and then increment the value.

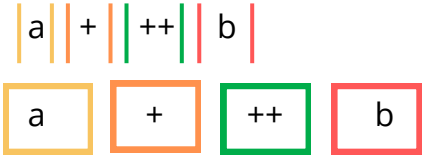
Pre-increment/ decrement in context of equation:
First increment the value and then use in the equation after completion of the equation.

- **What is the output of the following C++ program fragment:**

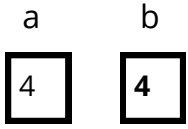
```
#include<iostream>
using namespace std;
int main()
{
 int a=4,b=3;
 printf("%d\n",a + ++b);
 return 0;
}
```

Output:
8

Here,



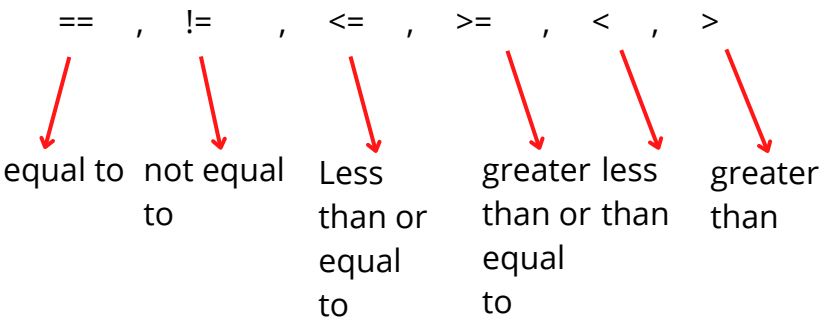
a+ ++ b
= 4 + 4
= 8



Post-increment/ decrement in context of equation:
First use the value in the equation and then increment the value.

Pre-increment/ decrement in context of equation:
First increment the value and then use in the equation after completion of the equation.

Relation operation:



Used for comparing two values

- all relational operators will return either true or False.

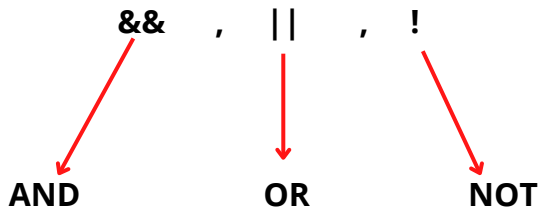
4==5 is equivalent to is 4 == 5 ?
Answer: False
4!=5 is equivalent to is 4!==5 ?
Answer: True

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
 int a=3,b=4;
 if(b>=a){
  printf("Bingo! You are in");
 }
 else{
  printf("OOPS! You are out");
 }
 return 0;
}
```

Output:
Bingo! You are in

Logical operators:



- && and || are used to combine two conditions. && - returns TRUE when all the conditions under consideration are true and returns FALSE when any one or more than one condition is false.

For example:

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
 int a = 5;
 if (a == 5 && a != 6 && a <= 56 && a > 4)
    {
 printf("Welcome to this beautiful world of
operators");
    }
  return 0;
}
```

Output:

Welcome to this beautiful world of operators

- || - returns TRUE when one or more than one condition under consideration is true and returns FALSE when all conditions are false.

For example:

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
 int a = 5;
 if (a == 5 || a != 6 || a <= 56 || a > 4)
 {
  printf("Welcome to this beautiful world of
operators");
 }
 return 0;
}
```

Output:
Welcome to this beautiful world of operators

- ! operator is used to complement the condition under consideration.
  ! - returns TRUE when condition is FALSE and returns FALSE and False when condition is TRUE.

For example:

- **What is the output of the following C program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
 int a = 5;
 if (a == 5 || a != 6 || a <= 56 || a > 4)
 {
  printf("Welcome to this beautiful world of
operators");
 }
 return 0;
}
```

Output:
Welcome to this beautiful world of operators

Concept of short circuit in logical operators:

Short circuit in case of &&: Simply means if there is a condition anywhere in the expression that returns false, then the rest of the conditions after that will not be evaluated.
For example:

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
 int a = 5, b=3;
 int incr;
 incr = (a<b) && (b++);
 printf("%d\n",incr);
 printf("%d\n",b);
 return 0;
}
```

Output:
0
3

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
 int a = 5,b=3;
 int incr;
 incr = (a>b) && (b++);
 printf("%d\n",incr);
 printf("%d\n",b);
 return 0;
}
```

Output:
1
4

Short circuit in case of ||: Simply means if there is a condition anywhere in the expression that returns True, then the rest of the conditions after that will not be evaluated.
For example:

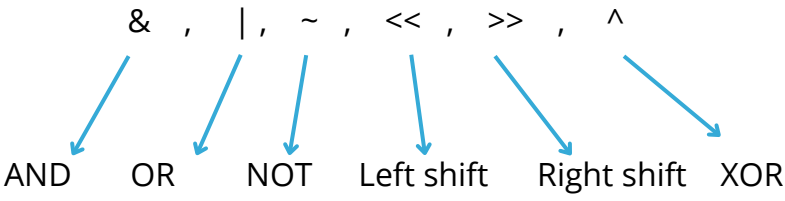
- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
 int a = 5,b=3;
 int incr;
 incr = (a>b) || (b++);
 printf("%d\n",incr);
 printf("%d\n",b);
 return 0;
}
```

Output:
1
3

Bitwise operators:

As name suggests - it dose bitwise manipulation.



BIT BY BIT

Bitwise AND (&) operator:

- It takes two bits at a time and perform AND operation.
- And (&) is binary operator. It takes two numbers and perform bitwise AND.
- Result of AND is 1 when both bits are 1.

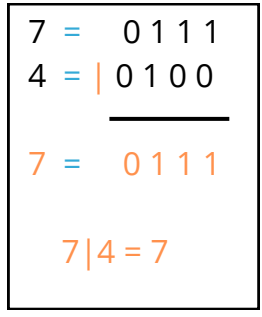
```
7 =    0 1 1 1
4 = &  0 1 0 0
-------------
4 =    0 1 0 0

  7&4 = 4
```

Truth Table

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise OR (|) operator:

- It takes two bits at a time and perform OR operation.
- OR (|) is binary operator. It takes two numbers and perform bitwise OR.
- Result of OR is 0 when both bits are 0.

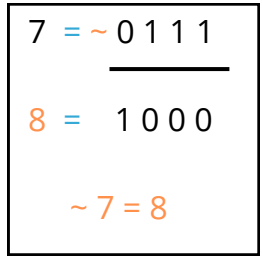


Truth Table

A	B	A&B
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise NOT (~) operator:

- NOT is a unary operator.
- It's job is to complement each bit one by one.
- Result of NOT is 0 when bit is 1 and 1 when bit is 0.



Truth Table

A	~A
0	1
1	0

Difference between bitwise and logical operators:

```
#include<iostream>
using namespace std;
int main()
{
 char x=1,y=2; /* x=1(0000 0001),
      y=2(0000 0010)
     1&2=0(0000 0000)
 1&&2 = TRUE && TRUE = TRUE =1 */

 if(x&y)
 printf("Result of x&y is 1\n");
 if(x&&y)
 printf("Result of x&&y is 1\n");
 return 0;
}
```

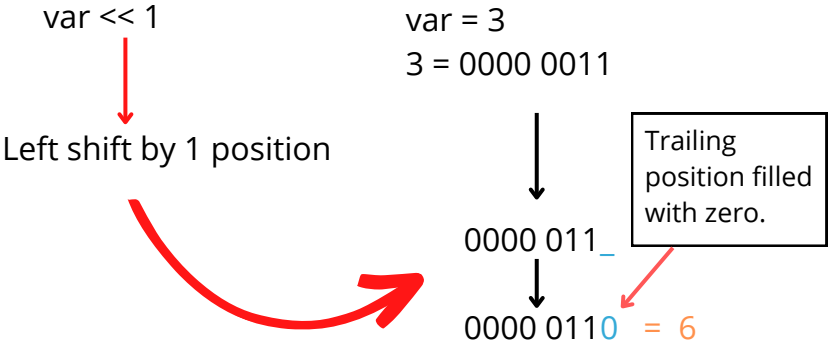
output:
Result of x&&y is 1

Left shift operator:

First operand << Second operand

Whose bits get left shifted

- How left shift works?



Important points:

Left shifting is equivalent to multiplication by 2^(right Operator) .

Example: var = 3;
var << 1 Output: 6 [3 x 2^1]
var << 4 Output: 48 [3 x 2^4]

Important points:

- When bits are shifted left then trailing positions are filled with zeros.

```
#include<iostream>
using namespace std;
int main()
{
 char var=3; // 3 in binary = 0000 0011
 printf("%d\n",var<<1);
 return 0;
}
```

char = 1 byte = 8 bits

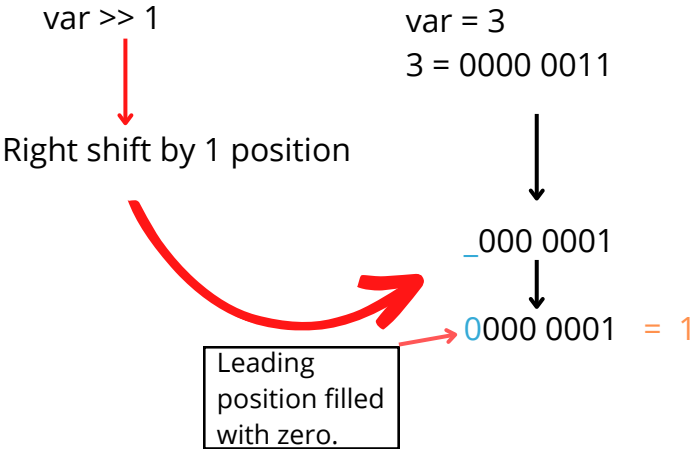
output:
6

Right shift operator:

First operand >> Second operand

Whose bits get right shifted

- How right shift works?



Important points:

Right shifting is equivalent to division by 2^(right Operator) .

Example:

var = 3;
var >> 1 Output: 1 [3 / 2^1]

var = 32
var >> 4 Output: 2 [32 / 2^4]

Important points:

- When bits are shifted right then leading positions are filled with zeros.

```
#include<iostream>
using namespace std;
int main()
{
 char var=3; // 3 in binary = 0000 0011
 printf("%d\n",var>>1);
 return 0;
}
```

output:
1

Bitwise XOR operator:

X - OR

Inclusive OR:

- Either A is 1 or B is 1 or Both are 1, then the output is 1.
- Including BOTH.

Exclusive OR:

- Either A is 1 or B is 1 then the output is 1 but when both A and B are 1 then Output is 0.
- Excluding BOTH

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Only difference

- Bitwise XOR(^) is binary operator. It takes two numbers and perform bitwise XOR.
- Result of XOR is 1 when two bits are different otherwise the result is 0.

```
7 =    0 1 1 1
4 = ^  0 1 0 0
       -------
7 =    0 0 1 1

7^4 = 3
```

Truth Table

A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

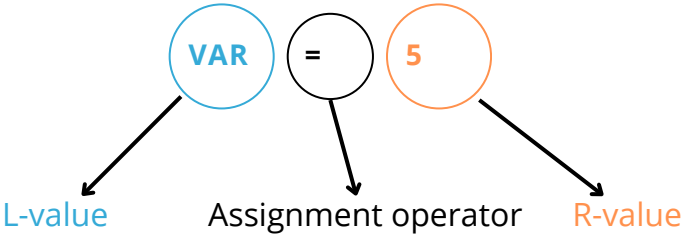
- **What is the output of the following C program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
 int a = 4,b=3;
 a=a^b;
 b=a^b;
 a=a^b;
 printf("After XOR, a=%d and b=%d\n",a,b);
 return 0;
}
```

Output:
After XOR, a=3 and b=4

Assignment operator:

Values to a variable can be assigned using assignment operator.
Requires two values - L-value and R-value.
This operator copies R-value to L-value.



Shorthand assignment operators:

+=	First addition than assignment
-=	First subtraction than assignment
*=	First multiplication than assignment
/=	First division than assignment

Example: a += 1 is equivalent to a = a +1
Similar concept for other shorthand assignment operators as well.

- **What is the output of the following C program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
 int a=7;
 a^=5;
 printf("%d", printf("%d", a+=3));
 return 0;
}
```

Output:
51

- **What will be the output of the program given bellow :**

```
#include<iostream>
using namespace std;
int main()
{
 int a=7;
 a^=5;
 printf("%d\n", printf("%d \n", a+=10));
 int n=printf("%d  \n", a+=10);
 cout<<n<<endl;
 printf("%d",  printf("%d ", printf("%d ", a+=10)));
 return 0;
}
```

Output:

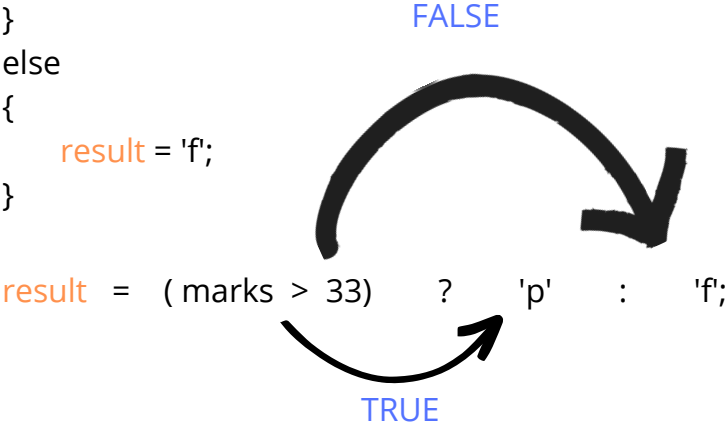
```
12
4
22
5
32 3 2
```

Conditional operator:

Look and feel: ? :

```
char result;
int marks;
if(marks > 33)
{
      result = 'p';
}
else
{
     result = 'f';
}
```

```
char result;
int marks;
result = (marks > 33) ? 'p' :  'f';
```



(marks > 33) is a boolean expression, therefore it will return either true or false.

(marks > 33) ? 'p ' : 'f' is a conditional expression, which is after all expression, therefore it is an r-value and result is l-value.

Ternary opereator

Ternary operator (? :) is used to replace if-else statements. It has the following general from.

Exp1 ? Exp2 : Exp3 ;

For example, consider the following code:

```
if(condition){
    var = 30;
}
else{
    var = 40;
}
```

Above code can be rewritten like this:

```
var = (Y<10) ? 30 : 40;
```

Here,
X is assigned the value of 30 if Y is less than 10 and 40 if it is not .

- **What will be the output of the program given bellow.**

```
#include <iostream>
using namespace std;
int main()
{
 int var, y = 7;
 if (y < 10){
  var = 30;
 }
 else{
  var = 40;
 }
 cout<<var<<endl;
 return 0;
}
```

Here,
The output will be 30.
Now we will do this program using ternary operator.

or,

```
#include <iostream>
using namespace std;
int main()
{
 int var,y=7;
 var = (y<10) ? 30 : 40;
 cout<<var<<endl;
 return 0;
}
```

Quick facts checklist:

- Conditional operator is the only ternary operator available in the list of operators in C language.
- As in Expression1 ? Expression2 : Expression3 , expression1 is the boolean expression. If we simply write 0 instead of some boolean expression than that simply means FALSE and therefore Expression3 will get evaluated.

Example:

```
int result;
result = 0 ? 2 : 1
```

result

1

Comma (.) opeartor:

Comma operator can be used as an "operator".

```
int a= (3,4,8);
printf("%d",a);
```

Output:
8

Comma operator returns the rightmost operand in the expression and it simply evaluates the rest of the operands and finally reject them.

Example:

```
int var = (print("%d\n", "HELLO"),  5  ;
print("%D"< var);
```

This value will be returned to var after evaluating the first operand

It will simply not rejected. First evaluated and then rejected

OUTPUT:
HELLO!
5

- Comma operator is having least precedence among all the operators available in C language.

Example 1:

```
int a;
a = 3, 4, 8;
printf("%d", a);
```

≡

```
int a;
(a = 3), 4, 8;
printf("%d", a);
```

Output: 3

Example 2:

```
int a = 3, 4, 8;
printf("%d",a);
```

Here,
Comma is behaving like a separator.

int a = 3, 4, 8;
is equivalent to
int a = 3; int 4; int 8;

ERROR!

Example 3:

```
int a;
a = (3,4,8);
printf("%d", a);
```

or

```
int a = (3,4,8);
printf("%d", a);
```

Here,
Bracket has the highest precedence than only other operator.

Output: 8

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
 int var;
 int num;
 num = (var = 15, var+35);
 printf("%d",num);
 return 0;
}
```

Output:
50

Precedence and Associativity of Operators::

Precedence of operators come into picture when in an expression we need to decide which operator will be evaluated first. Operator with higher precedence will be evaluated first.

Example: 2 + 3 * 5

Precedence of multiplication is greater than addition.

2 + (3 * 5) = 17

(2 + 3) * 5 = 25

Associativity of operators come into picture when precedence of operators are some need to decide which operator will be evaluated first.

Example: 10 / 2 * 5

Left to right: (10 / 2) * 5 = 25 

Right to left: 10 / (2 * 5) = 1 

Associativity can be either :

1. Left to right
2. Right to left

() - parenthesis in function calls:

Example: int var = fun();

operand

= operator is having less precedence as co,pared to () therefore, () belongs to fun will be treated as a function.

int var = (fun());

If suppose = operator is having greater precedence then, fun will belong to = operator and therefore it will be treated as a variable.

int (var = fun) ();

Member access operators (-> .) :

- They are used to access members of structures.
- We will talk about structures later in this course.

Postfix Increment/Decrement (++, --)

- Precedence of Postfix Increment/Decrement operator is greater than Prefix Increment/ Decrement.
- Associativity of Postfix is also different from Prefix. Associativity of postfix operators is from left to right and that of prefix operators is from right to left.

- Associativity can only help if there are two or more operators of some precedence and not when there is just one operator.
- Operators with some precedence have some associativity as well.

Example:

```
int main()
{
    int a;
    a = fun1() + fun2();
    printf("%d", a);
    return 0;
}
```

```
int fun1()
{   printf("Neso");
    return 1;
}

int fun2()
{   printf("Academy");
    return 1;
}
```

Which function is called first ?
fun1() or fun2()?

It is not defined whether fun1() will be called first or whether fun2() will be called. Behaviour is undefined and output is compiler dependent.

NOTE: Here associativity will not come into picture as we have just one operator and which function will be called first is undefined. Associativity will only work when we have more than one operators of same precedence

Example:

```
int main()
{
    int a;
    a = fun1() + fun2();
    printf("%d", a);
    return 0;
}
```

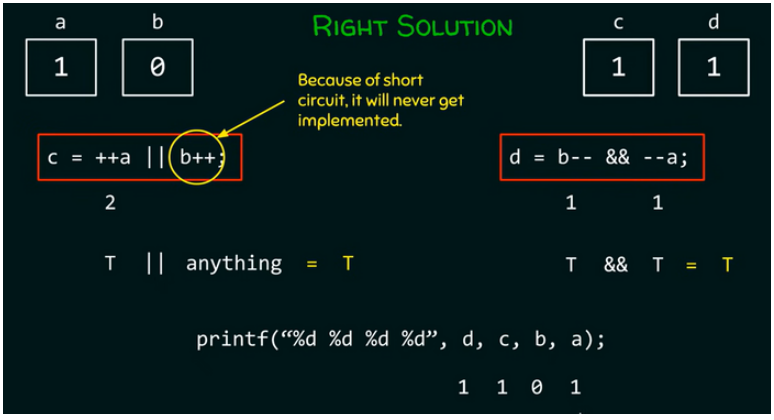
```
int fun1()
{   printf("Neso");
    return 1;
}

int fun2()
{   printf("Academy");
    return 1;
}
```

Output: NesoAcademy2
OR
AcademyNeso2

What is the output of the following C program fragment?

```
int a = 1;
int b = 1;
int c = ++a || b++;
int d = b-- && --a;

printf("%d %d %d %d", d, c, b, a);
```

a) 1 1 1 1
b) 0 1 0 0
c) 1 0 0 1
d) 1 1 0 1



- The following table lists the precedence and associativity of C operators.

Operators are listed top to bottom, in descending precedence.

Precedence	Operator	Description	Associativity
1	`( )`	Function call	
	`[ ]`	Array subscripting	
	`.`	Structure and union member access	
	`->`	Structure and union member access through pointer	
2	`++ --`	Prefix increment and decrement	Right-to-left
	`+ -`	Unary plus and minus	
	`! ~`	Logical NOT and bitwise NOT	
	`(type)`	Type cast	
	`*`	Indirection (dereference)	
	`&`	Address-of	
	`sizeof`	Size-of	
3	`* / %`	Multiplication, division, and remainder	Left-to-right
4	`+ -`	Addition and subtraction	
5	`<< >>`	Bitwise left shift and right shift	
6	`< <=`	For relational operators < and ≤ respectively	
	`> >=`	For relational operators > and ≥ respectively	
7	`== !=`	For relational = and ≠ respectively	
8	`&`	Bitwise AND	
9	`^`	Bitwise XOR (exclusive or)	
10	`|`	Bitwise OR (inclusive or)	
11	`&&`	Logical AND	
12	`||`	Logical OR	
13	`?:`	Ternary conditional	Right-to-Left
14	`=`	Simple assignment	
	`+= -=`	Assignment by sum and difference	
	`*= /= %=`	Assignment by product, quotient, and remainder	
	`<<= >>=`	Assignment by bitwise left shift and right shift	
	`&= ^= |=`	Assignment by bitwise AND, XOR, and OR	
15	`,`	Comma	Left-to-right

▲▼►▲▼►♦▼▼ ▲▼▲▲♦♦►◄▲▼►▼ ▼▲
►■►▼▲► ▼▲▲►■◄ ■▲■■▲
►►■■▲▲■►♦▲ ▼▲ ◄►

NFT NFT NFT

►▼▲▼♦►►■▲◄▼▲▼♦►►▼▼►■▲◄▼▲
►♦◄◄▼▲

▲▼►▲▼►▼▼▲►▼▲▲►◄▲▼►▼▼▲▼►▼
▲►▼▲▲►◄▲▲►►▲▲►◄▲

►▼▲▼O♦►OO◄▼▼◄▼◄O◄▲■►■♦▼♦
►◄►▲◄▼◄◄▲O◄▼♦►O►▼▼◄▲▼►♦O
◄►■■O►♦◄

°O ☼

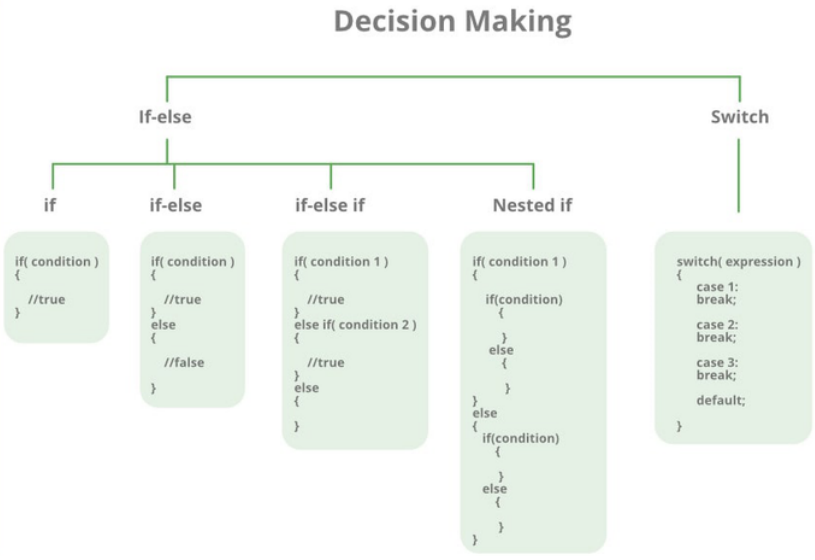
6. if-else

Decision-making statements in programming languages decide the direction of the flow of program execution.

Decision-making statements available in C or C++ are:

1. if statement
2. if-else statements
3. nested if statements
4. if-else-if ladder
5. switch statements
6. Jump Statements:

- break
- continue
- goto
- return



The if-else statement is used to perform two operations for a single condition. The if-else statement is an extension to the if statement using which, we can perform two different operations, i.e., one is for the correctness of that condition, and the other is for the incorrectness of the condition. Here, we must notice that if and else block cannot be executed simiulteneously. Using if-else statement is always preferable since it always invokes an otherwise case with every if condition.

Here,
for greater >,
for smaller <,
for equal ==,
for greater and equal >=,
for smaller and equal <=,
for not equal !=

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;
int main()
{
  int i = 10;

 if (i > 15) {
  printf("10 is greater than 15");
 }

 printf("I am Not in if");
  return 0;
}
```

Output:

I am Not in if

- **Make a program where the person who has the correct 50 Geek Bits has redeemed the gifts otherwise they can’t redeem.**

```
#include<iostream>
using namespace std;
int main()
{
  int i = 20;

    if (i < 15) {

        printf("i is smaller than 15");
    }
    else {

        printf("i is greater than 15");
    }
    return 0;
}
```

Output:
i is greater than 15

- **Make a program where the person having more than 50 Geek Bits and less than 150 Geek Bits he won the GFG T-shirt.**

```
#include <iostream>
using namespace std;
int main()
{
 int i = 10;

  if (i == 10) {
    if (i < 15)
      cout<<"i is smaller than 15\n";
    if (i < 12)
      cout<<"i is smaller than 12 too\n";
    else
      printf("i is greater than 15");
  }
  return 0;
}
```

Output:
i is smaller than 15
i is smaller than 12 too

- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;
int main()
{
  int i = 20;

    if (i == 10)
        cout<<"i is 10";
    else if (i == 15)
        cout<<"i is 15";
    else if (i == 20)
        cout<<"i is 20";
    else
        cout<<"i is not present";
    return 0;
}
```

Output:
i is 20

break :

This loop control statement is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stop there, and control returns from the loop immediately to the first statement after the loop.

- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;

void findElement(int arr[], int size, int key)
{

   for (int i = 0; i < size; i++) {
      if (arr[i] == key) {
         cout<<"Element found at position: "
               <<(i + 1);
         break;
      }
   }
}

int main()
{
   int arr[] = { 1, 2, 3, 4, 5, 6 };

   int n = 6;
```

```
    int key = 3;

    findElement(arr, n, key);
     return 0;
}
```

Output:
Element found at position: 3

continue :

This loop control statement is just like the break statement. The continue statement is opposite to that of the break statement, instead of terminating the loop, it forces to execute the next iteration of the loop.
As the name suggests the continue statement forces the loop to continue or execute the next iteration. When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and the next iteration of the loop will begin.

- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;
int main()
{
 // loop from 1 to 10
    for (int i = 1; i <= 10; i++) {
        if (i == 6)
            continue;

        else
            printf("%d ", i);
    }
  return 0;
}
```

Output:
1 2 3 4 5 7 8 9 10

- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;
int main()
{
  int gfg = 0; // local variable for main
  cout<<"Before if-else block "<<gfg<<endl;
  if (1)
  {
    int gfg = 100; // new local variable of if block
    cout<<"if block "<< gfg <<endl;
  }
  cout<<"After if block "<<gfg;
  return 0;
}
```

Output:
Before if-else block 0
if block 100
After if block 0

goto :

The goto statement in C/C++ also referred to as the unconditional jump statement can be used to jump from one point to another within a function.

- **What is the output of the following C++ program fragment:**

```
#include <stdio.h>

void printNumbers()
{
  int n = 1;
label:
  printf("%d ", n);
  n++;
  if (n <= 10){
    goto label;
  }

}

int main()
{
  printNumbers();
  return 0;
}
```

Output:
1 2 3 4 5 6 7 8 9 10

return :

The return in C or C++ returns the flow of the execution to the function from where it is called. This statement does not mandatorily need any conditional statements. As soon as the statement is executed, the flow of the program stops immediately and returns the control from where it was called. The return statement may or may not return anything for a void function, but for a non-void function, a return value must be returned.

- **What is the output of the following C++ program fragment:**

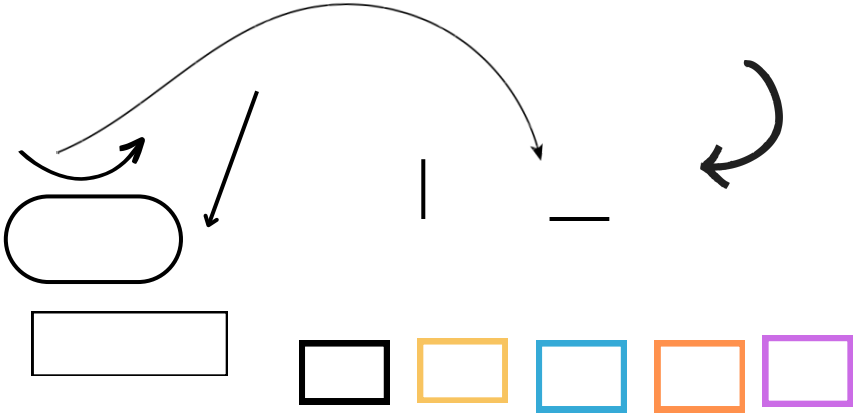
```
#include <stdio.h>

int SUM(int a, int b)
{
   int s1 = a + b;
   return s1;
}


void Print(int s2)
{
   printf("The sum is %d", s2);
   return;
}
```

```
int main()
{
   int num1 = 10;
   int num2 = 10;
   int sum_of = SUM(num1, num2);
   Print(sum_of);
   return 0;
}
```

Output:
The sum is 20

Operator Precedence and Associativity:

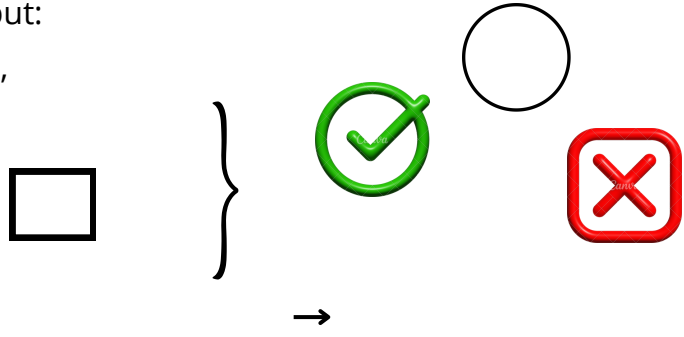
- **Make a program that can calculate area of triangle.**



Output:

Here,

Jim



- **What is the output of the following C++ program fragment:**

Usage of C++ :

7. switch-case

Switch is a multiple choice selection statement. It will be executed according to user choice.
Each switch-case must include break keyword.
Sometime time we should use default keyword.

- **Write a program that read a digit and display its spelling.**

```
#include <iostream>
using namespace std;
int main()
{
  int digit;
  printf("Enter a digit :");
  scanf("%d",&digit);
  switch(digit)
  {
    case 0:
    cout<<"Zero\n";
    break;

    case 1:
    cout<<"One\n";
    break;

    case 2:
    cout<<"Two\n";
    break;
```

```
    case 3:
    cout<<"Three\n";
    break;

    case 4:
    cout<<"Four\n";
    break;

    case 5:
    cout<<"Five\n";
    break;

    case 6:
    cout<<"Six\n";
    break;

    case 7:
    cout<<"Seven\n";
    break;

    case 8:
    cout<<"Eight\n";
    break;

    case 9:
    cout<<"Nine\n";
    break;

    default:
    cout<<"Not a valid digit\n";
    }
    return 0;
}
```

Output:
Enter a digit :8
Eight

- **What will be the output of the program given bellow.**

```
#include <iostream>
using namespace std;
int main()
{
  int a = 100;
  int b = 200;
  switch (a)
  {
  case 100:
    cout<<"Value of a is 100\n";
    switch (b)
    {
    case 200:
      cout<<"Value of b is 200\n";
    }
  }
  cout<<"Exact value of a is : "<<a<<endl;
  cout<<"Exact value of b is : "<<b<<endl;
  return 0;
}
```

Output:

```
Value of a is 100
Value of b is 200
Exact value of a is : 100
Exact value of b is : 200
```

Here,
It is possible to have a switch as part of the statement sequence of an outer switch. Even if the case constants of the inner and outer switch contain common values, no conflicts will arise.

8. while

A while loop in C++ programming repeatedly executes a target statement as long as a given condition is true.

- **Make a program that will show 1 to 10.**

```
#include <iostream>
using namespace std;
int main()
{
  int n = 1; // initialization
  while (n <= 10) // condition
  {
    printf("%d\n", n); // statements
    n++; // increment/decrement
  }
  return 0;
}
```

or,

```
#include <iostream>
using namespace std;
int main()
{
  int n=1;
  while (n<=100)
  {
    printf("%d\n",n);
    n++;
    if (n>10)
```

```
   {
     break;
   }

  }

  return 0;
}
```

Here,
Break is a statement which is used to left the loop.

- **What will be the output of the program given bellow.**

```
#include <iostream>
using namespace std;
int main()
{
  int n=1;
  while(n<=10){
    printf("%d\n",n);
  }
    n++;
    return 0;
}
```

Here,
We can see on the output that 1 is printing again and again.

- **Make a program that will show your name again and again, I mean infinity times.**

```
#include <iostream>
using namespace std;
int main()
{
  while(1==1){
    printf("Hamim\n");
  }
  return 0;
}
```

- **Make a program that will show odd number 1 to 10.**

```
#include <iostream>
using namespace std;
int main()
{
  int n=0;
  while (n<10)
  {
    n++;
    if (n%2==0)
    {
      continue;
    }
    printf("%d\n",n);

    }

    return 0;
}
```

Here,
Continue is a statement which is used to omit any of elements from the loop.

- **Make a program that will show multiplication table of 5.**

```
#include <iostream>
using namespace std;
int main()
{
  int n=5,i=1;
  while(i<=10)
  {
    printf("%dX%d=%d\n",n,i,n*i);
    i=i+1;
  }
  return 0;
}
```

Here
Increment : i=i+1/i++/++i/i+=1
Decrement : i=i-1/i--/--i/i-=1

- **Make a program that will show given bellow.**
  When i = 1
  j = 0, j = 1, j = 2, j = 3, j = 4,

  When i = 2
  j = 0, j = 1, j = 2, j = 3, j = 4,

  When i = 3
  j = 0, j = 1, j = 2, j = 3, j = 4,

  When i = 4
  j = 0, j = 1, j = 2, j = 3, j = 4,

```
#include <iostream>
using namespace std;
void main()
{
  int i = 0;
  while (i < 5) /* Outer loop */
  {
    printf("When i = %d\n", i);
    int j = 0;
    while (j < 5) /* Inner loop */
    {
      printf("j = %d, ", j);
      j++;
    }
    printf("\n\n");
    i++;
  }
}
```

9. do while

Unlike for and while loops, which test the loop condition at the top of the loop, the do...while loop in C++ programming checks its condition at the bottom of the loop.

A do...while loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.
The syntax of a do...while loop in C++ programming language is −

```
do {
   statement(s);
} while( condition );
```

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop executes once before the condition is tested.
If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.

- **Make a program that will print 1 to 10 using do while loop.**

```
#include <iostream>
using namespace std;
int main()
{
  int i=1;  // initialization
  do{
    printf("%d\n",i); //statements
    i++;  // increment/decrement
  }while(i<=10);  // condition
  return 0;
}
```

- **Make a program that will print 10 to 19 using do while loop.**

```
#include <iostream>
using namespace std;
int main()
{
 int a = 10;
 do
 {
  printf("value of a: %d\n", a);
  a = a + 1;
 }

 while (a < 20);

 return 0;
}
```

- **Make a program that will show given bellow.**
  When i = 1
  j = 0, j = 1, j = 2, j = 3, j = 4,

  When i = 2
  j = 0, j = 1, j = 2, j = 3, j = 4,

  When i = 3
  j = 0, j = 1, j = 2, j = 3, j = 4,

  When i = 4
  j = 0, j = 1, j = 2, j = 3, j = 4,

```
#include <iostream>
using namespace std;
void main()
{
  int i = 0;
  do
  {
    printf("When i = %d\n", i);
    int j=0;
    do
    {
      printf("j = %d, ", j);
      j++;
    } while (j < 5);/* Inner loop */
    printf("\n\n");
    i++;
  } while (i < 5);/* Outer loop */
}
```

10. for

The **for loop in C language** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like the array and linked list.

- **Make a program that will show multiplication table of 5 using for loop.**

```
#include <iostream>
using namespace std;
int main()
{
 int n=5,i;
 for(i=1; i<=10; i=i+1) // initialization; condition;
increment/decrement
 {
  printf("%dX%d=%d\n",n,i,n*i); // statements
 }
 return 0;
}
```

or,

```
#include <iostream>
using namespace std;
int main()
{
  int n=5,i=1;
  for(; i<=10; i=i+1)
  {
    printf("%dX%d=%d\n",n,i,n*i);
  }
  return 0;
}
```

or,

```
#include <iostream>
using namespace std;
int main()
{
  int n=5,i=1;
  for( ; ; )
  {
    printf("%dX%d=%d\n",n,i,n*i);
    i++;
    if (i>10)
    {
      break;
    }

  }
  return 0;
}
```

Output:

```
5X1=5
5X2=10
5X3=15
5X4=20
5X5=25
5X6=30
5X7=35
5X8=40
5X9=45
5X10=50
```

- **Make a program that will show multiplication table of 1 to 10.**

```
#include <iostream>
using namespace std;
int main()
{
  int i;
  for (int n=1;n<=10;n++)
  {
    for ( i = 1; i <10; i++)
    {
      printf("%dX%d=%d\n",n,i,n*i);
    }

  }
  return 0;
}
```

- **Make a program that will show given bellow.**

  1 1 1
  2 3 4
  3 5 7
  4 7 10
  5 9 13
  6 11 16
  7 13 19
  8 15 22
  9 17 25
  10 19 28

```
#include <iostream>
using namespace std;
int main()
{
  int i, j, k;

  for (i=1, j=1, k=1; i <= 10; i++, j +=2, k+=3)
  {
    cout<<i<<" "<<j<<" "<<k<<" "<<endl;
  }

  return 0;
}
```

- **Make a program that will show permutation of 1,2,3 and the output will be**
  1, 2, 3
  1, 3, 2
  2, 1, 3
  2, 3, 1
  3, 1, 2
  3, 2, 1

```
#include <iostream>
using namespace std;
int main()
{
  int a, b, c;
  for (a = 1; a <= 3; a++) {
    for (b = 1; b <= 3; b++) {
    if (b != a) {
      for (c = 1; c <= 3; c++) {
      if (c != b && c != a){
      printf ("%d, %d, %d\n", a, b, c);
                }
           }
        }
     }
  }
  return 0;
}
```

or,

```
#include <iostream>
using namespace std;
int main()
{
  int a, b, c;
  for (a = 1; a <= 3; a++) {
   for (b = 1; b <= 3; b++) {
    for (c = 1; c <= 3; c++) {
    if(b != a && c != a && c != b) {
    printf ("%d, %d, %d\n", a, b, c);
       }
     }
    }
  }
  return 0;
}
```

- **Make a program that will show given bellow.**

  When i = 1
  j = 0, j = 1, j = 2, j = 3, j = 4,

  When i = 2
  j = 0, j = 1, j = 2, j = 3, j = 4,

  When i = 3
  j = 0, j = 1, j = 2, j = 3, j = 4,

  When i = 4
  j = 0, j = 1, j = 2, j = 3, j = 4,

```
#include <iostream>
using namespace std;
int main()
{
 int i, j;
 for (i = 0; i < 5; i++) /* Outer loop */
 {
  printf("When i = %d\n", i);
  for (j = 0; j < 5; j++) /* Inner loop */
  {
   printf("j = %d, ", j);
  }
  printf("\n\n");
 }
}
```

- **Make a program that will calculate power(base, exponent) using for loop.**

```
#include <iostream>
using namespace std;
int main()
{
  double base,exp,result=1,i;
  printf("Enter base : ");
  scanf("%lf",&base);
  printf("Enter exponent : ");
  scanf("%lf",&exp);
  for(i=1;i<=exp;i++){
    result=result*base;
  }
  printf("%.1lf\n",result);
  return 0;
}
```

- **1+2+3+4+.....+100 solve it.**

```
#include <iostream>
#include<conio.h>
using namespace std;
int main()
{
  int i,s=0;
  for(i=1;i<=100;i++){
    s=s+i;
  }
  printf("%d",s);
  getch();
}
```

- **1+2+3+4+.....+n solve the problem.**

```
#include <iostream>
#include<conio.h>
using namespace std;
main()
{
    int i,n,s=0;
    scanf("%d",&n);
    for(i=1;i<=n;i++){
        s=s+i;
    }
    printf("%d",s);
    getch();
}
```

- **1+3+5+7+.....+n solve it.**

```
#include <iostream>
#include<conio.h>
using namespace std;
int main()
{
   int i,n,s=0;
   scanf("%d",&n);
   for(i=1;i<=n;i=i+2){
      s=s+i;
   }
   printf("%d",s);
   getch();
}
```

- **2+4+8+16+.....+n solve it.**

```
#include <iostream>
#include<conio.h>
using namespace std;
int main()
{
   int i,n,s=0;
   scanf("%d",&n);
   for(i=2;i<=n;i=i*2){
      s=s+i;
   }
   printf("%d",s);
   getch();
}
```

- **2^1+2^2+2^3+2^4+.....+2^n solve it.**

```
#include <iostream>
#include<conio.h>
#include<math.h>
using namespace std;
int main()
{
   int i,n;
   long int s=0;
   scanf("%d",&n);
   for(i=1;i<=n;i++){
      s=s+pow(2,i);
   }
   printf("%ld",s);
   getch();
}
```

Here,
Library function = pow(a,b) = a^b

- **1^2+2^2+3^2+4^2+.....+n^2 solve it.**

```
#include <iostream>
#include<conio.h>
#include<math.h>
using namespace std;
int main()
{
   int i,n;
   long int s=0;
   scanf("%d",&n);
   for(i=1;i<=n;i++){
      s=s+pow(i,2);
   }
   printf("%ld",s);
   getch();
}
```

- **1^2+3^2+5^2+7^2+.....+n^2 solve it.**

```
#include <iostream>
#include<conio.h>
#include<math.h>
using namespace std;
int main()
{
   int i,n;
   long int s=0;
   scanf("%d",&n);
   for(i=1;i<=n;i+=2){
      s=s+pow(i,2);
   }
   printf("%ld",s);
   getch();
}
```

- **100+99+98+.....+n solve it.**

```
#include <iostream>
#include<conio.h>
using namespace std;
int main()
{
   int i,n,s=0;
   scanf("%d",&n);
   for(i=100;i>=n;i--){
      s=s+i;
   }
   printf("%d",s);
   getch();
}
```

- **n+(n-1)+(n-2)+.....+1 solve it.**

```
#include <iostream>
#include<conio.h>
using namespace std;
int main()
{
   int i,n,s=0;
   scanf("%d",&n);
   for(i=n;i>=1;i--){
      s=s+i;
   }
   printf("%d",s);
   getch();
}
```

- **1-2+3-4+5-.....+n solve it.**

```
#include <iostream>
#include<math.h>
#include<conio.h>
using namespace std;
int main()
{
   int i,n,s=0;
   scanf("%d",&n);
   for(i=1;i<=n;i++){
      s=s+pow(-1,i+1)*i;
   }
   printf("%d",s);
   getch();
}
```

- **-1+2-3+4-5+.....+n solve it.**

```
#include <iostream>
#include<math.h>
#include<conio.h>
using namespace std;
int main()
{
   int i,n,s=0;
   scanf("%d",&n);
   for(i=1;i<=n;i++){
      s=s+pow(-1,i)*i;
   }
   printf("%d",s);
   getch();
}
```

- **5! = 1*2*3*4*5 solve it.**

```
#include <iostream>
#include<conio.h>
using namespace std;
int main()
{
   int i,f=1;
   for(i=1;i<=5;i++){
      f=f*i;
   }
   printf("%d",f);
   getch();
}
```

- **5! = 5*4*3*2*1 solve it.**

```
#include <iostream>
#include<conio.h>
using namespace std;
int main()
{
   int i,f=1;
   for(i=5;i>=1;i--){
      f=f*i;
   }
   printf("%d",f);
   getch();
}
```

- **n! = 1*2*3*.....*(n-1)*n solve it.**

```
#include <iostream>
using namespace std;
int main()
{
   int i,n,f=1;
   scanf("%d",&n);
   for(i=1;i<=n;i++){
      f=f*i;
   }
   printf("%d",f);
}
```

or,

```
#include <iostream>
using namespace std;
int main()
{
   int i,n,f=1;
   scanf("%d",&n);
   if(n>=0){
   for(i=1;i<=n;i++){
      f=f*i;
   }
   printf("%d",f);
   }
   else{
      printf("Invalid Input");
   }
}
```

11. goto

A goto statement in C++ language provides an unconditional jump from the goto to a labeled statement in the function.
The given label must be reside in the same function.
goto label;
..
.
label : statement;

- **What will be the output of the program given bellow.**

```
#include <iostream>
using namespace std;
int main()
{
  int a = 10;
loop:
  do
  {
    if (a == 15)
    {
      a = a + 1;
      goto loop;
    }
    printf("Value of a: %d\n", a);
    a++;
  } while (a < 20);
  return 0;
}
```

Output:

```
Value of a: 10
Value of a: 11
Value of a: 12
Value of a: 13
Value of a: 14
Value of a: 16
Value of a: 17
Value of a: 18
Value of a: 19
```

12. Function

A function is a group of statements that together perform a task. Every C++ program has at least one function, which is main(), and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.
A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.
The C++ standard library provides numerous built-in functions that your program can call. For example, strcat() to concatenate two strings, memcpy() to copy one memory location to another location, and many more functions.
A function can also be referred as a method or a sub-routine or a procedure, etc.

Defining a Function :
The general form of a function definition in C++ programming language is as follows −

```
return_type   function_name( parameter list ) {
  body of the function
}
```

A function definition in C++ language consists of a function header and a function body. Here are all the parts of a function −

Return Type − A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.

Function Name − This is the actual name of the function. The function name and the parameter list together constitute the function signature.

Parameters/arguments − A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

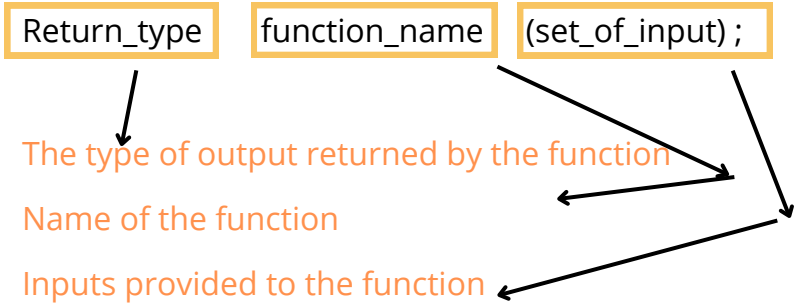
Function Body − The function body contains a collection of statements that define what the function does.

Basic of Function:

Definition:

Function is basically a set of statements that takes inputs, perform some computation and products output.

Syntax:



Why Functions ?

There are two important reasons of why we are using function:

1. Reusability:
Once the function is defined, it can be reused over and over again.
2. abstraction:
If you are just using the function in your program then you don't have to worry about how it works inside!

example: scanf function

- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;

int areaOfRect(int length, int breadth)
{
 int area;
 area = length * breadth;
 return area;
}

int main()
{
 int l=10, b=5;
 int area = areaOfRect(l, b);
 cout<<area<<endl;
 return 0;
}
```

Output:
50

- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;

int areaOfRect(int length, int breadth)
{
  int area;
  area = length * breadth;
  return area;
}
int main()
{
  int l=10, b=5;
  int area = areaOfRect(l, b);
  cout<<areaOfRect(10,5)<<endl;

  l=50, b=20;
  area = areaOfRect(l, b);
  printf("%d\n",area);
}
```

Output:

50
1000

Function Declaration:

As we already know , when we declared a variable, we declare its properties to the compiler.

For example: int var;
properties:

1. Name of variable: var
2. Type of variable: int

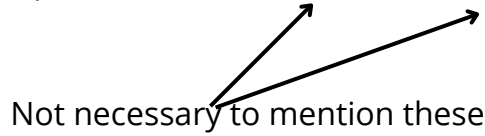
Similarly, function declaration (also called function prototype) means declaring the properties of a function to the compiler.

For example: int fun(int, char);
Properties:

1. Name of function: fun
2. Return type of function: int
3. Number of parameters: 2
4. Type of parameter 1: int
5. Type of parameter 2: char

It is not necessary to put the name of the parameters in function.

For example: int fun (int var1, char var2);



- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;
char fun();  // Function prototype
int main()
{
  char c = fun();
  printf("Character is : %c",c);
}
char fun()
{
  return 'a';
}
```

Output:
Character is : a

Here,
Declare the function before using it is not necessary but it is preferred to declare before using it.

or,

```
#include <iostream>
using namespace std;
char fun()
{
  return 'a';
}

int main()
{
 char c = fun();
 printf("Character is : %c",c);
}
```

- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;
int main()
{
  char c = fun();
  printf("Character is : %c",c);
}
char fun()
{
  return 'a';
}
```

Output:

```
warning: implicit declaration of function 'fun' [-
Wimplicit-function-declaration]
   char c = fun();
            ^~~
At top level:
 error: conflicting types for 'fun'
 char fun()
      ^~~
note: previous implicit declaration of 'fun' was here
   char c = fun();
            ^~~
```

Function definition:

Function definition consists of block of code which is capable of performing some specific task.

For example:

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

Here,

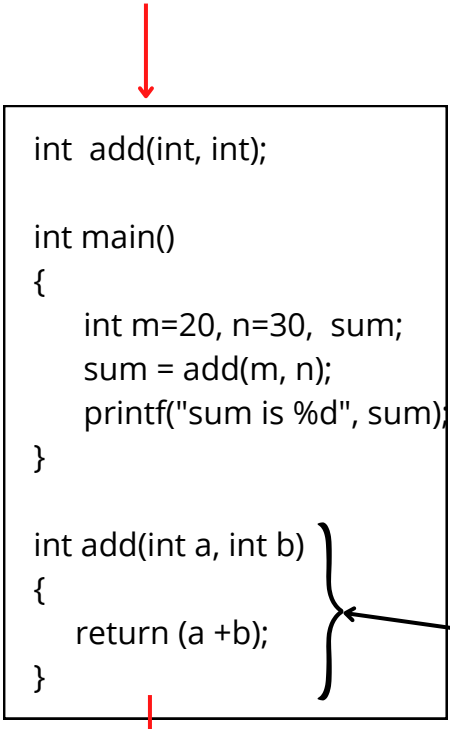
```
int  add(int, int);

int main()
{
     int m=20, n=30,  sum;
     sum = add(m, n);
     printf("sum is %d", sum);
}

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

This is the way you call a function.

Note: while calling a function, you should not mention the return type of the function.
Also you should not mention the data types of the arguments.

```
int  add(int, int);

int main()
{
      int m=20, n=30,  sum;
      sum = add(m, n);
      printf("sum is %d",
sum);
}

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

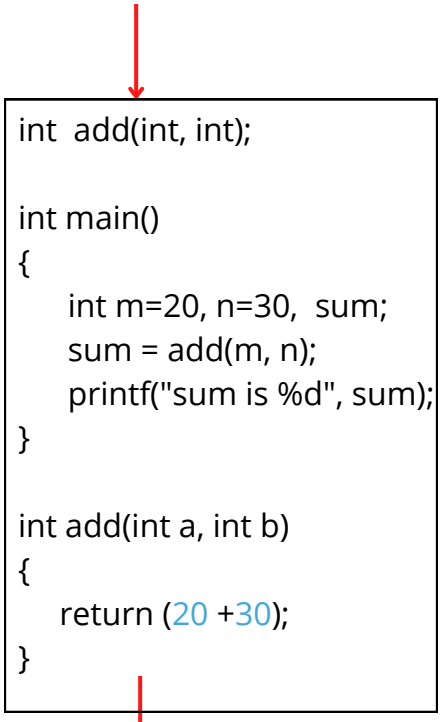
```
int  add(int, int);

int main()
{
      int m=20, n=30,  sum;
      sum = add(m, n);
      printf("sum is %d", sum);
}

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

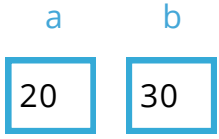


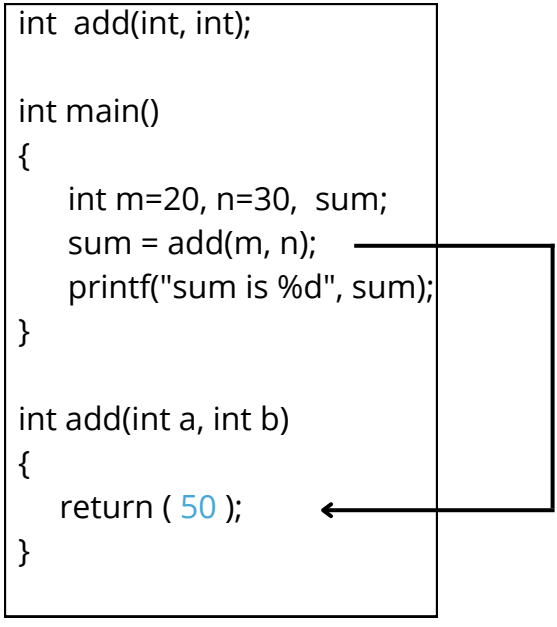
This is the way how you define a function.
Note: it is important to mention both data type and name of parameters.

```
int  add(int, int);

int main()
{
     int m=20, n=30,  sum;
     sum = add(m, n);
     printf("sum is %d", sum);
}

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

a	b
20	30


int add(int, int);
int main()
{
int m=20, n=30, sum;
sum = add(m, n);
printf("sum is %d", sum);
}
int add(int a, int b)
{
return (20 +30);
}


a
b
20
30


int add(int, int);
int main()
{
int m=20, n=30, sum;
sum = add(m, n);
printf("sum is %d", sum);
}
int add(int a, int b)
{
return (50);
}

```
int  add(int, int);

int main()
{
      int m=20, n=30,  sum;
      sum = 50;
      printf("sum is %d", 50);
}

int add(int a, int b)
{
     return ( 50 );
}
```

or,

```
#include<stdio.h>
int  add(int, int);

int main()
{
      int m=20, n=30,  sum;
      sum = add(m, n);
      printf("sum is %d", sum);
}

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

Output;
sum is 50

Local variable and Global variable:

In general, the scope is defined as the extent up to which something can be worked with. In programming also the scope of a variable is defined as the extent of the program code within which the variable can be accessed or declared or worked with.

There are mainly two types of variable scopes:

1. Local Variables
2. Global Variables

```
#include<iostream>
using namespace std;
```
Global Variable
```
// global variable
int global = 5;

// main function
int main()
```
Local variable
```
{
    // local variable with same
    // name as that of global variable
    int global = 2;

    cout << global << endl;
}
```

Output:
2

Local Variables :

Variables defined within a function or block are said to be local to those functions.

- Anything between '{' and '}' is said to inside a block.
- Local variables do not exist outside the block in which they are declared, i.e. they can not be accessed or used outside that block.
- Declaring local variables: Local variables are declared inside a block.

- **What will be the output of the program given bellow?**

```
#include <iostream>
using namespace std;
int main()
{
  // x is a local variable
  int x = 10 ;
  cout<<"Inside the main function x = "<< x
<<endl;
  return 0;
}
```

Output:
Inside the main function x = 10

- **What will be the output of the program given bellow?**

```
#include <iostream>
using namespace std;
void display()
{
  cout<<"Inside the display function x = "<< x <<endl;
}

int main()
{
  // x = Local variable
  int x = 10 ;
  display();
  return 0;
}
```

Output:

```
Earth.cpp: In function 'void display()':
Earth.cpp:5:46: error: 'x' was not declared in this scope
   cout<<"Inside the display function x = "<< x <<endl;
                                              ^
```

Global Variables :

As the name suggests, Global Variables can be accessed from any part of the program.

- They are available through out the life time of a program.
- They are declared at the top of the program outside all of the functions or blocks.
- Declaring global variables: Global variables are usually declared outside of all of the functions and blocks, at the top of the program. They can be accessed from any portion of the program.
- **What will be the output of the program given bellow?**

```
#include <iostream>
using namespace std;
int x = 10; // x = global variable

void display()
{
  x = 4;
  cout << "Inside the display function x = " << x << endl;
}

int main()
{
  display();
  cout << "In main function x = " << x;
  return 0;
}
```

Output:
Inside the display function x = 4
In main function x = 4

- **What will be the output of the program given bellow?**

```
#include <iostream>
using namespace std;
int x = 10; // x = global variable

void display()
{
  cout << "Inside the display function x = " << x << endl;
}

int main()
{
  int x = 50; // x = local variable
  cout << "Inside the main function x = " << x << endl;
  display();
  return 0;
}
```

Output:
Inside the main function x = 50
Inside the display function x = 10

- **What will be the output of the program given bellow?**

```
#include <iostream>
using namespace std;
int x = 10; // x = global variable

void display()
{
  cout << x << endl;
}

int main()
{
  int x = 50; // x = local variable
  cout << x << endl;
  return 0;
}
```

Output:
50

Here,
Local variable has printed because in every function local variables' priority is higher than global variables and all programs starts from main() function.

scope resolution operator(::) :

scope resolution operator(::) is used to print global variables in every function.

- **What will be the output of the program given bellow?**

```
#include <iostream>
using namespace std;
int x = 10; // x = global variable

void display()
{
  cout << x << endl;
}

int main()
{
  int x = 50; // x = local variable
  cout << :: x << endl;
  return 0;
}
```

Output:
10

- **What will be the output of the program given bellow?**

```
#include <iostream>
using namespace std;

int x = 10; // x = global variable

int main()
{
  int x = 50; // x = local variable
  :: x = 20;
  cout << ::x << endl;
  return 0;
}
```

Output:
20

- **What will be the output of the program given bellow?**

```
#include <iostream>
using namespace std;

int x = 10; // x = global variable
void display()
{
  cout<<x<<endl;
}
int main()
{
  int x = 50; // x = local variable
```

```
  display();

  :: x = 20;
  cout << ::x << endl;
  display();
  return 0;
}
```

Output:
10
20
20

- **What will be the output of the program given bellow?**

```
#include <iostream>
using namespace std;

int x = 10; // x = global variable

int main()
{
  :: x = 20;
  cout << ::x << endl;
  return 0;
}
```

Output:
20

- **What will be the output of the program given bellow?**

```
#include <iostream>
using namespace std;
int x = 10; // x = global variable

int main()
{
  :: x = 20;
  cout << x << endl;
  return 0;
}
```

Output:

20

- **What will be the output of the program given bellow?**

```
#include <iostream>
using namespace std;

int x = 10; // x = global variable

int main()
{
  int x = 50; // x = local variable
  :: x = 20;
  cout << x << endl;
  return 0;
}
```

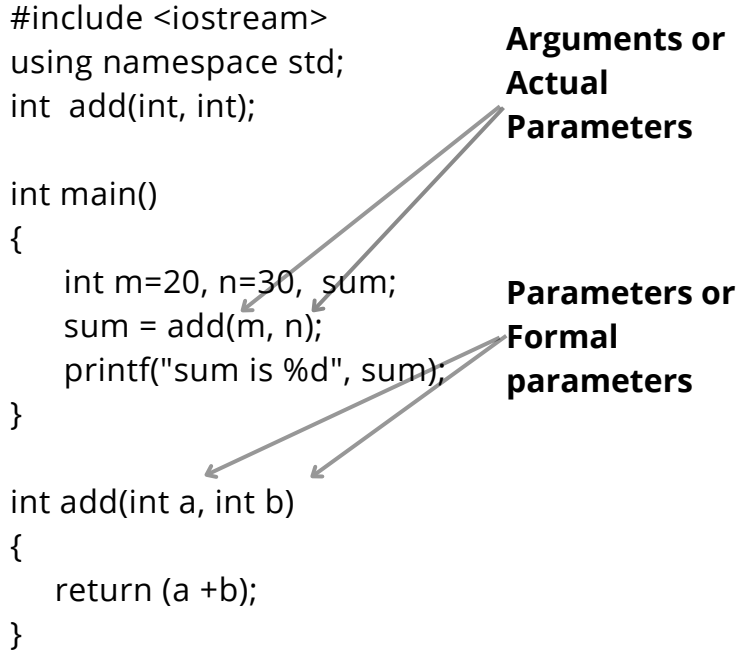
Output:

50

Difference between an Argument and Parameter:

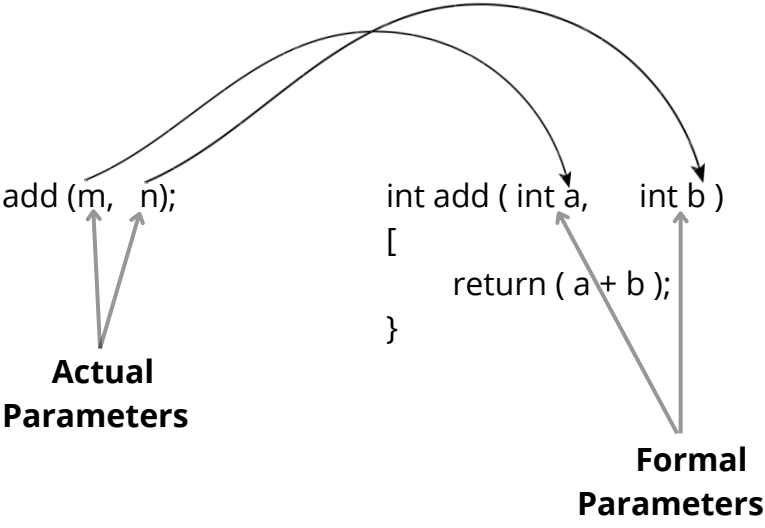
Parameter: is a variable in the declaration of the function.
Argument: is the actual value of the parameter that gets passed to the function.

Note: Parameter is also called as Formal Parameter and Argument is also called as Actual parameter.



Recall:

Actual Parameters: The parameters passed to a function.
Formal Parameters: the parameters received by a function.



Type of Function call

While calling a function, there are two ways that arguments can be passed to a function:

Call Type	Description
Call by value	This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
Call by reference	This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

Call by Value:

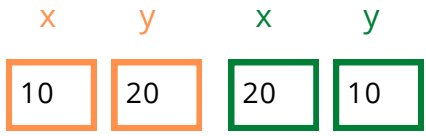
Here values of actual parameters will be copied to formal parameters and these two different parameters store values in different locations.

- In this case, changes made to the parameter inside the function have no effect on the argument.
- By default, C++ programming language uses call by value method to pass arguments. In general, this means that code within a function cannot alter the arguments used to call the function. Consider the function swap() definition as follows.

```
int x = 10, y = 20;
fun(x, y);
printf("x = %d, y = %d,x,y);
```

```
int fun(int x, int y)
{
   x=20;
   y=10;
}
```

Output:
x=10, y=20



- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;

void swap(int a, int b){
 int temp;
 temp = a;
 a = b;
 b = temp;
}

int main()
{
 int a = 100;
 int b = 200;

 printf("Before swap, value of a : %d\n",a);
 printf("Before swap, value of b : %d\n",b);

 swap(a, b);
```

```
  printf("After swap, value of a : %d\n", a);
  printf("After swap, value of a : %d\n", b);

}
```

Output::

```
Before swap, value of a : 100
Before swap, value of b : 200
After swap, value of a : 100
After swap, value of a : 200
```

Here,
The output shows that there is no change in the values though they had been changed inside the function.

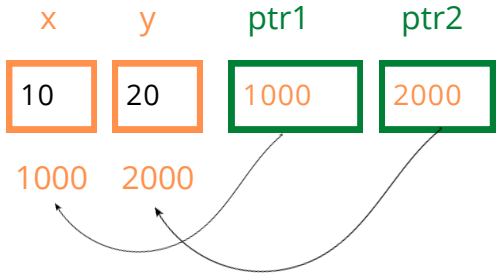
Call by Reference:

The call by reference method of passing arguments to a function copies the address of an argument into the formal parameter.

- Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the passed argument.
- To pass the value by call by reference, argument pointers are passed to the functions.
- To pass the value by reference, argument pointers are passed to the functions just like any other value.

```
int x = 10, y = 20;
fun(&x, &y);
printf("x = %d, y = %d,x,y);

Output:
x=20, y=10
```

```
int fun (int *ptr1, int *ptr2)
{
   *ptr1=20;
   *ptr2=10;
}
```



- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;

void swap(int *a, int *b){
 int temp;
 temp = *a;
 *a = *b;
 *b = temp;
}

int main()
{
  int a = 100;
  int b = 200;
```

```
  printf("Before swap, value of a : %d\n",a);
  printf("Before swap, value of b : %d\n",b);

  swap(&a, &b);

  printf("After swap, value of a : %d\n", a);
  printf("After swap, value of a : %d\n", b);

}
```

Output:

```
Before swap, value of a : 100
Before swap, value of b : 200
After swap, value of a : 200
After swap, value of a : 100
```

function templates :

- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;

template <class myTemplate>
myTemplate add(myTemplate a, myTemplate b)
{
  return a+b;
}

int main()
{
  cout<< add(10,20)<<endl;
  cout<< add(10.5,20.2)<<endl;
  return 0;
}
```

Output:
30
30.7

Here,
This method is used for the same type of parameters of the function.

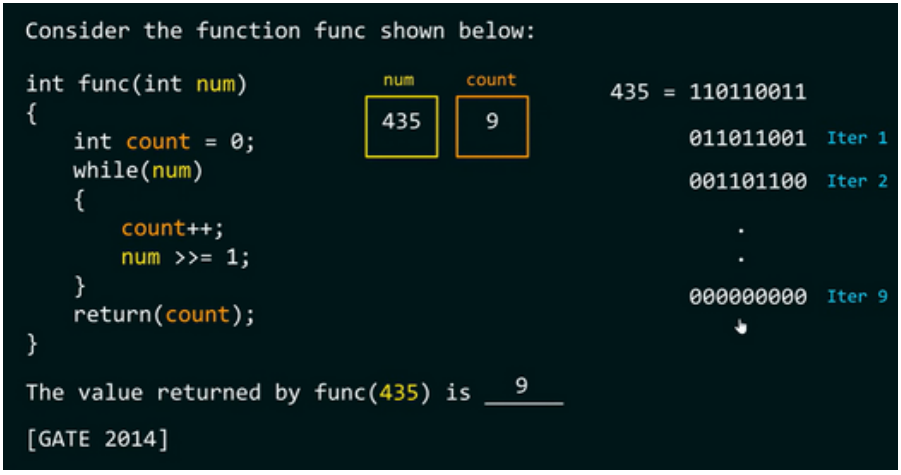
- **What is the output of the following C++ program fragment:**

```
#include<iostream>
using namespace std;

template <class myTemplate1, class myTemplate2>
myTemplate1 add(myTemplate1 a, myTemplate2 b)
{
  return a+b;
}

int main()
{
  cout<< add(10,20)<<endl;
  cout<< add(10.5,20)<<endl;
  return 0;
}
```

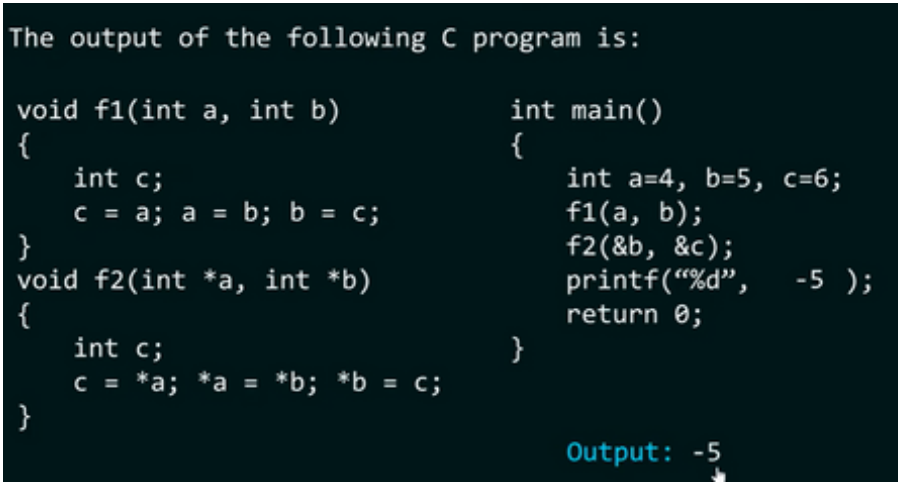
Output:
30
30.5

Here,
This method is used for the different types of parameters of the function.

- **What is the output of the following C++ program fragment:**



- **What is the output of the following C++ program fragment:**



- **What is the output of the following C++ program fragment:**

Consider the following C program:

```
int fun()
{
  static int num = 16;
  return num--;
}

int main()
{
  for(fun(); fun(); fun())
  printf("%d ", fun());
  return 0;
}
```

What is the output of the C program available in the LHS?

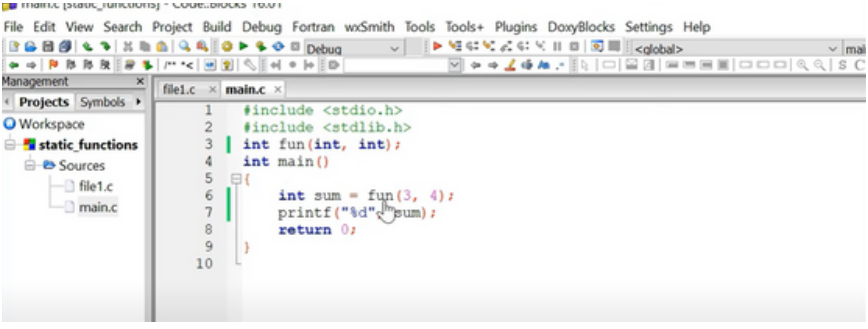
a) Infinite loop
b) 13 10 7 4 1
c) 14 11 8 5 2
d) 15 12 8 5 2

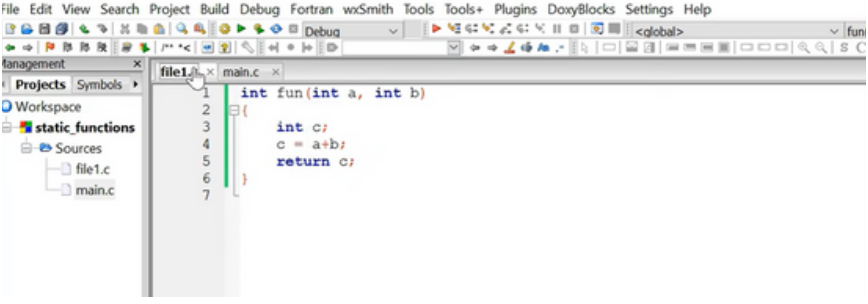
Static function in C:

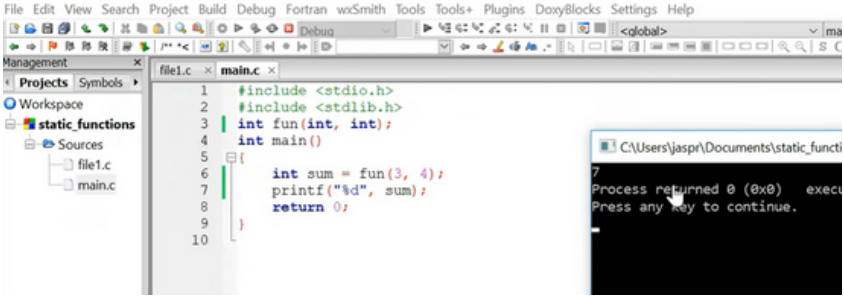
Basics:

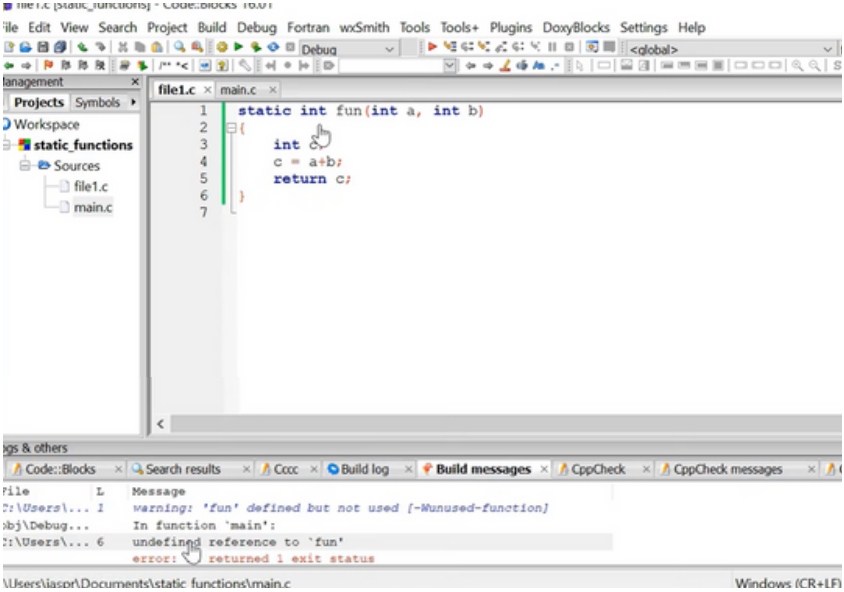
- In C++, functions are global by default.
- This means if we want to access the function outside from the file where it is declared, we can access it easily.
- now if we want to restrict this access, then we make our function static by simply putting a keyword static in front of the function.

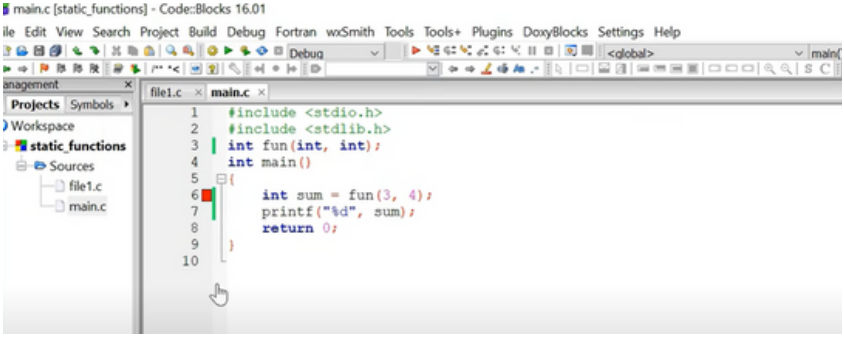
For example:






File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
<global>
Management
Projects Symbols
Workspace
static_functions
Sources
file1.c
main.c
file1.c main.c
#include <stdio.h>
#include <stdlib.h>
int fun(int, int);
int main()
{
int sum = fun(3, 4);
printf("%d", sum);
return 0;
}
C:\Users\jaspr\Documents\static_functi
7
Process returned 0 (0x0)
Press any key to continue.


File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
<global>
Projects Symbols
Workspace
static_functions
Sources
file1.c
main.c
file1.c main.c
static int fun(int a, int b)
{
int c;
c = a+b;
return c;
}
Code::Blocks Search results Cccc Build log Build messages CppCheck CppCheck messages
File L Message
warning: 'fun' defined but not used [-Wunused-function]
In function 'main':
undefined reference to 'fun'
error: returned 1 exit status
Windows (CR+LF)


main.c [static_functions] - Code::Blocks 16.01
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
<global>
Projects Symbols
Workspace
static_functions
Sources
file1.c
main.c
file1.c main.c
#include <stdio.h>
#include <stdlib.h>
int fun(int, int);
int main()
{
int sum = fun(3, 4);
printf("%d", sum);
return 0;
}

- **Make a program that will add two numbers using function.**

```
#include <iostream>
using namespace std;

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

int main()
{
  int num1,num2;
  printf("Enter two number : ");
  cin>>num1>>num2;

  cout<<"The sum is : "<<sum(num1,num2)<<endl;
  return 0;

}
```

Output:
Enter two numbers : 12 2
The sum is : 14

or,

```
#include <iostream>
using namespace std;
template <class myt>
myt sum(myt a, myt b){
  return a+b;
}
int main()
{
   double num1, num2;
   printf("Enter two number : ");
   cin>>num1>>num2;

   cout<<"The sum is : "<<sum(num1,num2)<<endl;
   return 0;
}
```

Output:

```
Enter two number : 1 2.3
The sum is : 3.3
```

or,

```
#include <iostream>
using namespace std;
int main()
{
  int num1, num2;
  printf("Enter two number : ");
  scanf("%d %d", &num1, &num2);

  printf("The sum is : %d\n", sum(num1, num2));
  return 0;
}

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

Here,
We will found an error that sum was not declared in this scope. But we don't need to warry about it because it has a solution. And the solution is we have to write the prototype of sum function before the main function. The prototype is int sum(int a, int b);

or,

```
#include <iostream>
using namespace std;
int sum(int a, int b);
int main()
{
  int num1, num2;
  printf("Enter two number : ");
  scanf("%d %d", &num1, &num2);

  printf("The sum is : %d\n", sum(num1, num2));
  return 0;
}

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

- **What will be the output of the program given bellow.**

```
#include <iostream>
using namespace std;
int test_function(int x)
{
   int y=x;
   x=2*y;
   return (x*y);
}

int main()
{
   int x=10,y=20,z=30;
   z=test_function(x);
   printf("%d %d %d\n",x,y,z);
   return 0;
}
```

Here,
The output will be 10 20 200 where we expect that the output will be 20 10 200. Why it is happening ? The reason is every functions have different variables. It is called local variable. We have print the value of x, y of main function but do not print the value of x, y of test_function. The existence of local variable of a function do not stay in other function.
If we want the existence of variable in all function then we have to declare global variable.

- **Make a program that will show sum of 1 and 2 using function.**

```
#include <iostream>
using namespace std;
int sum(int a, int b)
{
  return a+b;
}

int main()
{

  int result = sum(1,2);

  cout<<"The sum is : "<<result<<endl;
  return 0;

}
```

or,

```
#include <iostream>
using namespace std;
int sum(int a, int b)
{
 return a + b;
}

int main()
{
 cout << "The sum is : " << sum(1, 2) << endl;
 return 0;
}
```

- **Make a program that will show sum of 1,2 and 5,6 using function and also print 5 without function.**

```
#include <iostream>
using namespace std;
int sum(int a, int b)
{
 return a + b;
}
int main()
{
 int result = sum(1, 2);
 cout<<"The sum is : "<<result<<endl;

 result = sum(5, 6);
 cout<<"The sum is : "<<result<<endl;
 cout<<"The sum is : "<< 5 <<endl;
 return 0;
}
```

or,

```
#include <iostream>
using namespace std;
int sum(int a, int b)
{
 return a + b;
}
int main()
{
 cout<<"The sum is : "<<sum(1, 2)<<endl;
 cout<<"The sum is : "<<sum(5, 6)<<endl;
 cout<<"The sum is : "<< 5 <<endl;
 return 0;
}
```

- **Make a program that will show sum of 1,2,3 and 5,6,7 using function.**

```
#include <iostream>
using namespace std;

int sum(int a,int b,int c)
{
  return a+b+c;
}
int main()
{
  cout<<"The sum is : "<<sum(1,2,3)<<endl;
  cout<<"The sum is : "<<sum(5,6,7)<<endl;
  return 0;
}
```

or,

```
#include <iostream>
using namespace std;

int sum(int a,int b,int c)
{
  cout<<"The sum is : "<<a+b+c<<endl;
}
int main()
{
  sum(1,2,3);
  sum(5,6,7);
  return 0;
}
```

- **Make a program that will show sum of 1,2,3 using function.**

```
#include <iostream>
using namespace std;

int sum(int a,int b,int c)
{
  cout<<"The sum is : "<<a+b+c<<endl;
}
int main()
{
  sum(1,2,3);
  return 0;
}
```

or,

```
#include <iostream>
using namespace std;

void sum(int a,int b,int c)
{
  cout<<"The sum is : "<<a+b+c<<endl;
}
int main()
{
  sum(1,2,3);
  return 0;
}
```

Here,
void is used for return nothing from the function.

- **What is the output of the following C++ program fragment:**

```
#include <iostream>
using namespace std;

int sum(int a,int b,int c)
{
  cout<<"The sum is : "<<a+b+c<<endl;
  return 0;
}
int main()
{
  sum(1,2,3);
  cout<<sum(1,2,3);

  return 0;
}
```

Output:

```
The sum is : 6
The sum is : 6
0
```

- **Make a program that will show sum of 1,2,3 and 5,6,7 and also subtraction of 5,4 using function.**

```
#include <iostream>
using namespace std;

void sum(int a,int b,int c)
{
 cout<<"The sum is : "<<a+b+c<<endl;
}

void sub (int a,int b)
{
 cout<<"The subtraction is : "<<a-b<<endl;
}
int main()
{
  sum(1,2,3);
  sum(5,6,7);
  sub (5,4);
  return 0;
}
```

Output:

```
The sum is : 6
The sum is : 18
The subtraction is : 1
```

- **Make a program that will show square of any integer number using function.**

```
#include <iostream>
using namespace std;
int square(int a)
{
  return a*a;
}
int main()
{
  int num;
  cout<<"Enter any integer number : ";
  cin>>num;
  int result=square(num);
  cout<<"Square is : "<<result;
  return 0;
}
```

or,

```
#include <iostream>
using namespace std;
int square(int a)
{
 return a * a;
}
int main()
{
 int num;
 cout << "Enter any integer number : ";
 cin >> num;
 cout << "Square is : " << square(num);
 return 0;
}
```

- **Make a program that will show area of a triangle using function.**

```
#include <iostream>
using namespace std;
double area_of_traingle(double base, double height);
int main()
{
 double base, height, result;
 cout<<"Enter base of the traingle: ";
 cin>>base;
 cout<<"Enter base of the traingle: ";
 cin>>height;
 result = area_of_traingle(base, height);
 cout<<"Area of traingle is: "<<result<<endl;
}
double area_of_traingle(double base, double height)
{
 return .5*base*height;
}
```

or,

```
#include <iostream>
using namespace std;
double area_of_traingle(double base, double
height)
{
 return .5*base*height;
}
int main()
{
 double base, height, result;
```

```
 cout<<"Enter base of the traingle: ";
 cin>>base;
 cout<<"Enter base of the traingle: ";
 cin>>height;
 result = area_of_traingle(base, height);
 cout<<"Area of traingle is: "<<result<<endl;
}
```

- **Make a program that will calculate power(base, exponent) using function.**

```
#include <iostream>
using namespace std;
double power(double a, int b)
{
 double result=1, i;
 for(i=1;i<=b;i++){
  result=result*a;
 }
 return result;
}
int main()
{
 double base, Ans;
 int exponent;
 cout<<"Enter base in double and exponent in
integer: ";
 cin>>base>>exponent;
 Ans =power(base,exponent);
 cout<<"Ans: "<<Ans<<endl;
 return 0;
}
```

or,

```
#include <iostream>
using namespace std;
void power(double base, double exp)
{
  double result=1,i;
  for(i=1;i<=exp;i++){
   result=result*base;
  }
  cout<<result<<endl;
}
int main()
{
  power(2,3);
  return 0;
}
```

- **Make a program that will show given bellow using function.**

  2.0

  4.0

  8.0

```
#include <iostream>
using namespace std;
void power(double base, double exp)
{
   double result=1,i;
   for(i=1;i<=exp;i++){
      result=result*base;
   }
   cout<<result<<endl;
}
```

```
 int main()
 {
   power(2,1);
   power(2,2);
   power(2,3);
   return 0;
}
```

- **Make a program that will show Passing Array to function.**

```
#include <iostream>
using namespace std;

void display(int a[])
{
  int i;
  for(i=0;i<=4;i++){
  cout<<a[i]<<" ";
  }
}
int main()
{
 int num[]={10,20,30,40,50};
 display(num);
  return 0;
}
```

- **Make a program that will show maximum value from an array using function.**

```
#include <iostream>
using namespace std;

int maximum(int a[5])
{
 int i;
 int max = a[0];

  for (i = 1; i <= 4; i++)
  {
   if (max < a[i])
   {
    max = a[i];
   }
  }
  return max;
}
int main()
{
 int num[5] = {10, 20, 30, 40, 50};
 int maximumValue = maximum(num);
 cout<<"Maximum value is "<<maximumValue;
 return 0;
}
```

or,

```
#include <iostream>
using namespace std;
float maximum(float a[], int n)
{
 float max;
 for (int i = 0; i < n; i++)
 {
  if (i == 0)
  {
   max = a[0];
  }
  if (max < a[i])
  {
   max = a[i];
  }
 }
 return max;
}
int main()
{
 int n, i;
 cout<<"How many number do you want check: ";
 cin>>n;
 float num[n];
 for (i = 0; i < n; i++)
 {
  cin>>num[i];
 }
 cout<<"maximum number = "
<<maximum(num,n);
}
```

- **Make a program that will show Passing String to function.**

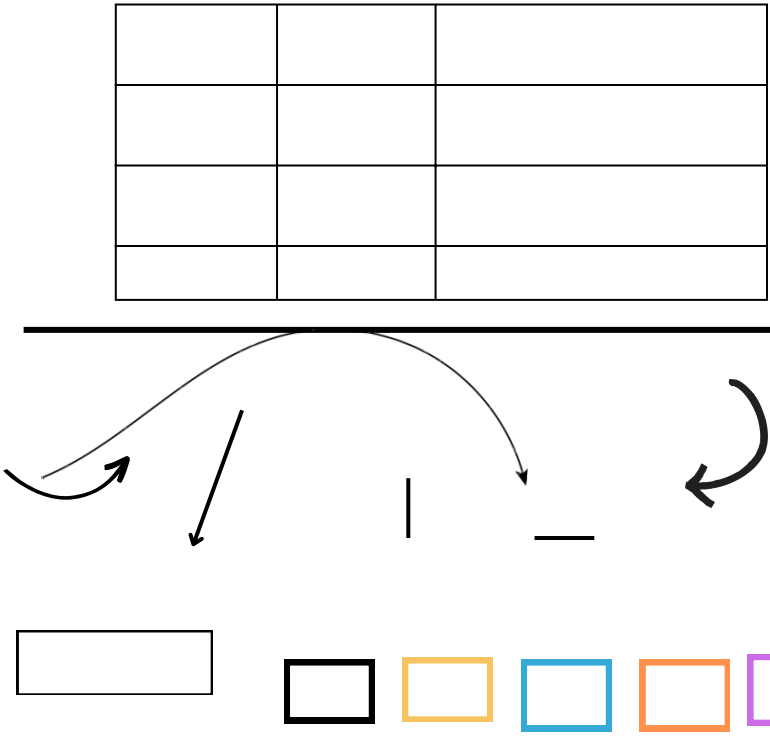
```
#include <iostream>
using namespace std;

void display(char str2[]){
 int i = 0;
 while (str2[i] != '\0')
 {
  cout << str2[i] << endl;
  i++;
 }
}
int main()
{
 char str1[] = "Hamim";
 display(str1);
 return 0;
}
```

- **Make a program that can copy String using function.**

```
#include <iostream>
using namespace std;
void strcpying (char *t[], char *s[])
{
    int i;
    for(i=0;s[i]!='\0';i++){
        t[i] = s[i];
    }
}
```

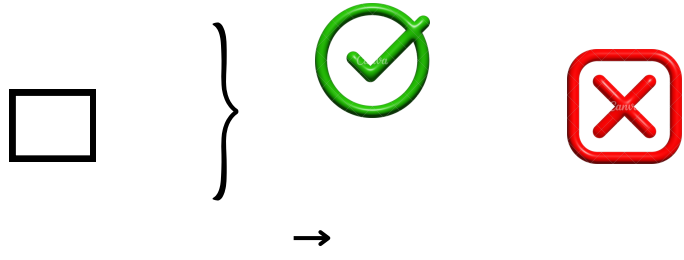
```
int main()
{
    char *source[100],*target[100];
    scanf(" %[^\n]",&source);
    strcpying(target,source);
    printf("%s\n",target);
    return 0;
}
```

Operator Precedence and Associativity:

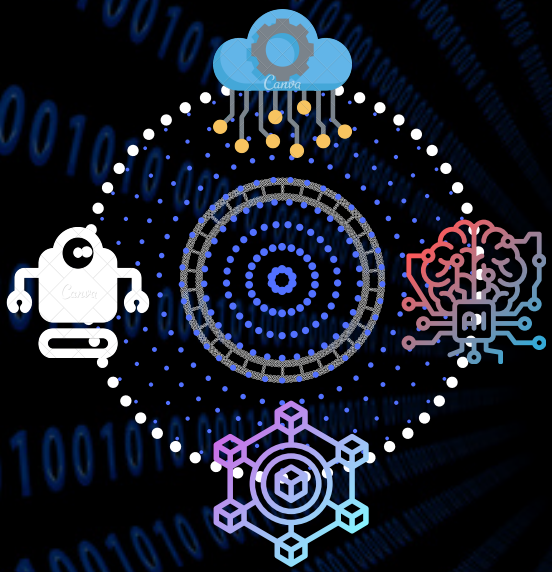


Hamim

Jim



- **What is the output of the following C program fragment:**



C++ LANGUAGE
(FIRST PART)
T.I.M. HA-MEAM

ABC
PROKASHONI