Unit-1

Introduction to Object-Oriented Programming – Programming Paradigms, Data Types, Variables, Constants, Operators,Decision Statements &Control Structures, Arrays, Namespace, Default Arguments, Constant Arguments, Parameter passing techniques, Features of Object-Oriented Programming.

Introduction:

- C++ is an object oriented programming language, C++ was developed by BJarne Stroustrup in 1983 at AT & T Bell laboratories, USA.
- C++ was developed from C and simula 67 language. C++ was early called 'C with classes'.
- C++ is derived from C Language. It is a Superset of C.
- In C++, the major change was the addition of classes and a mechanism for inheriting class objects into other classes.
- Most C Programs can be compiled in C++ compiler.
- C++ expressions are the same as C expressions.
- All C operators are valid in C++.

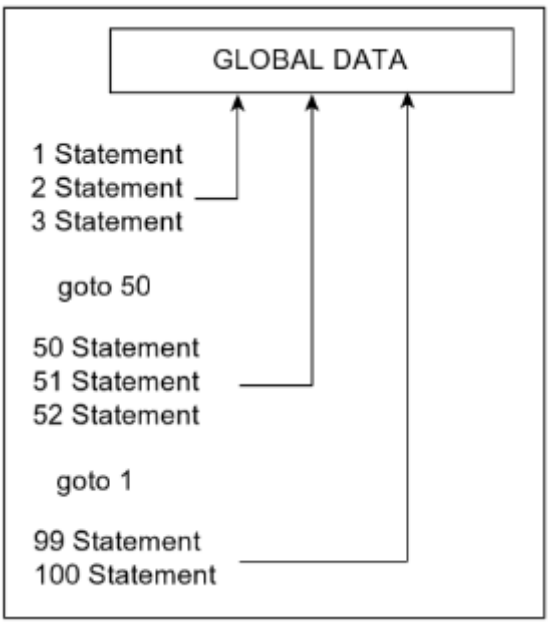
1.Programming Paradigms

The programming paradigm is the way of writing computer programs. There are four programmingparadigms and they are as follows.

- **Monolithic programming paradigm**
- **Structured-oriented programming paradigm**
- **Procedural-oriented programming paradigm**
- **Object-oriented programming paradigm**

Monolithic Programming Paradigm

The Monolithic programming paradigm is the oldest. It has the following characteristics. It is alsoknown as the imperative programming paradigm.



- In this programming paradigm, the whole program is written in a single block.
- We use the **goto** statement to jump from one statement to another statement.
- It uses all data as global data which leads to data insecurity.
- There are no flow control statements like if, switch, for, and while statements in this paradigm.
- There is no concept of data types.

An **example** of a Monolithic programming paradigm is **Assembly**

language.

Structure-oriented Programming Paradigm

The Structure-oriented programming paradigm is the advanced paradigm of the monolithic paradigm.It has the following characteristics.

- This paradigm introduces a modular programming concept where a larger program is dividedinto smaller modules.
- It provides the concept of code reusability.
- It is introduced with the concept of data types.
- It also provides flow control statements that provide more control to the user.
- In this paradigm, all the data is used as global data which leads to data insecurity.

Examples of a structured-oriented programming paradigm is **ALGOL, Pascal, PL/I**

and Ada.

Procedure-oriented Programming Paradigm

The procedure-oriented programming paradigm is the advanced paradigm of a structure-orientedparadigm. It has the following characteristics.

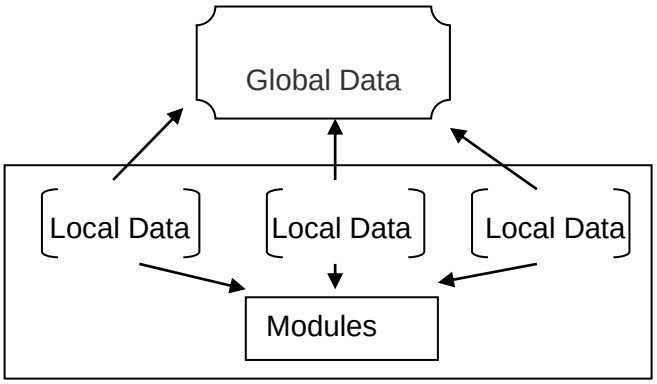


Fig : Program in Procedural Programming

- This paradigm introduces a modular programming concept where a larger program is dividedinto smaller modules.
- It provides the concept of code reusability.
- It is introduced with the concept of data types.
- It also provides flow control statements that provide more control to the user.
- It follows all the concepts of structure-oriented programming paradigm but the data is definedas global data, and also local data to the individual modules.
- In this paradigm, functions may transform data from one form to another.

Examples of procedure-oriented programming paradigm is **C, visual basic, FORTRAN**, etc.

Object-oriented Programming Paradigm

The object-oriented programming paradigm is the most popular. It has the following characteristics.

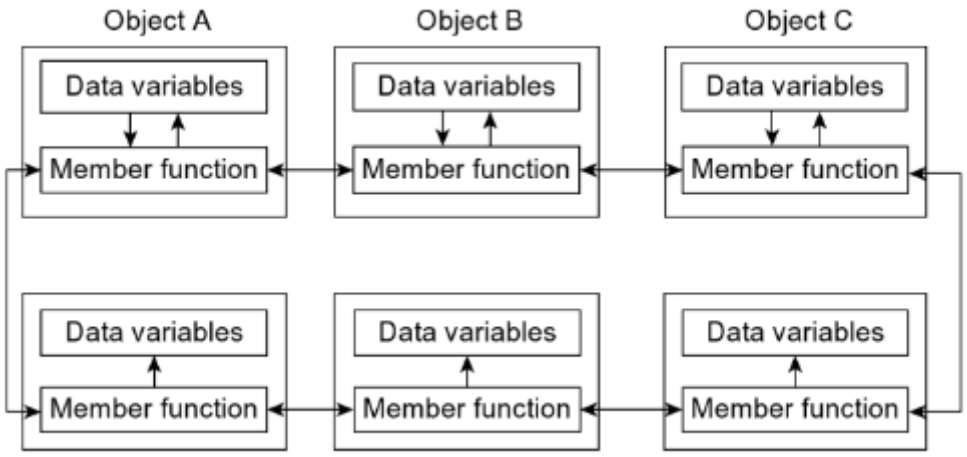


Fig : Relation between data and member function in OOP

- In this paradigm, the whole program is created on the concept of objects.
- In this paradigm, objects may communicate with each other through function.
- This paradigm mainly focuses on data rather than functionality.
- In this paradigm, programs are divided into what are known as objects.
- It follows the bottom-up flow of execution.
- It introduces concepts like data abstraction, inheritance, and overloading of functions andoperators overloading.
- In this paradigm, data is hidden and cannot be accessed by an external function.
- It has the concept of friend functions and virtual functions.
- In this paradigm, everything belongs to objects.

Examples of object-oriented programming paradigm is **C++, Java, C#, Python**, etc.

OOP Vs Procedural Paradigm

Both object-oriented and procedure-oriented paradigms are very popular and most commonly usedprogramming paradigms. The following table gives the differences between those.

Procedure-oriented	**Object-oriented**
It is often known as POP (procedure-oriented programming).	It is often known as OOP (object-oriented programming).
It follows the top-bottom flow of execution.	It follows the bottom-top flowof execution.
Larger programs have divided into smaller modules called as functions.	The larger program hasdivided into objects.
The main focus is on solving the problem.	The main focus is on datasecurity.
It doesn’t support data abstraction.	It supports data abstractionusing access specifiers thatare public, protected, and private.

It doesn't support inheritance.	It supports the inheritance offour types.
Overloading is not supported.	It supports the overloading offunction and also the operator.
There is no concept of friend function and virtual functions.	It has the concept of friend function and virtual functions.
Examples - C, FORTRAN	Examples - C++ , Java , VB.net, C#.net, Python, RProgramming, etc.

2.C++ Data Types

The data type is a category of data. In other words, a data type is a collection of data values with similar characteristics. In the C++ programming language, every variable needs to be created with a datatype.

The data type of a variable specifies the following.

- The type of value can be stored in a variable.
- The amount of memory in bytes has to be allocated for a variable.
- The range of values has to be stored in a variable.

The C++ programming language supports the following data types.

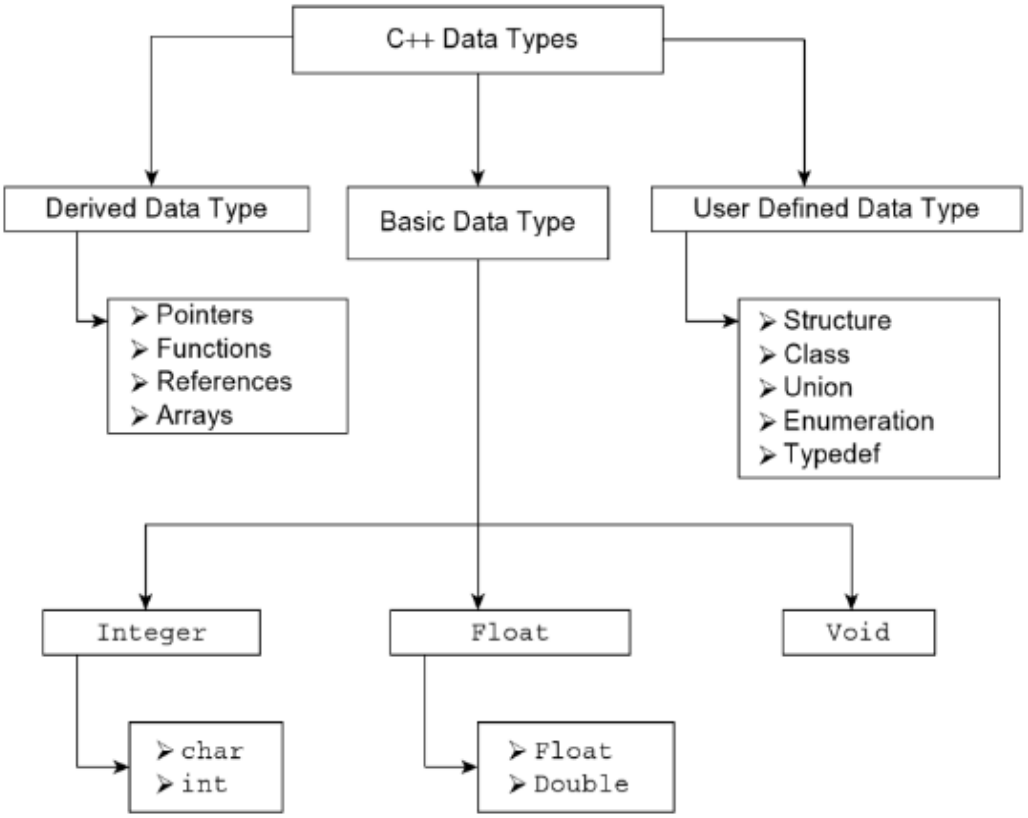


Fig : C++ Data Types

Let's look at each data type in detail.

Integer Data type

An integer is a number without a decimal part. For example, the value 10 is an integer. In c++ integer data type has the following characteristics.

- Integer data type is represented using the keyword "**int**".
- The integer data type allows storing any value in the range from -32768 to 32767. However, itmay change using type modifier.

Floating-point Data type

A floating-point value is a number with a decimal part. For example, the value 10.65 is a floating-point.In c++ floating-point data type has the following characteristics.

- Floating-point data type is represented using the keyword "**float**".
- The floating-point data type allows storing any value in the range from 2^{-31} to 2^{31}.
- The floating-point data type does not allows type modifier.
- The floating-point data type allows upto 6 decimal points.

Double Data type

The double data type is similar to the floating-point data type but it allows a wide range of values. The double data type has the following characteristics.

- The double data type is represented using the keyword "**double**".
- The double data type allows storing any value in the range from 2^{-63} to 2^{63}.
- The double data type does not allows type modifier except long.
- The double data type allows double precision upto 12 decimal points.

Character Data type

A character is a symbol enclosed in a single quotation. Here, the symbol may be anything like an alphabet, a digit, or a special symbol. In C++, the character data type has the following characteristics.

- The character data type is represented using the keyword "**char**".
- The character data type allows storing any value in the range from -128 to 127.
- The character data type allows only signed and unsigned type modifiers.

Boolean Data type

The boolean data type contains only two values 0 and 1. Here the value 0 represents false, and 1 represents true. The boolean data type has the following characteristics.

- The boolean data type is represented using the keyword "**bool**".
- The boolean data type allows storing only 0 and 1.
- The boolean data type does not allows type modifier.

Void Data type

The void data type is a value less data type, and it represents nothing. The void data type has the following characteristics.

- The void data type is represented using the keyword "**void**".
- The void value does not have any type specifier.
- The void data type does not allows storing any value.
- The void data type does not allows type modifier.

Wide character Data type

The wide-character data type is similar to character data type, but it takes 2 bytes of memory. It allows a wide range of character values compare to the character data type. This data type has supported by the latest compilers only. The keyword **wchar_t** is used to represent wide character data type.

The following table describes the characteristics of each data type briefly.

Example :

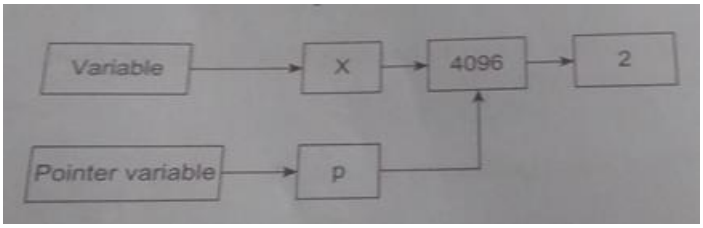
```
#include<iostream>
using namespace std;
int main() {
  wchar_t x =L'@';
  cout << "x is: " << x << endl;
 }
```

Pointer:

A pointer is a memory variable that stores a memory address.

Ex:

int x=2,*p;



Functions:

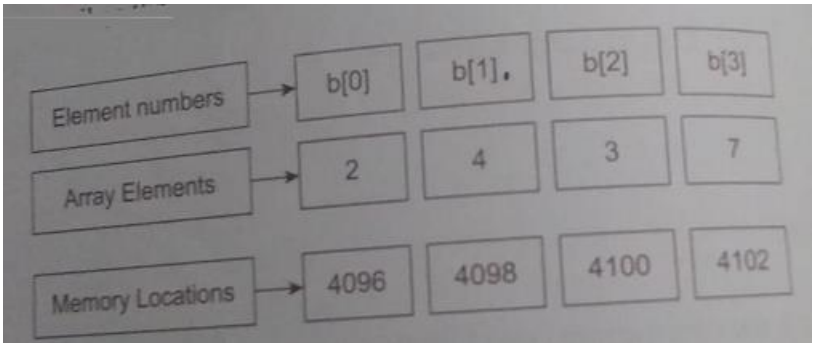
A function is a self-contained block or a subprogram of one or more statements that perform a specific task when

called.

Arrays:

Array is a collection of elements of similar data type in which each element is located in separate memory location.

Int a[4];



References:

The referencing operator is used to define referencing variable.A reference variable prepares an alternative name

for previously defined variable.

Syntax:

Data_type & reference_variable=another_variable;

Ex:

Int x=10;

Int &y=x;

Enumerated:

The enum is a keyword.It is used for declaring enumeration data types.The programmer can declare new data type

and define the variables of these data type that can hold.

Ex:

Enum logical {false,true};

Enum component {solid,liquid,gas};

3.C++ Variables

A variable is named memory location, where the user can store different values of the specified data type. In simple words, a variable is a value holder. Every variable in the program has the following properties.

- Every variable has a name (user-specified) and an address.
- Every variable must be created with a data type.
- Every variable is allocated with a memory based on the data type of it.
- Every variable has a value.

Types of Variables

Based on the location of variable declaration, variables are classified into five types. They are as follows.

- Local Variables
- Global Variables
- Formal Variables
- Member Variables
- Instance Variables

Local Variables

The variables that are declared inside a function or a block are called local variables. The local variable is visible only inside the function or block in which it is declared. That means the local variables can be accessed by the statements that are inside the function or block in which the variable has declared. Outside the function or block, the local variable can't be accessible.

The local variables are created upon execution control enters into the function or block and destroyed upon exit.

Global Variables

The variables that are declared outside a function are called global variables. The global variable is visible inside all the functions that are defined after its declaration. That means the global variables can be accessed by all the functions that are created after the global variable declaration. The functions that have created before the global variable declaration can't access it.

The local variables are created upon execution starts and destroyed upon exit.

Formal Variables

The variables that are created in the function definition as receivers to the parameter values are called formal variables. The formal variables are also known as formal parameters, and they act as local variables inside the function.

Member Variables

The variables that are created in a class are known as member variables. The member variables are accessible to all the methods of that class. The member variables are also accessible to the methods of other classes using inheritance mechanism.

Instance Variables

The instance variable is a special type of variable of a user-defined data type called class. That means an instance variable is a variable of class type. The instance variables are also known as objects. The instance variables are used to access the class members from outside the class.

C++ Expressions

An expression is collection of operators and operands which produce a unique value as result. The expressions are used to perform mathematical and logical operations in a program. In an expression operator is a symbol with pre-defined task and operands are the data values on which the operation isperformed.

4.C++ Operators

The C++ programming language provides a wide range of operators to perform mathematical, logical, and other operations. An operator is a symbol used to perform mathematical and logical operations. Every operator has a pre-defined functionality. However, C++ language provides a concept operator overloading to assign user-defined functionality to most of the operators.

In C++ language, the operators are classified as follows.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Increment and Decrement Operators
- Assignment Operators
- Bitwise Operators
- Conditional Operators
- Scope Resolution Operators
- Other Operators

Arithmetic Operators (+, -, *, /, %)

The arithmetic operators are the symbols that are used to perform basic mathematical operations like addition, subtraction, multiplication, division and percentage modulo. The following table provides information about arithmetic operators.

Operator	Meaning	Example
+	Addition	10 + 5 = 15
-	Subtraction	10 - 5 = 5

*	Multiplication	10 * 5 = 50
/	Division	10 / 5 = 2
%	Remainder of the Division	5 % 2 = 1

Relational Operators (<, >, <=, >=, ==, !=)

The relational operators are the symbols that are used to compare two values. That means the relational operators are used to check the relationship between two values. Every relational operator has two results TRUE or FALSE. In simple words, the relational operators are used to define conditions in a program. The following table provides information about relational operators.

Operator	Meaning	Example
<	Returns TRUE if the first value is smaller than second value otherwisereturns FALSE	10 < 5 is FALSE
>	Returns TRUE if the first value is larger than second value otherwisereturns FALSE	10 > 5 is TRUE
<=	Returns TRUE if the first value is smaller than or equal to second valueotherwise returns FALSE	10 <= 5 is FALSE
>=	Returns TRUE if the first value is larger than or equal to second valueotherwise returns FALSE	10 >= 5 isTRUE

Operator	Meaning	Example
==	Returns TRUE if both values are equal otherwise returns FALSE	10 == 5 is FALSE
!=	Returns TRUE if both values are not equal otherwise returns FALSE	10 != 5 isTRUE

Logical Operators (&&, ||, !)

The logical operators are the symbols that are used to combine multiple conditions into one condition. The following table provides information about logical operators.

Operator	Meaning	Example
&&	**Logical AND** - Returns TRUE if all conditions are TRUEotherwise returns FALSE	10 < 5 && 12 > 10 isFALSE

||	**Logical OR** - Returns FALSE if all conditions are FALSEotherwise returns TRUE	10 < 5 || 12 > 10 isTRUE
!	**Logical NOT** - Returns TRUE if condition is FLASE and returnsFALSE if it is TRUE	!(10 < 5 && 12 > 10) isTRUE

Increment & Decrement Operators (++ & --)

The increment and decrement operators are called unary operators because both need only one operand. The increment operators adds one to the existing value of the operand and the decrement operator subtracts one from the existing value of the operand. The following table provides information about increment and decrement operators.

Operator	Meaning	Example
++	**Increment** - Adds one to existing value	int a = 5; a++; ⇒ a = 6
--	**Decrement** - Subtracts one from existing value	int a = 5; a--; ⇒ a = 4

The increment and decrement operators are used infront of the operand (++a) or after the operand (a++). If it is used infront of the operand, we call it as **pre-increment** or **pre-decrement** and if it is used after the operand, we call it as **post-increment** or **post-decrement**.

Pre-Increment or Pre-Decrement

In the case of pre-increment, the value of the variable is increased by one before the expression evaluation. In the case of pre-decrement, the value of the variable is decreased by one before the expression evaluation. That means, when we use pre-increment or pre-decrement, first the value of the variable is incremented or decremented by one, then the modified value is used in the expression evaluation.

Post-Increment or Post-Decrement

In the case of post-increment, the value of the variable is increased by one after the expression evaluation. In the case of post-decrement, the value of the variable is decreased by one after the expression evaluation. That means, when we use post-increment or post-decrement, first the expression is evaluated with existing value, then the value of the variable is incremented or decremented by one.

Assignment Operators (=, +=, -=, *=, /=, %=)

The assignment operators are used to assign right-hand side value (Rvalue) to the left-hand side variable (Lvalue). The assignment operator is used in different variants along with arithmetic operators. The following table describes all the assignment operators in the C programming language.

Operator	Meaning	Example
=	Assign the right-hand side value to left-hand side variable	A = 15

+=	Add both left and right-hand side values and store the result into left-hand sidevariable	A += 10 ⇒ A = A+10
-=	Subtract right-hand side value from left-hand side variable value and store theresult into left-hand side variable	A -= B ⇒ A = A-B
*=	Multiply right-hand side value with left-hand side variable value and store theresult into left-hand side variable	A *= B ⇒ A = A*B
/=	Divide left-hand side variable value with right-hand side variable value andstore the result into the left-hand side variable	A /= B ⇒ A = A/B
%=	Divide left-hand side variable value with right-hand side variable value andstore the remainder into the left-hand side variable	A %= B ⇒ A = A%B

Bitwise Operators (&, |, ^, ~, >>, <<)

The bitwise operators are used to perform bit-level operations in the c programming language. When we use the bitwise operators, the operations are performed based on the binary values. The following table describes all the bitwise operators in the C programming language.Let us consider two variables A and B as A = 25 (11001) and B = 20 (10100).

Operator	Meaning	Example
&	the result of Bitwise AND is 1 if all the bits are 1 otherwise it is 0	A & B ⇒ 16 (10000)

Operator	Meaning	Example
|	the result of Bitwise OR is 0 if all the bits are 0 otherwise it is 1	A | B ⇒ 29 (11101)
^	the result of Bitwise XOR is 0 if all the bits are same otherwise it is 1	A ^ B ⇒ 13 (01101)
~	the result of Bitwise once complement is negation of the bit (Flipping)	~A ⇒ 6 (00110)

<<	the Bitwise left shift operator shifts all the bits to the left by the specifiednumber of positions	A << 2 ⇒ 100 (1100100)
>>	the Bitwise right shift operator shifts all the bits to the right by the specifiednumber of positions	A >> 2 ⇒ 6 (00110)

Conditional Operator (?:)

The conditional operator is also called a **ternary operator** because it requires three operands. This operator is used for decision making. In this operator, first we verify a condition, then we perform one operation out of the two operations based on the condition result. If the condition is TRUE the first option is performed, if the condition is FALSE the second option is performed. The conditional operator is used with the following syntax.

Condition ? TRUE Part : FALSE Part;

Example

```
A = (10<15)? 100 : 200; // ⇒ A value is 100
```

Scope Resolution Operator (: :)

The scope resolution operator in C++ is used to access the global variable when both local and globalvariables are having the same name, to refer to the static members of a class, and to define a function definition outside the class. We discuss in detail about scope resolution operator in the later tutorial in this series.

Special Operators (sizeof, pointer, comma, dot, etc.)

The following are the special operators in c programming language.

sizeof operator

This operator is used to find the size of the memory (in bytes) allocated for a variable. This operator isused with the following syntax.sizeof(variableName);

Example

```
sizeof(A); // ⇒ the result is 2 if A is an integer
```

Pointer operator (*)

This operator is used to define pointer variables in c programming language.

Comma operator (,)

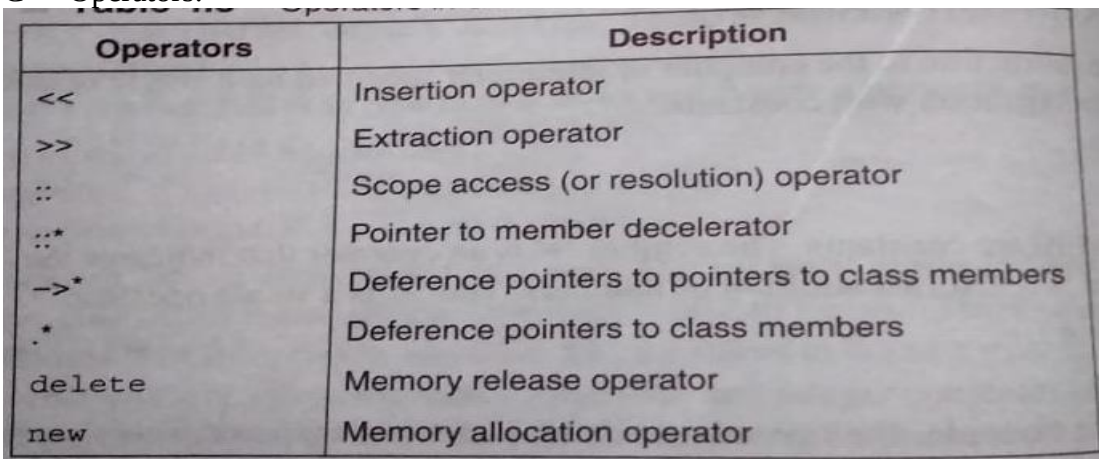
This operator is used to separate variables while they are declaring, separate the expressions in function calls, etc.

Dot operator (.)

This operator is used to access members of a class, and structure.

The operators are also classified based on the number of operands required by the operator.

C++ Operators:



Operators	Description
<<	Insertion operator
>>	Extraction operator
::	Scope access (or resolution) operator
::*	Pointer to member decelerator
->*	Deference pointers to pointers to class members
.*	Deference pointers to class members
delete	Memory release operator
new	Memory allocation operator

Operator Precedance

In any programming language, every operator has provided a preference that is used at the time of expression evaluation. In C++, the following list provides the operators' preference from higher to lower.

1. Pre-increment (or) pre- decrement
2. Peranthasis , shifting operators , sizeof
3. Astrick (*) , multiplication and division
4. Addition , substraction
5. Relational operators
6. Assignment operators
7. Post increment/post decrement

C++ Keywords

A keyword is a reserved word. You cannot use it as a variable name, constant name etc. **A list of 32 Keywords in C++ Language which are also available in C language are given below.**

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

A list of 30 Keywords in C++ Language which are not available in C language are given below.

asm	dynamic_cast	Namespace	reinterpret_cast	bool

explicit	new	static_cast	false	catch
operator	template	Friend	private	class
this	inline	Public	throw	const_cast
delete	mutable	Protected	true	try
typeid	typename	Using	virtual	wchar_t

C++ Identifiers

C++ identifiers in a program are used to refer to the name of the variables, functions, arrays, or other user-defined data types created by the programmer. They are the basic requirement of any language. Every language has its own rules for naming the identifiers.

In short, we can say that the C++ identifiers represent the essential elements in a program which aregiven below:

- **Constants**
- **Variables**
- **Functions**
- **Labels**
- **Defined data types**

Some naming rules are common in both C and C++. They are as follows:

- Only alphabetic characters, digits, and underscores are allowed.
- The identifier name cannot start with a digit, i.e., the first letter should be alphabetical. Afterthe first letter, we can use letters, digits, or underscores.
- In C++, uppercase and lowercase letters are distinct. Therefore, we can say that C++ identifiers are case-sensitive.
- A declared keyword cannot be used as a variable name.

5. Decision Statements C++ if-else

In C++ programming, if statement is used to test the condition. There are various types of ifstatements in C++.

- if statement
- if-else statement
- nested if statement
- if-else-if ladder

C++ IF Statement

The C++ if statement tests the condition. It is executed if condition is true.

```
if(condition){
//code to be executed
}
```

C++ If Example

```
1. #include <iostream>
2. using namespace std;
3. int main () {
4.int num = 10;
6.      if (num % 2 == 0)
7.      {
8.         cout<<"It is even number";9.   }
10.         return 0;
11. }
```

Outputt:/p>

```
It is even number
```

C++ IF-else Statement

The C++ if-else statement also tests the condition. It executes if block if condition is true otherwiseelse block is executed.

```
1.  if(condition){
2.  //code if condition is true
3.  }else{
4. //code if condition is false5.
   }
```

C++ If-else Example

```
1.  #include <iostream>
2.  using namespace std;
3.  int main () {
4.int num = 11;
5.      if (num % 2 == 0)
6.      {
7.         cout<<"It is even number";8.   }
9.      else
10.     {
11.        cout<<"It is odd number";
12.     }
13.         return 0;
14. }
```

Output:

```
It is odd number
```

C++ If-else Example: with input from user

```
1.  #include <iostream>
2.  using namespace std;
3.  int main () {
4. int num;
5. cout<<"Enter a Number: ";
6. cin>>num;
7.      if (num % 2 == 0)
8.      {
9.         cout<<"It is even number"<<endl;
10.     }
11.     else
12.     {
13.        cout<<"It is odd number"<<endl;
14.     }
15.         return 0;
16. }
```

Output:

```
Enter a number:11
It is odd number
```

C++ IF-else-if ladder Statement

The C++ if-else-if ladder statement executes one condition from multiple statements.

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2
is true5.  }
  else if(condition3){
//code to be executed if condition3
is true8.  }
...
else{
//code to be executed if all the conditions are false
}
```

C++ If else-if Example

```
#include <iostream>
using namespace std;
int main () {
 int num;
 cout<<"Enter a number to check grade:";
 cin>>num;
    if (num <0 || num >100)
    {
       cout<<"wrong number";
    }
    else if(num >= 0 && num < 50){
       cout<<"Fail";
    }
    else if (num >= 50 && num < 60)
    {
       cout<<"D Grade";
    }
    else if (num >= 60 && num < 70)
    {
       cout<<"C Grade";
    }
    else if (num >= 70 && num < 80)
    {
       cout<<"B Grade";
    }
    else if (num >= 80 && num < 90)
    {
       cout<<"A Grade";
    }
    else if (num >= 90 && num <= 100)
    {
       cout<<"A+ Grade";
    }
      }
```

Output:

```
Enter a number to check grade:66
C Grade
```

The C++ switch statement executes one statement from multiple conditions. It is like if-else-if ladderstatement in C++.

```
1.  switch(expression){
2.  case value1:
3.   //code to be executed;
4.   break;
5.  case value2:
6.   //code to be executed;
7.   break;
8. ......
9.
10. default:
11.  //code to be executed if all cases are not matched;
12.  break;
13. }
```

C++ Switch Example

```
1.  #include <iostream>
2.  using namespace std;
3.  int main () {
4.   int num;
5.   cout<<"Enter a number to check grade:";
6.   cin>>num;
7.      switch (num)
8.     {
9.        case 10: cout<<"It is 10"; break;
10.       case 20: cout<<"It is 20"; break;
11.       case 30: cout<<"It is 30"; break;
12.       default: cout<<"Not 10, 20 or 30"; break;
13.    }
14.         }
```

Output:

```
Enter a number:
10
It is 10
```

6.C++ For Loop

The C++ for loop is used to iterate a part of the program several times. If the number of iteration isfixed, it is recommended to use for loop than while or do-while loops.

The C++ for loop is same as C/C#. We can initialize variable, check condition and increment/decrement value.

```
for(initialization; condition; incr/decr){
//code to be executed
```

```
}
```

C++ For Loop

Example

```
#include <iostream> using namespace std; int main() {
    for(int i=1;i<=10;i++){
      cout<<i <<"\n";
     }
  }
```

C++ Nested For Loop

In C++, we can use for loop inside another for loop, it is known as nested for loop. The inner loop is executed fully when outer loop is executed one time. So if outer loop and inner loop are executed 4 times, inner loop will be executed 4 times for each outer loop i.e. total 16 times.

C++ Nested For Loop Example

Let's see a simple example of nested for loop in

C++.

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

int main () {
    for(int i=1;i<=3;i++){
      for(int j=1;j<=3;j++){
      cout<<i<<"
      "<<j<<"\n";
     }
     }
  }
```

C++ Infinite For Loop

If we use double semicolon in for loop, it will be executed infinite times. Let's see a simple example ofinfinite for loop in C++.

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

int main () {
    for (; ;)
      {
          cout<<"Infinitive For Loop";
      }
  }
```

C++ While loop

In C++, while loop is used to iterate a part of the program several times. If the number of

iteration isnot fixed, it is recommended to use while loop than for loop.

```
while(condition){
//code to be executed
}
```

C++ While Loop Example

Let's see a simple example of while loop to print table

of 1.

```
#include <iostream>
using namespace std;
int main() {
 int i=1;
       while(i<=10)
      {
         cout<<i
         <<"\n";i++;
       }
   }
```

C++ Do-While Loop

The C++ do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

The C++ do-while loop is executed at least once because condition is checked after loop body.

```
do{
//code to be executed
}while(condition);
```

C++ do-while Loop Example

Let's see a simple example of C++ do-while loop to print the

table of 1.

```
#include <iostream>
using namespace std;
int main() {
    int i = 1;
        do{
           cout<<i<<"\n
           ";i++;
        } while (i <= 10) ;
}
```

C++ Break Statement

The C++ break is used to break loop or switch statement. It breaks the current flow of the program atthe given condition. In case of inner loop, it breaks only inner loop.

```
jump-statement;
break;
```

C++ Comments

The C++ comments are statements that are not executed by the compiler. The comments in C++ programming can be used to provide explanation of the code, variable, method or class. By the helpof comments, you can hide the program code also.

There are two types of comments in C++.

- Single Line comment
- Multi Line comment

C++ Single Line Comment

The single line comment starts with // (double slash). Let's see an example of single line comment inC++.

```
#include <iostream>
using namespace std;
int
main(){
int x = 11; // x is a
variablecout<<x<<"\n";
}
```

C++ Multi Line Comment

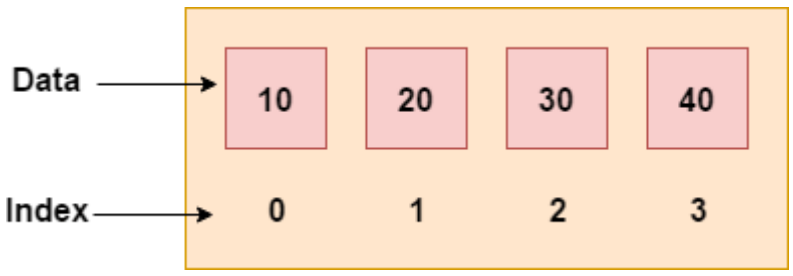
The C++ multi line comment is used to comment multiple lines of code. It is surrounded by slash andasterisk (/* */). Let's see an example of multi line comment in C++.

```
#include <ostream>
using namespace std;
int
main(){
/* declare and
print variable in C++. */
int x = 35;
cout<<x<<"\n";
}
```

7.C++ Arrays

Like other programming languages, array in C++ is a group of similar types of elements that havecontiguous memory location.

In C++ **std::array** is a container that encapsulates fixed size arrays. In C++, array index starts from 0.We can store only fixed set of elements in C++ array.



Advantages of C++ Array

- Code Optimization (less code)

- Random Access
- Easy to traverse data
- Easy to manipulate data
- Easy to sort data etc.

Disadvantages of C++ Array

- Fixed size

C++ Array Types

There are 2 types of arrays in C++ programming:

1. Single Dimensional Array
2. Multidimensional Array

C++ Single Dimensional Array

Let's see a simple example of C++ array, where we are going to create, initialize and traverse array.

```
1.  #include <iostream>
2.  using namespace std;
3. int
main()4.
   {
5.   int arr[5]={10, 0, 20, 0, 30};  //creating and initializing array
6.   //traversing array
7.   for (int i = 0; i < 5; i++)
8.   {
9.      cout<<arr[i]<<"\n";
10.  }
11.  }
```

C++ Array Example: Traversal using foreach loop

We can also traverse the array elements using foreach loop. It returns array element one by one.

```
1.  #include <iostream>
2.  using namespace std;
3. int
main()4.
   {
5.   int arr[5]={10, 0, 20, 0, 30}; //creating and initializing array
6.   //traversing array
7.   for (int i: arr)
8.   {
9.      cout<<i<<"\n";
10.  }
11.  }
```

C++ Multidimensional Arrays

The multidimensional array is also known as rectangular arrays in C++. It can be two dimensional orthree dimensional. The data is stored in tabular form (row * column) which is also known as matrix.

C++ Multidimensional Array Example

Let's see a simple example of multidimensional array in C++ which declares, initializes and traversetwo dimensional arrays.

```
1.  #include <iostream>
2.  using namespace std;
3. int main()
4. .        {
5.    int test[3][3];  //declaration of 2D array
6. test[0][0]=5;
//initialization7.
   test[0][1]=10;
8. test[1][1]=15;
9. test[1][2]=20;
10.         test[2][0]=30;
11.         test[2][2]=10;
12.         //traversal
13.         for(int i = 0; i < 3; ++i)
14.         {
15.  for(int j = 0; j < 3; ++j)
16.  {
17.     cout<< test[i][j]<<" ";
18.  }
19.  cout<<"\n"; //new line at each row
20.         }
21.         return 0;
22. }
```

Output:

```
5 10 0
0 15 20
30 0 10
```

8.Parameter Passing techniques

Actual Parameters are the parameters that appear in the function call statement.

Formal Parameters are the parameters that appear in the declaration of the function which has beencalled.

Now let's look at how each call mechanism works.

Call by Value

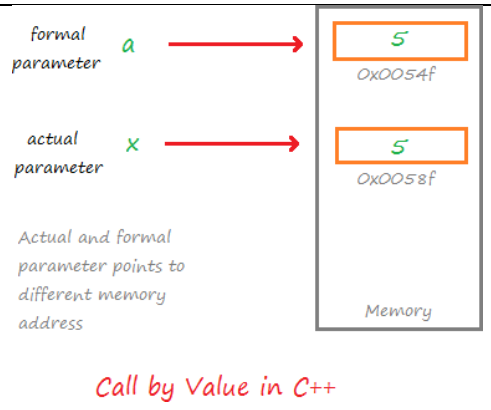
When a function is called in the call by value, the **value of the actual parameters is copied into formal parameters.**
Both the actual and formal parameters have their own copies of values, therefore any change in one ofthe types of parameters will not be reflected by the other.

This is because both actual and formal parameters point to different locations in memory (i.e. they bothhave different memory addresses).

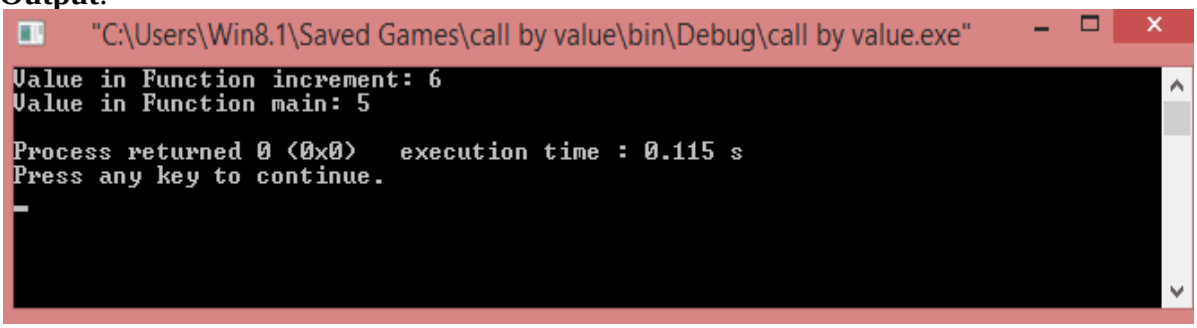
Call by value method is useful when we do not want the values of the actual parameters to be changed by the function that has been invoked.

C++ Example implementing Call by Value

```
#include <iostream>
 using namespace std;
void increment(int a){
a++;
cout << "Value in Function increment: "<< a <<endl;
}
int main()
{
int x = 5; increment(x);
cout << "Value in Function main: "<< x <<endl; return 0;
}
```



Output:



Note the output of the program. The value of 'a' has been increased to 6, but the value of 'x' in the main method remains the same.

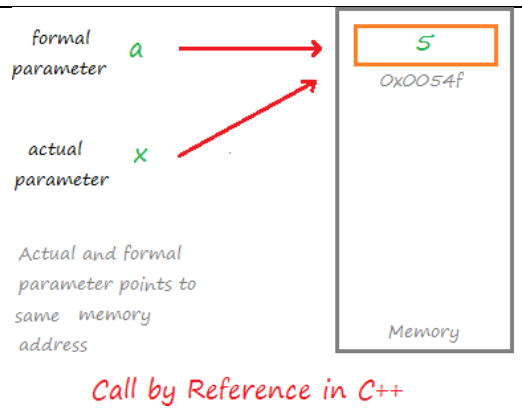
This proves that the value is being copied to a different memory location in the call by value.

Call by Reference

In the call by reference, **both formal and actual parameters share the same value.**Both the actual and formal parameter points to the same address in the memory.

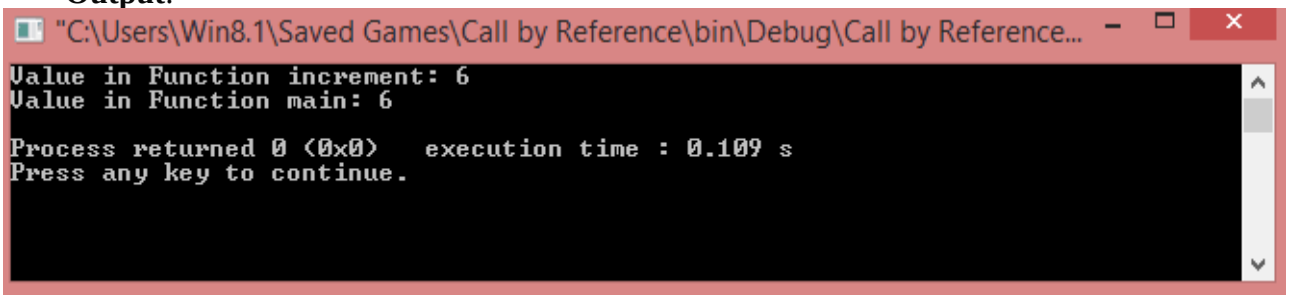
C++ Example implementing Call by Reference

```
#include <iostream>
using namespace std;
void increment(int &a){
a++;
cout << "Value in Function increment: "<< a <<endl;
}
```



```
int main()
{
int x = 5; increment(x);
cout << "Value in
Function main: "<< x
<<endl; return 0;
}
```

Output:



That means any change on one type of parameter will also be reflected by other.

Calls by reference are preferred in cases where we do not want to make copies of objects or variables, but rather we want all operations to be performed on the same copy.

Note: *For creating reference, the '**&**' operator is used in preceding of variable name.*
Note the output in this case. The value of 'a' is increased to 6, the value of 'x' in the main also changes to 6.
This proves that changes made to formal parameters are also reflected by the actual parameters as theyshare the same memory address space.

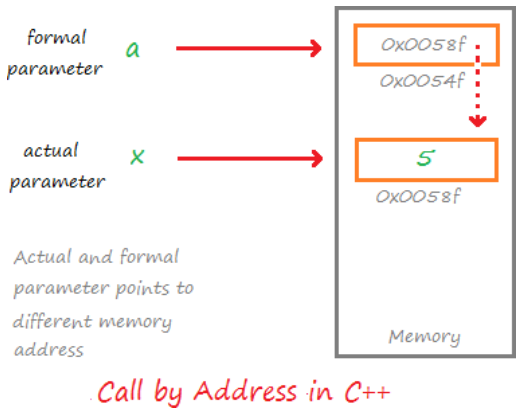
Call by Address

In the call by address method, **both actual and formal parameters indirectly share the same variable.**

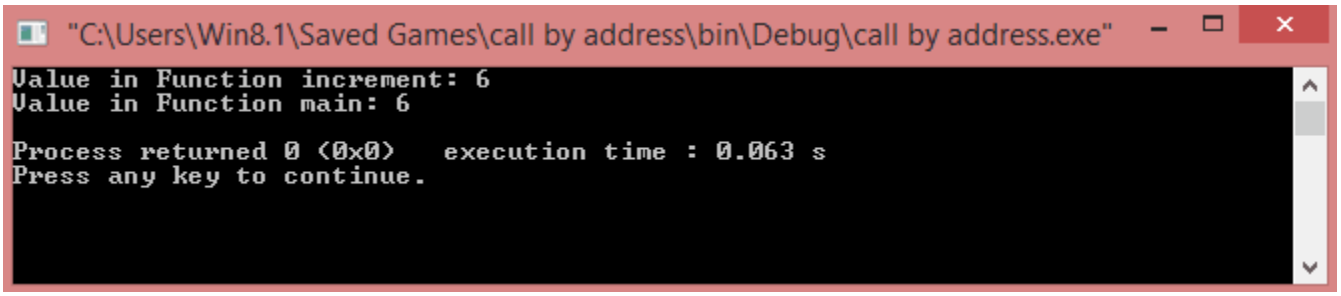
In this type of call mechanism, pointer variables are used as formal parameters.
The formal pointer variable holds the address of the actual parameter, hence the changes done by the formal parameter is also reflected in the actual parameter.

C++ Example implementing Call by Address

```
#include <iostream>
 using namespace std;
 void increment(int *a){
(*a)++;
cout << "Value in Function increment: "<< *a
<<endl;
}
int main()
{
int x = 5;
increment(&x); //Passing address of x
cout << "Value in Function main: "<< x <<endl;
return 0;
}
```



Output:



The output here is the same as in the case of call by reference i.e. the value of both 'a' and 'x' changes.

As demonstrated in the diagram, both parameters point to different locations in memory, but since the formal parameter stores the address of the actual parameter, they share the same value.

9.C++ Namespaces

As the same name can't be given to multiple variables, functions, classes etc. in the same scope.so to overcome this situation name space is introduced.

Namespace in one of the new feature introduced by ANSI C++.It is used by programmer to avoid name classes when programmer uses more than one library.

Defining Name space:
A namespace defining begins with the keyboard namespace followed by the namespace name
Syntax:
namespace namespace_name
{
----------;
----------;
}
Example

Let's see the simple example of namespace which include

variable and functions.

```
#include <iostream>
using namespace std;
namespace first_space{
  void func() {
      int x=20;
   cout<<"First Namespace"<<endl;     }
}
namespace second_space  {
   void func() {
   int x=10;
     cout<<"Hello Second Namespace"<<endl;
   }
}
int main()
```

```
{
 First_space::func
();
 Cout<< First_space::x;
 second_space::func();
 Cout<< second_space::x;
 return 0;
}
```

Using namespace:

```
 Using namespace
first_space;
 Int main(){
 Func();
 Cout<<x;
 }
```

Using namespace std:

Here std is a name space whose members are used in the programs. Those members are cout,cin,endl etc..This namespace present in iostream.h header file

```
 Namespace std
 {
 ostream cout;
 Istream cin;
 …
 ..
 }
```

10.Default arguments in C++

Definition

A default argument is a value in the function declaration automatically assigned by the compiler if thecalling function does not pass any value to that argument.

Characteristics for defining the default arguments

Following are the rules of declaring default arguments -

The values passed in the default arguments are not constant. These values can be overwritten if thevalue is passed to the function. If not, the previously declared value retains.

During the calling of function, the values are copied from left to

right.All the values that will be given default value will be on the

right.

```
#include<iostream>
using namespace std;
int sum(int x, int y, int z=0, int w=0) // Here there are two values in the default arguments
{ // Both z and w are initialised to zero
   return (x + y + z + w); // return sum of all parameter values
}
int main()
{
   cout << sum(10, 15) << endl; // x = 10, y = 15, z = 0, w = 0
   cout << sum(10, 15, 25) << endl; // x = 10, y = 15, z = 25, w =
   0 cout << sum(10, 15, 25, 30) << endl; // x = 10, y = 15, z =
   25, w = 30
   return 0;
}
```

11.Constant argument:

A constant argument is the one whose modification cannot take place by the function. Furthermore, in order to make an argument constant to a function, the use of a keyword const can take place like- int sum (const int a, const int b).

```
#include<iostrea
m> using
namespace std;
float area(int r,const int pi=3.14);// function
declarationint main()
{
int r;
cout<<"Enter radius of a
circle:";cin>>r;
cout<<"The area of circle is
:"<<area(r);return 0;
}
float area(int r,const int pi)
{
return pi*r*r;
//pi++ cannot be done as it is constant;
}
```

12.Object-Oriented Programming In C++

Object-oriented programming revolves around data. The main programming unit of OOP is the object.An object is a representation of a real-time entity and consists of data and methods or functions that operate on data. This way, data, and functions are closely bound and data security is ensured.

In OOP, everything is represented as an object and when programs are executed, the objects interact with each other by passing messages. An object need not know the implementation details of anotherobject for communicating.

Apart from objects, OOP supports various features which are listed below:

- **Classes**
- **Encapsulation**
- **Abstraction**
- **Inheritance**
- **Polymorphism**

Using OOP, we write programs using classes and objects by utilizing the above features. A programming language is said to be a true object-oriented programming language if everything it represents is using an object. Smalltalk is one language which is a pure object-oriented programminglanguage.

On the other hand, programming languages like C++, and Java are said to be partial object-orientedprogramming languages.

Why C++ Is Partial OOP?

C++ language was designed with the main intention of using object-oriented features to C language.

Although C++ language supports the features of OOP like Classes, objects, inheritance, encapsulation, abstraction, and polymorphism, there are few reasons because of which C++ is classified as a partial object-oriented programming language.

We present a few of these reasons below:

#1) Creation of class/objects is Optional

In C++, the main function is mandatory and is always outside the class. Hence, we can have only onemain function in the program and can do without classes and objects.

This is the first violation of Pure OOP language where everything is represented as an object.

#2) Use of Global Variables

C++ has a concept of global variables that are declared outside the program and can be accessed by any other entity of the program. This violates encapsulation. Though C++ supports encapsulation withrespect to classes and objects, it doesn't take care of it in case of global variables.

#3) Presence of a Friend Function

C++ supports a friend class or function that can be used to access private and protected members of other classes by making them a friend. This is yet another feature of C++ that violates OOP paradigm.

To conclude, although C++ supports all the OOP features mentioned above, it also provides featuresthat can act as a workaround for these features, so that we can do without them. This makes C++ a partial Object-oriented programming language.

OOP Features

Here we will introduce various OOP features that are used for programming.

Classes & Objects

An object is a basic unit in object-oriented programing. An object contains data and methods orfunctions that operate on that data. Objects take up space in memory.

A class, on the other hand, is a blueprint of the object. Conversely, an object can be defined as aninstance of a class. A class contains a skeleton of the object and does not take any space in the memory.

Let us take an **Example** of a car object. A car object named "Maruti" can have data such as

color; make, model, speed limit, etc. and functions like accelerate. We define another object "ford". This canhave similar data and functions like that of the previous object plus some more additions.
Similarly, we can have numerous objects of different names having similar data and functions andsome minor variations.

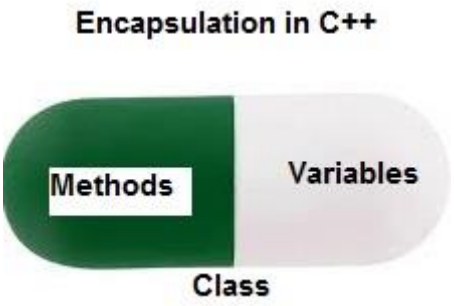
Thus instead of defining these similar data and functions in these different objects, we define a blueprint of these objects which is a class called Car. Each of the objects above will be instances ofthis class car.

Abstraction

Abstraction is the process of hiding irrelevant information from the user. **For Example**, when we are driving the car, first we start the engine by inserting a key. We are not aware of the process that goeson in the background for starting the engine.
Using abstraction in programming, we can hide unnecessary details from the user. By usingabstraction in our application, the end user is not affected even if we change the internal implementation.

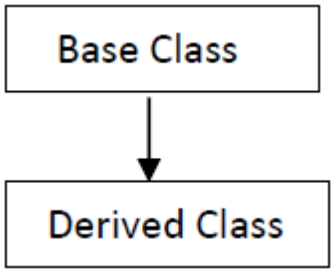
Encapsulation

Encapsulation is the process using which data and the methods or functions operating on them are bundled together. By doing this, data is not easily accessible to the outside world. In OOP we achieveencapsulation by making data members as private and having public functions to access these data members.



Inheritance

Inheritance is the process of acquiring (getting) the properties of some other class. The class whose properties are being inherited is called as base class and the class which is getting the properties is called as derived class.
Sub Class: The class that inherits properties from another class is called Sub class or Derived Class.
Super Class: The class whose properties are inherited by sub class is called Base Class or Super class.



Reusability: Using the already existing code is called as reusability. This is mostly used in inheritance. The already existing code is inherited to the new class. It saves a lot of time and effort. It also reduces the size of the program.

Polymorphism

Polymorphism: Polymorphism means the ability to take many forms. Polymorphism allows to take

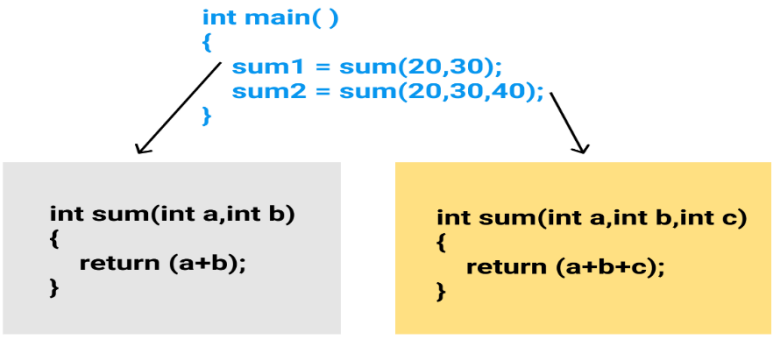
different implementations for same name.

poly -> many

morphism -> forms

There are two types of polymorphism, Compile time polymorphism and run time polymorphism. In Compile time polymorphism binding is done at compile time and in runtime polymorphism binding is done at runtime. e.g.: Function overloading, operator overloading

Function Overloading: Function overloading is a part of polymorphism. Same function name having different implementations with different number and type of arguments.



Operator Overloading: Operator overloading is a part of polymorphism. Same operator can have different implementations with different data types.

Virtual Functions: Virtual functions are special type of functions which are defined in the base class and are redefined in the derived class. When virtual function is called with a base pointer and derived object then the derived class function will be called. A function can be defined as virtual by placing the keyword virtual for the member function.

Dynamic Binding

OOP supports dynamic binding in which function call is resolved at runtime. This means that the codeto be executed as a result of a function call is decided at runtime. Virtual functions are an example of dynamic binding.

Message Passing

Message Passing: An object-oriented program contains a set of objects that communicate with one another.

The process of object oriented programming contains the basic steps:

1. Creating classes
2. Creating objects
3. Communication among objects

This communication is done with the help of functions (i.e., passing objects to functions)

Advantage of OOP

Object oriented technology provides many advantages to the programmer and the user. This technology solves many problems related to software development, provides improved quality and low cost software.

1. Object oriented programs can be comfortably upgraded.
2. Using inheritance, we can eliminate redundant program code and continue the use of previously defined classes.
3. The technology of data hiding facilitates the programmer to design and develop safe programs that do not disturb code in other parts of the program.
4. The encapsulation feature provided by OOP languages allows programmer to define the class with many functions and characteristics and only few functions are exposed to the user.
5. All object oriented programming languages allows creating extended and reusable parts of programs.
6. Object oriented programming changes the way of thinking of a programmer. This results in rapid development of new software in a short time.

7. Objects communicate with each other and pass messages.

Disadvantage of OOP:

1. With OOP, classes tend be overly generalized.
2. The relation among classes become artificial at times.
3. The OOP programs design is tricky.
4. Also one needs to do proper planning and proper design for OOP Programming.
5. To Program with OOP, Programmer need proper skills such as design skills , programming skills, thinking in term of objects etc.

Applications of OOPS

The promising areas for application of OOP includes:

- Real-time systems
- Simulation and modeling
- Object-oriented databases
- Hypertext, hypermedia and expert text
- AI and expert systems
- Neural networks and parallel programming
- Decision support and Automation system
- CIM/CAM/CAD systems