# Object Oriented Programming using C++

CSE-III
CSE206-OOP

Dr. Priyanka Singh

## UNIT I: INTRODUCTION

What is object-oriented programming? Comparison of procedural programming and Object-Oriented Programming - Characteristics of Object-Oriented Languages - C++ Programming Basics: Basic Program Construction - Data Types, Variables, Constants - Type Conversion, Operators, Library Functions - Loops and Decisions, Structures - Functions: Simple Functions, passing arguments, Returning values, Reference Arguments. - Recursion, Inline Functions, Default Arguments - Storage Classes - Arrays, Strings, Addresses, and pointers. Dynamic Memory management. Linked lists in C++.

# Course Plan

| Unit No. | Unit Name | Required Hours |
|---|---|---|
| Unit 1 | INTRODUCTION | 9 |
| 1 | Understanding the Object-Oriented World View, A way of viewing world – Agents and Communities, messages and methods, Responsibilities, Classes, Objects, and Methods. | 1 |
| 2 | OOP principles | 1 |
| 3 | An overview of C++, basic program construction - data types, variables, constants - type conversion, operators. | 1 |
| 4 | Decision making and looping constructs | 1 |
| 5 | Arrays, strings and pointers | 2 |
| 6 | Functions, passing arguments, Returning values, Reference Arguments | 1 |
| 7 | Storage Classes | 1 |
| 8 | Dynamic memory management in C++ | 1 |

## Introduction

- The C programming language: Developed in Bell Labs by Dennis Ritchi.
- It was developed as a system implementation language for nascent Unix operating System.
- It is a high-level language.

## Origins of C++

- C++ developed in 1979 by Bjarne Stroustrup, while working on Simula.
- It is the first language to support Object Oriented Programming.
- His objective was to include OOP features in software development, but Simula cannot be used because of slow speed.
- So, he worked on C language and included OOP features ("initially termed as C with Classes").
- C was one of the most liked and widely used professional programming languages in the world, the invention of C++ was necessitated by one major programming factor: increasing complexity.

## Why Use a Language Like C++?

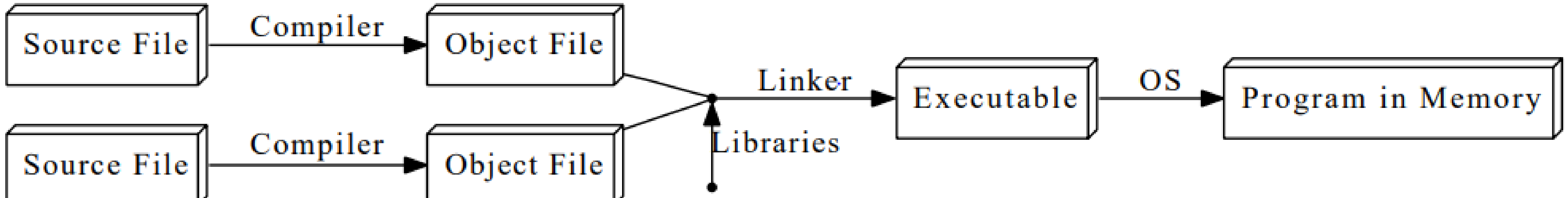
- **Conciseness**: programming languages allow us to express common sequences of commands more concisely. C++ provides some especially powerful short hands.
- **Maintainability**: modifying code is easier when it entails just a few text edits, instead of rearranging hundreds of processor instructions. C++ is object oriented, which further improves maintainability.
- **Portability**: different processors make different instructions available. Programs written as text can be translated into instructions for many different processors; one of C++’s strengths is that it can be used to write programs for nearly any processor.

## Why Use a Language Like C++?

- C++ is a high-level language: when you write a program in it, the shorthands are sufficiently expressive that you don’t need to worry about the details of processor instructions.
- C++ does give access to some lower-level functionality than other languages (e.g. memory addresses).

## The Compilation Process

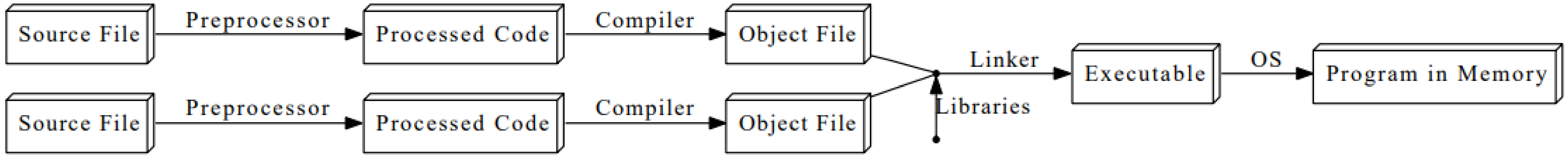
A program goes from text files (or source files) to processor instructions as follows:



- Object files are intermediate files that represent an incomplete copy of the program: each source file only expresses a piece of the program, so when it is compiled into an object file, the object file has some markers indicating which missing pieces it depends on.
- The linker takes those object files and the compiled libraries of predefined code that they rely on, fills in all the gaps, and spits out the final program, which can then be run by the operating system (OS).

# The Compilation Process

- The compiler and linker are just regular programs.
- The step in the compilation process in which the compiler reads the file is called parsing. In C++, all these steps are performed ahead of time, before you start running a program.
- In some languages, they are done during the execution process, which takes time. This is one of the reasons C++ code runs far faster than code in many more recent languages.
- C++ actually adds an extra step to the compilation process: the code is run through a preprocessor, which applies some modifications to the source code, before being fed to the compiler. Thus, the modified diagram is:



## Evolution of C++

- **C++98 (1998):**
  - Templates
  - Standard Template Library (STL) with containers and algorithms
  - Stings
  - I/O streams

- **C++11 (2011):**
  - Multi-threading
  - Smart pointers
  - Hash tables
  - Smart pointers

## Evolution of C++

- **C++14 (2014):**
  - Reader-writer locks
  - Generic lambda functions
- **C++17 (2017):**
  - Fold expressions
  - Structured binding
  - Parallel algorithms
- **C++20 (2020):**
  - Coroutines
  - Modules
  - Ranges Library

# Benefits of C++

- Highly portable language and is a language of choice for multi-device, multi-platform application development.
- Supports OOP features like classes, inheritance, polymorphism, data-abstraction, and encapsulation.
- Has a rich function library (e.g. STL).
- Allows exception handling, function and operator overloading.
- C++ is a very powerful, efficient and fast language that supports wide range of general/GUI applications, 3D graphics for games and real-time mathematical simulations.

Summary: C++ is immensely popular, particularly for applications that require speed and/or access to some low-level features.

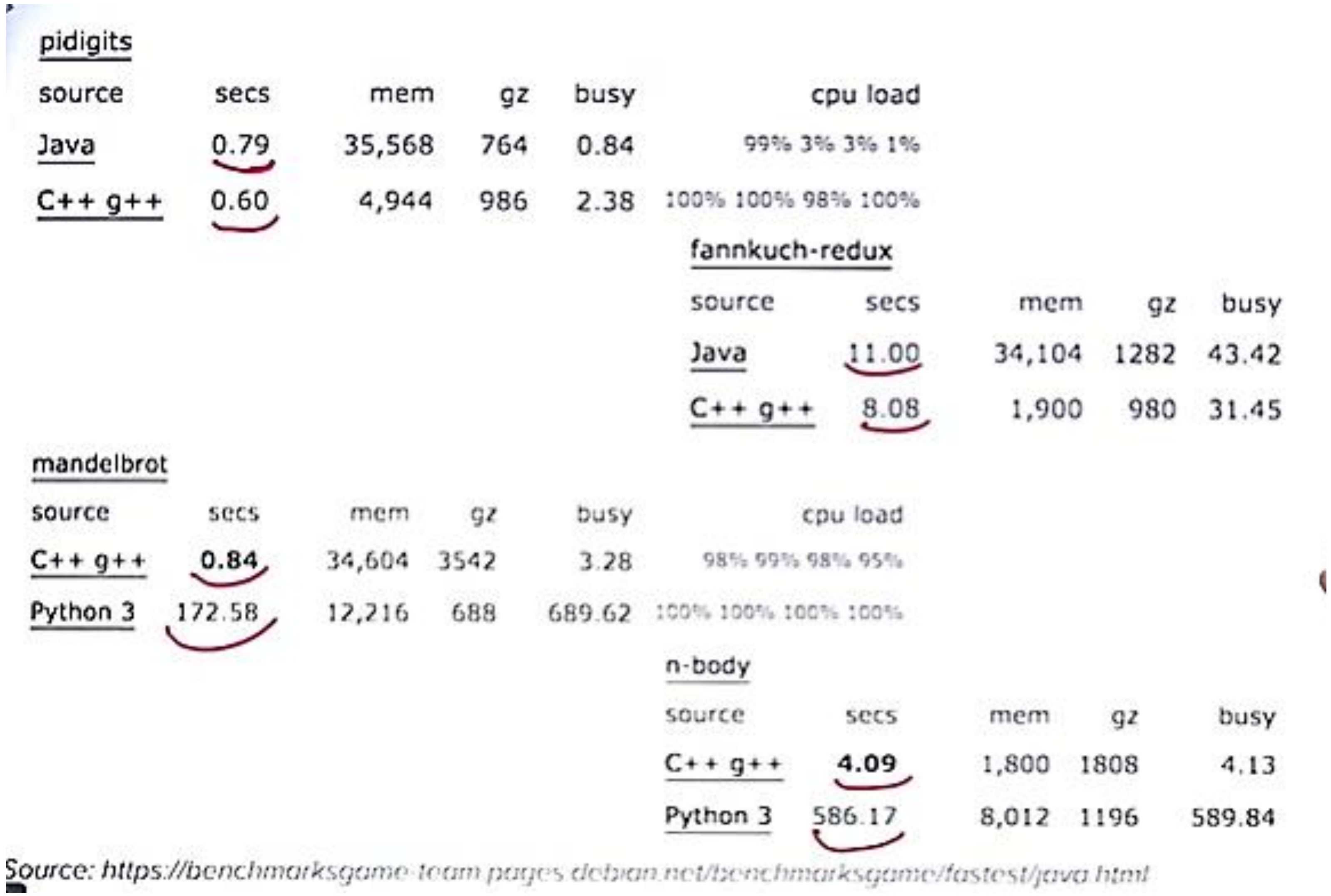
## C++ for Competitive programming

- C++ is a very fast and evolving language. It has many standards such as C++ 11, 14, 17, and 20.
- C++ 20 has many new features to solve problems, e.g. three way comparison operator, co-routines, modules, etc.
- STL: Standard Template Library, a collection of C++ templates to help programmers handle basic data structure and functions such as Lists, stacks, arrays, containers classes, iterators, etc, that are very helpful in competitive programming.
- Ease of learning, fast, efficient, and concise in comparison to Java.

## C++ Scope in industry

- Companies like Microsoft and Adobe mostly use C++ to create desktop applications. Popular C++ coded applications are Adobe reader, office 365, etc.
- In general, C++ is used to create Native apps that are platform dependent, but can execute in a stand alone mode, without internet connectivity.
- Major trading/Fintech. companies like Tower research, Graviton research, that are specialised in Algorithmic trading (requiring very low latency) heavily rely on implementing their trading platforms in C++ .
- Hardware companies such as Samsung, Intel, Qualcomm, etc, use C/C++ for designing embedded systems. Here, they need to work at very low level and work around the memory structure, writing custom device drivers, or writing low level code that directly executes on the bare metal.

# C++ vs Java/Python



pidigits

| source | secs | mem | gz | busy | cpu load |
|---|---|---|---|---|---|
| Java | 0.79 | 35,568 | 764 | 0.84 | 99% 3% 3% 1% |
| C++ g++ | 0.60 | 4,944 | 986 | 2.38 | 100% 100% 98% 100% |

fannkuch-redux

| source | secs | mem | gz | busy |
|---|---|---|---|---|
| Java | 11.00 | 34,104 | 1282 | 43.42 |
| C++ g++ | 8.08 | 1,900 | 980 | 31.45 |

mandelbrot

| source | secs | mem | gz | busy | cpu load |
|---|---|---|---|---|---|
| C++ g++ | 0.84 | 34,604 | 3542 | 3.28 | 98% 99% 98% 95% |
| Python 3 | 172.58 | 12,216 | 688 | 689.62 | 100% 100% 100% 100% |

n-body

| source | secs | mem | gz | busy |
|---|---|---|---|---|
| C++ g++ | 4.09 | 1,800 | 1808 | 4.13 |
| Python 3 | 586.17 | 8,012 | 1196 | 589.84 |

*Source: https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/java.html*

# C++ vs Java

| **Comparison Index** | C++ | Java |
|---|---|---|
| **Platform-independent** | Platform-dependent. | Platform-independent. |
| **Mainly used for** | Mainly used for system programming. | Mainly used for application programming. It is widely used in Windows-based, web-based, enterprise, and mobile applications. |
| **Design Goal** | Designed for systems and applications programming. | Java was designed and created as an interpreter for printing systems but later extended as a support network computing. |
| **Multiple inheritance** | C++ supports multiple inheritance. | Java doesn't support multiple inheritance through class. It can be achieved by using interfaces in java. |
| **Operator Overloading** | C++ supports operator overloading. | Java doesn't support operator overloading. |
| **Pointers** | C++ supports pointers. You can write a pointer program in C++. | Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java. |

# C++ vs Java

| | | |
|---|---|---|
| **Compiler and Interpreter** | Uses **compiler** only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent. | Uses both **compiler** and **interpreter**. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent. |
| **Call by Value and Call by reference** | Supports both call by value and call by reference. | Java supports call by value only. There is no call by reference in java. |
| **Structure and Union** | Supports structures and unions. | Doesn't support structures and unions. |
| **unsigned right shift >>>** | C++ doesn't support >>> operator. | Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. |
| **Inheritance Tree** | C++ always creates a new inheritance tree. | Java always uses a single inheritance tree because all classes are the child of the Object class in Java. The Object class is the root of the inheritance tree in java. |
| **Hardware** | C++ is nearer to hardware. | Java is not so interactive with hardware. |

## C++ vs JAVA/Python

- https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/cpp.html
- https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/gpp-java.html
- https://www.youtube.com/watch?v=UCYqPvYsOz4

## Why is OOP popular ?

- OOP become a popular since it provide a better programming style, you don't need to write code which you really need to run anytime you need (such as in structured programming and assembler).
- You just make a class of object and you may call/instantiate the class and use it from any part of your application.
- It is reusable. Compiler provide a library of those class, and you just use it, no need to write the codes.
- For most of OOP languages like C++, there are increasingly large number of libraries that assist in the development of ubiquitous applications.

## Why is OOP popular ?

- Your application mostly concern on managing the interaction among object, not write a command for any specific jobs.
- OOP is an evolutionary idea and thus scales very well, from very trivial set of problems to complex tasks.
- It provides a form of abstraction that resonates with techniques people use to solve problems in their day-to-day life.

## Thinking Object Oriented

- It defines a way of viewing the world, how we might go about handling a real-world situation and how we could make the computer more closely model the techniques employed.
- Suppose, **Alice** wish to send a flower to **Bob**, who lives in a city miles away. Physically, it is not possible for **Alice** to pick the flower and carry to his home.
- Feasible solution: **Alice** merely goes to the local florist (say **Jay**), tell her the variety and quantity of flowers she wishes to gift **Bob**.
- Here, firstly **Alice** finds an agent (i.e. **Jay**) to solve his problem and pass a request to **Jay** (message) containing his request.

## Thinking Object Oriented

- It is the responsibility of **Jay** to satisfy **Alice’s** request. Here, **Jay** uses some method, some algorithm or set of operations to accomplish that.
- Moreover, Alice needn’t to know the particular method **Jay** used to transfer the flowers. This information is often hidden from **Alice**.
- For instance, **Jay** may in turn use his subordinates to make the flower arrangement, then transfers a message for the delivery person, and so on.
- Observation: Here, the solution to the problem required the help of many individuals.

## Thinking Object Oriented

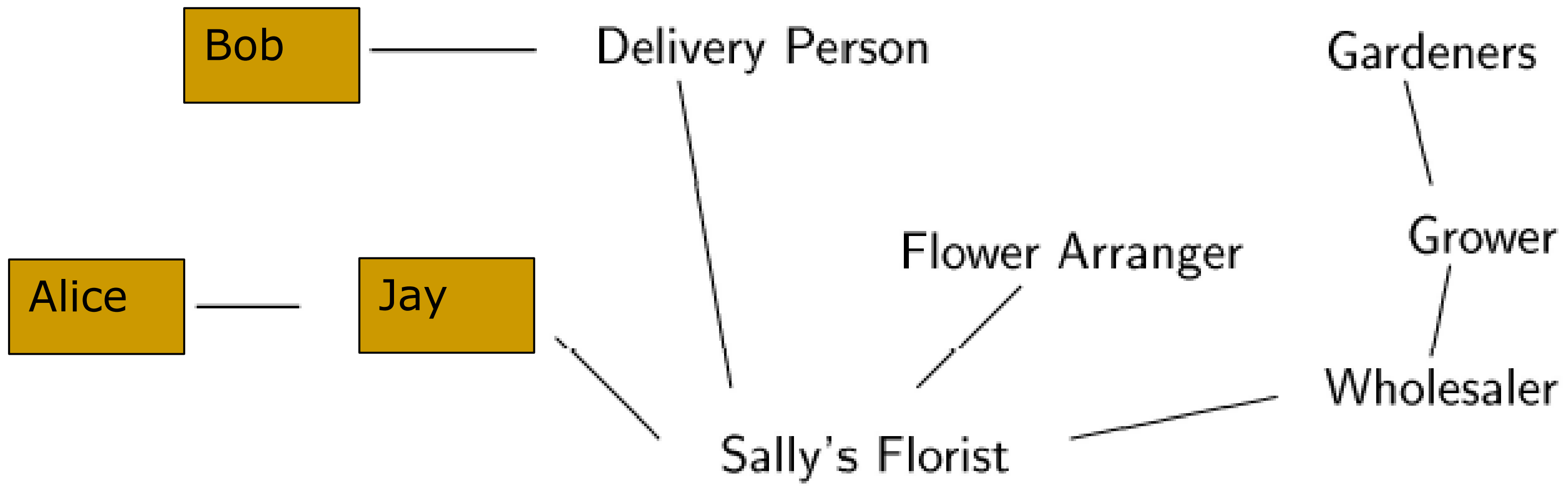


Figure: The community of agents (objects) helping Bob

## Thinking Object Oriented

- Definition 1: An object oriented program is structured as a community of interacting agents, called Objects. Each object has a role to play. Each object provides a service, or performs an action, that is used by other members of the community.
- Design and development using modular approach.
- Bottom-Up approach: Combining small components together to give rise to solutions to complex problems.
- Responsibilities: A fundamental concept in OOP is to describe the behaviour in terms of responsibilities. Alice's request for an action indicates only the desired outcome (sending flowers to Bob). Jay is free to pursue any technique that achieve the desired objective and is not hampered/effected by any interference from Alice.
- By discussing any problem in terms of responsibilities, we increase the level of abstraction. This permits greater independence between objects.

## Thinking Object Oriented

- **Classes, Objects, and Instances:** Although I have only dealt with Jay, a few times, I have a little idea of the ***behaviour*** I can expect when I go to his shop and present him with my request. However, I can make certain assumptions from the information I have about the florists in general, and I expect that Jay being an ***instance*** of this (florist) category, will fit the general pattern.
- In other words, we can use the term Florist to represent the category (***Class***) of all florists. Thus,
- All *objects* (agents) are instances of a class. The method invoked by an object in response to a message is determined by the class of the receiver. All objects of a given class use the same method in response to similar messages.

## Summary

- **Alan Kay, considered as father of OOP, identified the following characteristics as fundamental to OOP:**
  - Everything is an object.
  - Computation is performed by objects communicating with each other. Objects communicate by sending and receiving messages
  - Each object has its own memory, which consists of other objects.
  - Every object is an instance of a class (which must be an object). A class simply represents a grouping of similar objects.
  - The class is the repository for behavior associates with an object, i.e. all objects that are instances of the same class can perform the same actions.
  - Classes are organized into a single rooted tree structure, called the inheritance tree.  The memory and behavior associated with  instances of a class are automatically available to any class associated with a descendent in this tree structure.

## OOP vs Procedural Languages

| Procedural Oriented Programming | Object Oriented Programming |
|---|---|
| In procedural programming, program is divided into small parts called ***functions***. | In object oriented programming, program is divided into small parts called ***objects***. |
| Procedural programming follows ***top down approach***. | Object oriented programming follows ***bottom up approach***. |
| There is no access specifier in procedural programming. | Object oriented programming have access specifiers like private, public, protected etc. |
| Adding new data and function is not easy. | Adding new data and function is easy. |
| Procedural programming does not have any proper way for hiding data so it is ***less secure***. | Object oriented programming provides data hiding so it is ***more secure***. |
| In procedural programming, overloading is not possible. | Overloading is possible in object oriented programming. |
| In procedural programming, function is more important than data. | In object oriented programming, data is more important than function. |
| Procedural programming is based on ***unreal world***. | Object oriented programming is based on ***real world***. |
| Examples: C, FORTRAN, Pascal, Basic etc. | Examples: C++, Java, Python, C# etc. |

## C++ as OOP language

- C++ is an object-oriented, general purpose programming language derived from C.

- Existing code on C can be used with C++ (mode compatible). Even existing pre-compiled libraries can be used with new C++ code.

- Major improvements over C:

  a. Stream I/O

  b. In-lining

  c. Parameter passing by reference.

  d. Default argument values.

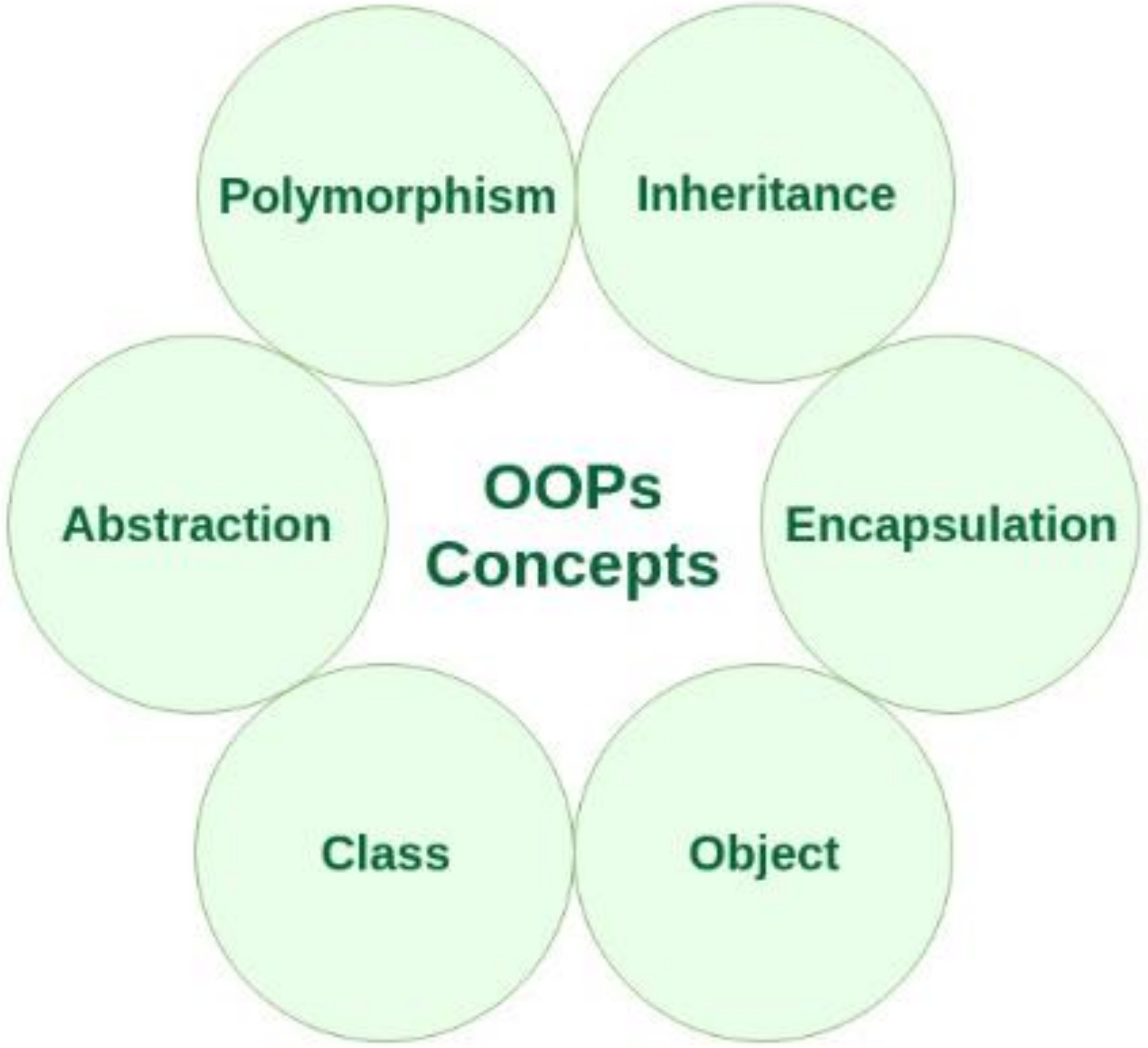
  e. OOP features.

## OOP principles

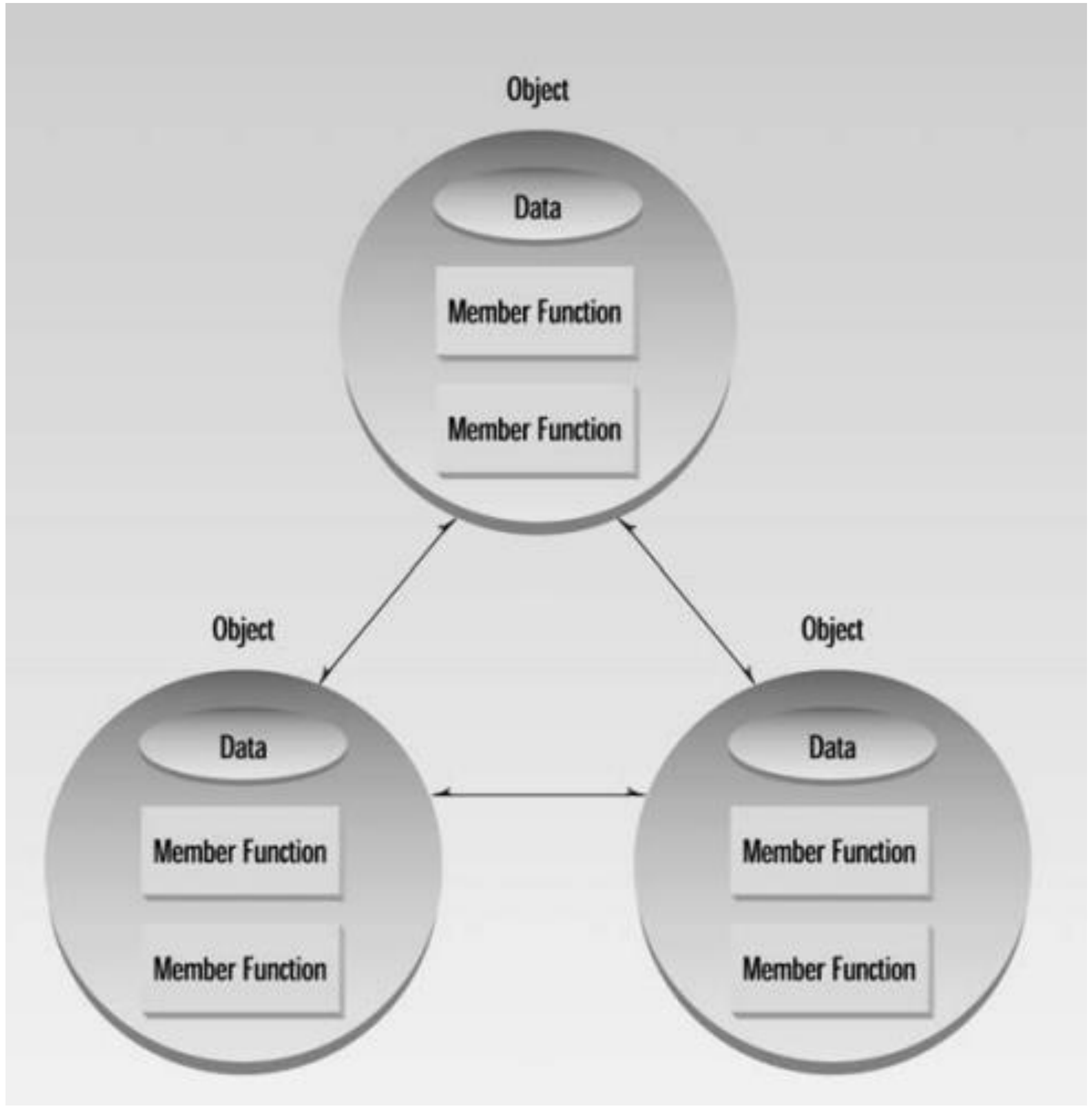
- **Encapsulation**: It is the process of hiding data members (variables, properties) and member functions (methods) into a single unit. Best example: Class.
- **Abstraction**: Refers to represent necessary features without including more details or explanation. Data abstraction relies on the separation of interface and implementation.
- **Inheritance**: The mechanism of deriving a new class from an old class is called inheritance.
- **Polymorphism**: Ability of object to take more than one form.

# The Object-Oriented Approach

- The fundamental idea behind object-oriented languages is to combine into a single unit both *data* and the *functions that operate on that data*. Such a unit is called an *object*.
- An object's functions, called *member functions* in C++, typically provide the only way to access its data. If you want to read a data item in an object, you call a member function in the object. It will access the data and return the value to you. You can't access the data directly.
- The data is *hidden*, so it is safe from accidental alteration. Data and its functions are said to be *encapsulated* into a single entity. *Data encapsulation* and *data hiding* are key terms in the description of object-oriented languages.
- If you want to modify the data in an object, you know exactly what functions interact with it: the member functions in the object. No other functions can access the data. This simplifies writing, debugging, and maintaining the program


Polymorphism
Inheritance
Abstraction
OOPs Concepts
Encapsulation
Class
Object

## *The object-oriented paradigm.*



## Characteristics of Object-Oriented Languages- Objects

What kinds of things become objects in object-oriented programs?

- **Physical objects**
  - Automobiles in a traffic-flow simulation
  - Electrical components in a circuit-design program
  - Countries in an economics model
  - Aircraft in an air traffic control system
- **Elements of the computer-user environment**
  - Windows
  - Menus
  - Graphics objects (lines, rectangles, circles)
  - The mouse, keyboard, disk drives, printer
- **Data-storage constructs**
  - Customized arrays
  - Stacks
  - Linked lists
  - Binary trees
- **Human entities**
  - Employees
  - Students
  - Customers
  - Salespeople

An object is often called an “instance” of a class.

## Characteristics of Objects

- **Identity:** makes an object different from other objects created from the same class.

- **State:**
  - defined by the contents of an object's attributes.
  - Objects state vary during the execution of program.

- **Behavior:** Defined by the messages (functions/methods) an object provides.

## Object Life Cycle

- Object Creation
- Object Usage
- Object Disposal

## Object Creation

- **Objects are created by instantiating a class.**

  ClassName object = new ClassName();

- **Object creation involves three actions:**

➢ **Declaration**: Declaring the type of data

➢ **Instantiation:** Creating a memory space to store the object

➢ **Initialization:** Initializing the attributes in the object.

## Characteristics of Object-Oriented Languages- Classes

- It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.

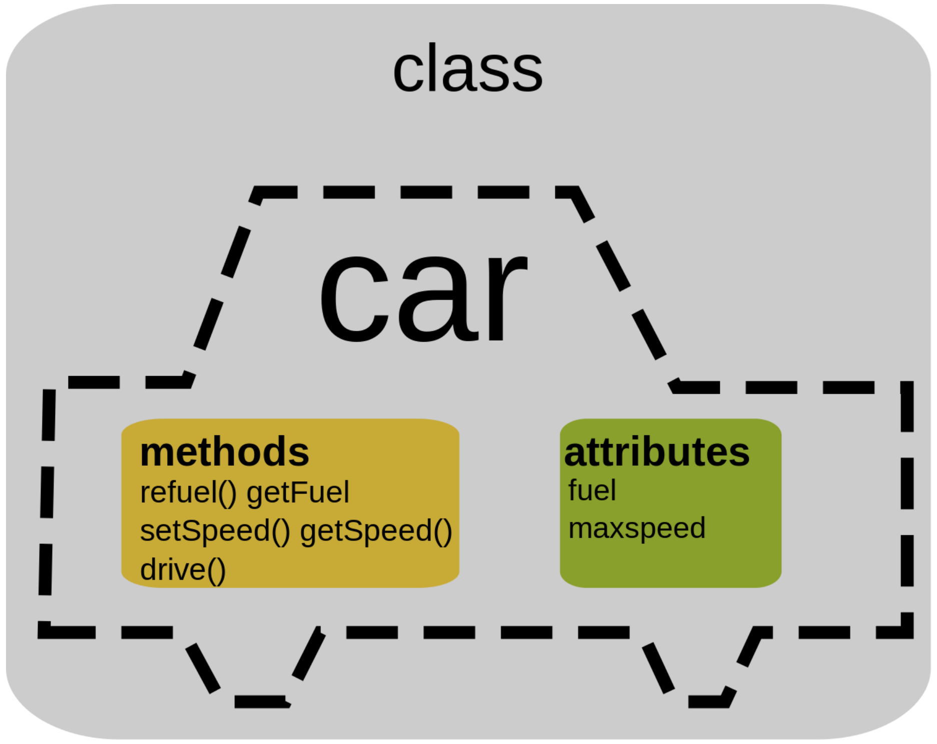
- For Example: Consider the Class of **Cars**. There may be many cars with different names and brand but all of them will share some common properties like all of them will have *4 wheels*, *Speed Limit*, *Mileage range* etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

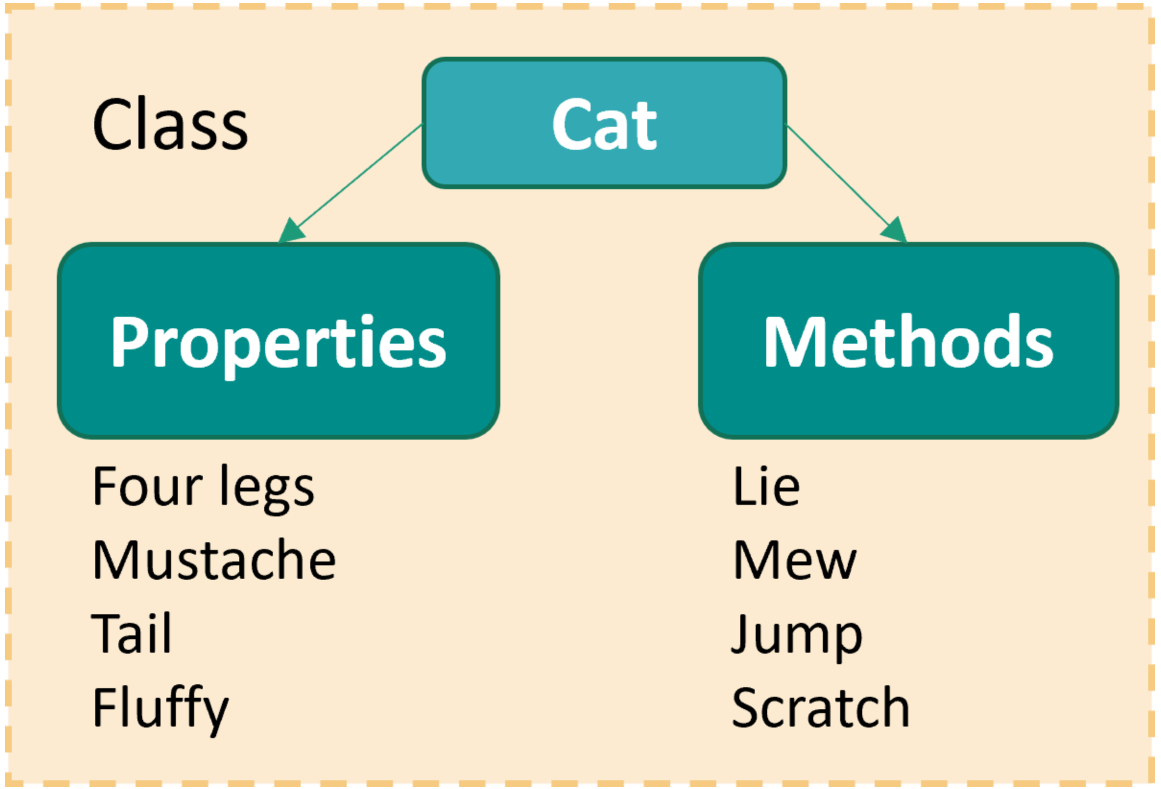
- A Class is a user defined data-type which has data members and member functions.

## Characteristics of Object-Oriented Languages- Classes

- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.
- In the above example of class *Car*, the data member will be *speed limit*, *mileage* etc. and member functions can be *apply brakes*, *increase speed* etc.
- Class serves as a plan, or blueprint. It specifies what data and what functions will be included in objects of that class.
- A class is thus a description of a number of similar objects.

## Example of class:

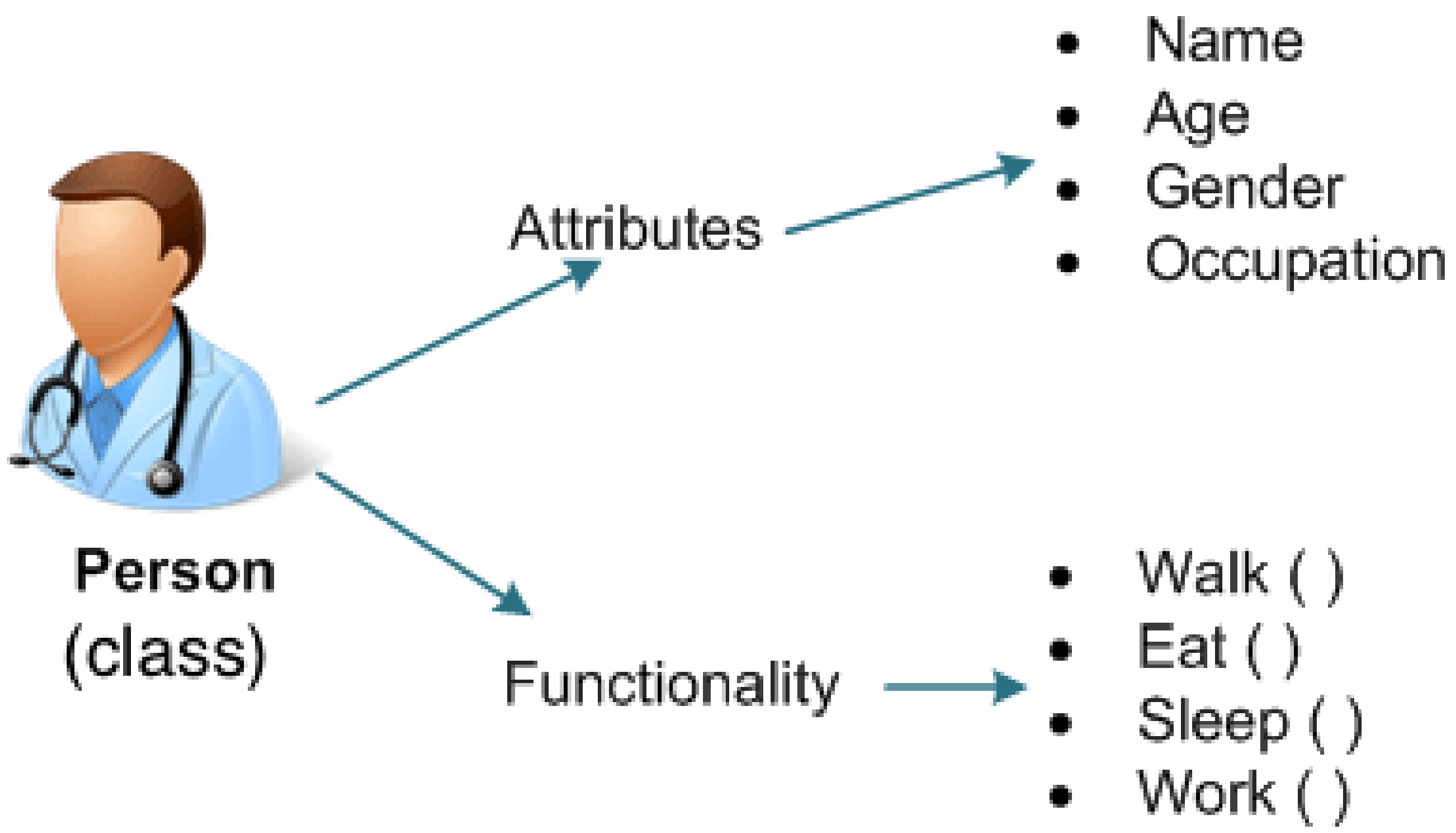




# Class Vs Object

- A class is a template/blueprint that describes the behaviors/states that object of its type support. It is a piece of the program's source code that describes a particular type of objects. OO programmers write class definitions.
- No memory is allocated for a class.
- Written by a programmer.
- Specifies the structure (the number and types) of its objects' attributes — the same for all of its objects.
- Specifies the possible behaviors of its objects.

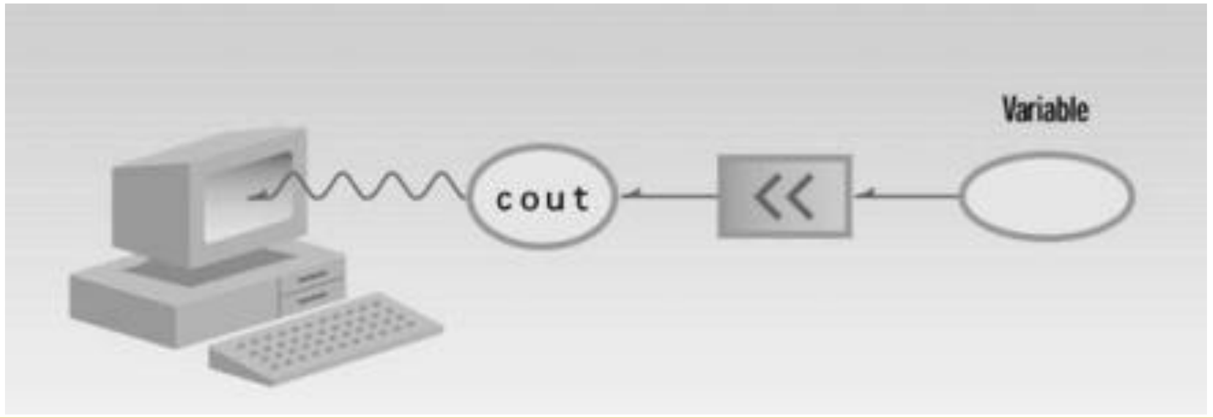
- An object is an instance of a class. A program can create and use more than one object (instance) of the same class.
- An entity in a running program.
- Created when the program is running (by the main method or a constructor or another method). Object is real and each object has its own memory.
- Holds specific values of attributes; these values can change while the program is running.
- Behaves appropriately when called upon.


Person
(class)
Attributes
Functionality
Name
Age
Gender
Occupation
Walk ( )
Eat ( )
Sleep ( )
Work ( )

## C++ Programming Basics: Basic Program Construction

```
#include <iostream> //preprocessor directive
using namespace std; //"using" directive
int main() //function name "main"
{ //start function body
cout << "Have a nice day!\n"; //statement
return 0; //statement
} //end function body
```

std::cout << "Have a nice day!\n";



## Writing first C++ program : Hello World example

```
// Header file for input output functions.
#include<iostream>
using namespace std;
// main function
int main()
 {
        // prints hello world
        cout<<"Hello World";
        return 0;
}
```

```
#include<iostream>
int main()
{
        // prints hello world
        std::cout<<"Hello World";
        return 0;
}
```

- The **#include** directive tells the compiler to include a file. It tells the compiler to include the standard iostream file which contains declarations of all the standard input/output library functions.
- The built in C++ library routines are kept in the standard namespace. That includes stuff like cout, cin, string, vector, map, etc. Because these tools are used so commonly, it's popular to add "using namespace std" at the top of your source code so that you won't have to type the std:: **prefix** constantly.
- In C++ input and output is performed in the form of a sequence of bytes or more commonly known as **streams**.
  - **Input Stream:** If the direction of flow of bytes is from device to the main memory then this process is called input.
  - **Output Stream:** If the direction of flow of bytes is opposite, i.e. from main memory to device then this process is called output.

- >> (insertion operator)
- << (extraction operator)
- Even if your compiler accepts “void main()” avoid it, or risk being considered ignorant by C and C++ programmers.
- namespace is a declarative region that provides a scope to the identifiers (names of the types, function, variables etc) inside it.

## Writing a Simple C++ Program

- Every C++ program contains one or more functions, one of which must be named main. The operating system runs a C++ program by calling main.

```
int main()   //Function name
{
return 0;     //Return value          //Function body.
}
```

- return is a statement that terminates a function.
- When a return statement includes a value, the value returned must have a type that is compatible with the return type of the function.

## Compiling and executing the program

- Save the program using suffix conventions such as .cc, .cxx, .cpp, .cp, and .C
- Assuming that we are compiling the program in a console window on a Linux system, we use `$ gcc prog1.c` to compile the program.
- Here, the Linux compiler put the executables in files named a.out. Thus, to run an executable on UNIX (Linux), we use the full file name, including the file extension: `$ a.out`
- If we need to specify the file's location, we'd use a "." followed by a forward slash to indicate that our executable is in the current directory: `$ ./a.out`

# Input/Output (IO)

- C++ includes an extensive standard library that provides IO (and many other facilities).

- Generally, iostream library is used for handling input and output. Fundamental to the iostream library are two types named istream and ostream, which represent input and output streams, respectively. (A stream is a sequence of characters read from or written to an IO device.)
- The iostream library The library defines four IO objects.
- To handle input, we use an object of type istream named cin
- To handle input we use an ostream object named cout

## Input/Output - Example

```
#include <iostream>
int main()
{
       std::cout << "Enter two numbers:" << std::endl;
       int v1 = 0, v2 = 0;  //int is the data type of variables
       std::cin >> v1 >> v2;
       std::cout << "The sum of " << v1 << " and " << v2
                 << " is " << v1 + v2 << std::endl;
     return 0;
}
```

# Input/Output - Example

- **#include <iostream>** tells the compiler that we want to use the iostream library. Every program that uses a library facility must include its associated header. Typically, we put all the #include directives for a program at the beginning of the source file.
- **<<** is an output operator. The left-hand operand must be an ostream object; the right-hand operand is a value to print.
- std::cout << "Enter two numbers:" << std::endl; can be replaced by:

```
std::cout << "Enter two numbers:";
std::cout << std::endl;
```

- The second operator prints endl, which is a special value called a **manipulator**. Writing endl has the effect of ending the current line and flushing the buffer associated with that device. Flushing the buffer ensures that all the output the program has generated so far is actually .

- In this program we use std::cout and std::endl rather than just cout and endl. The prefix std:: indicates that the names cout and endl are defined inside the **namespace named** std. Namespaces allow us to avoid inadvertent collisions between the names we define and uses of those same names inside a library. All the names defined by the standard library are in the std namespace.
- One side effect of the library's use of a namespace is that when we use a name from the library, we must say explicitly that we want to use the name from the std namespace. Writing std::cout uses the scope operator (the :: operator) to say that we want to use the name cout that is defined in the namespace std.
- The statement `std::cin >> v1 >> v2` is equivalent to:

```
std::cin >> v1;
std::cin >> v2;
```

## Exercise

- Write a program to print Hello, World on the standard output.
- The previous program used the addition operator, +, to add two numbers. Write a program that uses the multiplication operator, *, to print the product instead.

## Creating Class

To create a class, use the class keyword.

```
  class Student {
  int roll_number;   // Attribute

  string Name;  // Attribute

};
```

# Creating an object

- An object is created from a class.
- To create an object, specify class name followed by the object name.
- To access the class attributes, use the dot (.) syntax on the object.

```
Student  Obj1; // Create an object of Student class

 Obj1. roll_number = 10;
 Obj1.Name = “Alice”;
```

## Class methods - Defining a method: 1

- Methods are functions that belongs to the class. They define the behavior/action taken by the object.

```
class Student
{
    public:  // Access specifier (will be explained in Unit-II)
        void Method1()
        {   // defining the method
            cout<<“Hello World”;
        }
};
```

## Class methods - Defining a method: 1

```
class Student
{
      public: // Access specifier (will be explained in Unit-II)
      void Method1();      // Method declaration
 };

 void Student::Method1() // Defining outside of the class
{
     cout<<“Hello World”;
}
```

## Class Methods: Example

```
class Student{
Public:
 // Declaration of state/properties.
  string name;         //instance variable
  int roll_number;     //instance variable
  static int age;      //instance variable
// Declaration of Actions
void display_details()
{  // Instance method.
//Method Body
 }
};
```

## Class work- Create a class and its members for the following scenario.

1. Create a class that captures students. Each student has a first name, last name, class year, and major. Create two examples of students.

2. Create a class that captures planets. Each planet has a name, a distance from the sun, and its gravity relative to Earth’s gravity. For distance and gravity, use the type double which captures real numbers. Make objects for Earth and your favorite non-earth planet.

## Self practice- Create a class and its members for the following scenario.

Object- Person

State/Properties:

Black hair

5.10 height

Weighs 70 kg

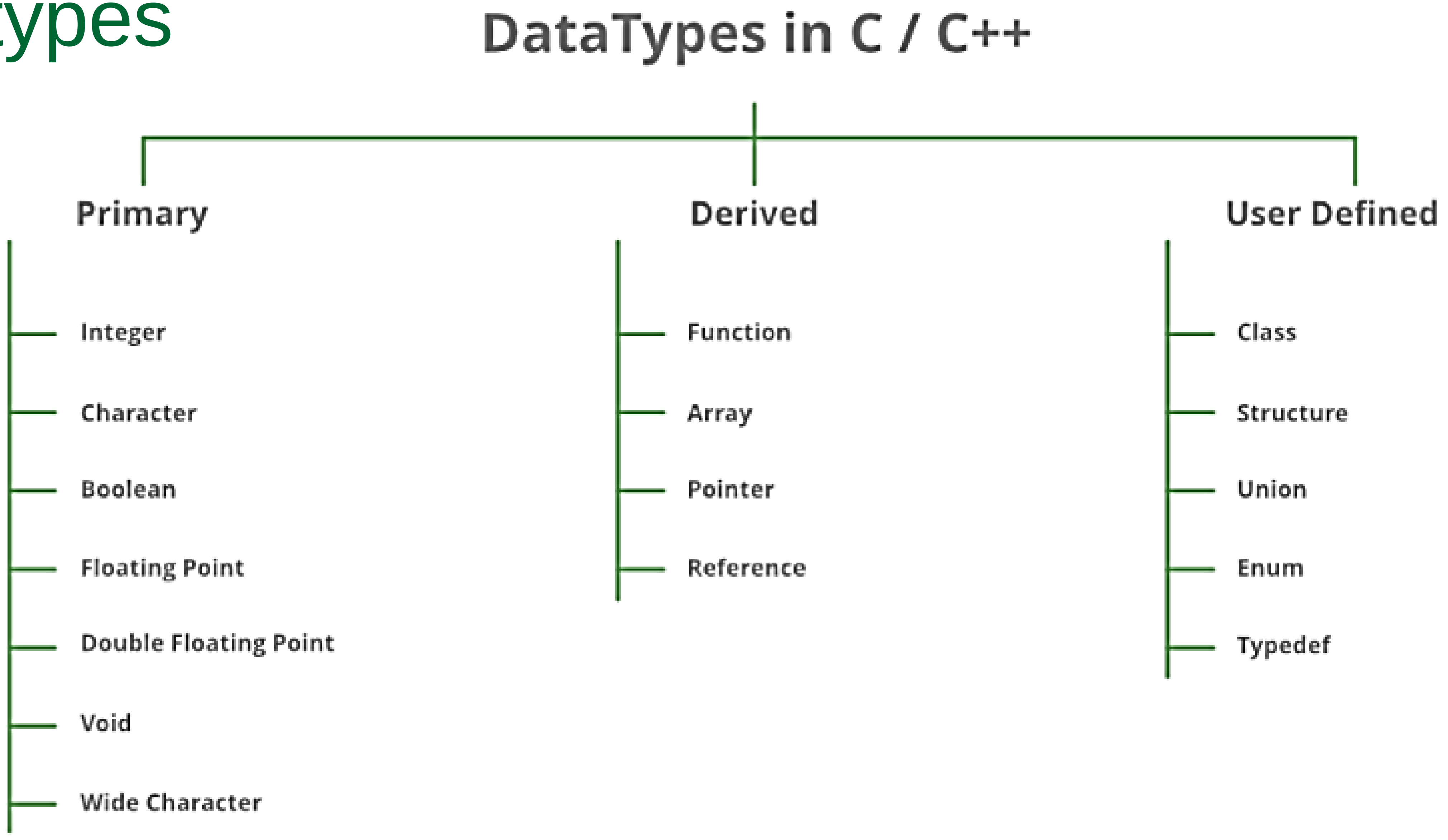
Behaviour/Action:

Eat

Sleep

Run

## Datatypes


DataTypes in C / C++
Primary
Integer
Character
Boolean
Floating Point
Double Floating Point
Void
Wide Character
Derived
Function
Array
Pointer
Reference
User Defined
Class
Structure
Union
Enum
Typedef

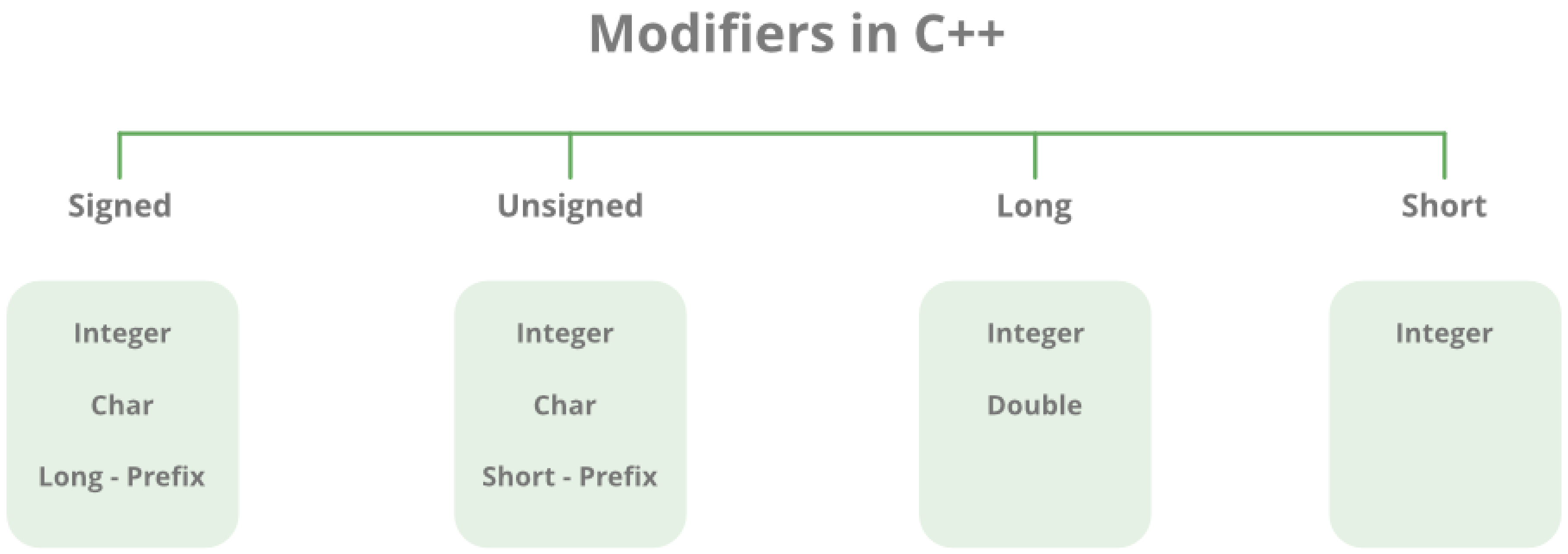
## Datatypes

- **Integer**: Keyword used for integer data types is int. Integers typically requires 4 bytes of memory space and ranges from -2147483648 to 2147483647.
- **Character**: Character data type is used for storing characters. Keyword used for character data type is char. Characters typically requires 1 byte of memory space and ranges from -128 to 127 or 0 to 255.
- **Boolean**: Boolean data type is used for storing boolean or logical values. A boolean variable can store either true or false. Keyword used for boolean data type is bool.
- **Floating Point**: Floating Point data type is used for storing single precision floating point values or decimal values. Keyword used for floating point data type is float. Float variables typically requires 4 byte of memory space.

## Datatypes

- **Double Floating Point**: Double Floating Point data type is used for storing double precision floating point values or decimal values. Keyword used for double floating point data type is **double**. Double variables typically requires 8 byte of memory space.
- **void**: Void means without any value. void datatype represents a valueless entity. Void data type is used for those function which does not returns a value.
- **Wide Character**: Wide character data type is also a character data type but this data type has size greater than the normal 8-bit datatype. Represented by **wchar_t**. It is generally 2 or 4 bytes long.

## Datatype Modifiers



| Data Type | Size (in bytes) | Range |
|---|---|---|
| short int | 2 | -32,768 to 32,767 |
| unsigned short int | 2 | 0 to 65,535 |
| unsigned int | 4 | 0 to 4,294,967,295 |
| int | 4 | -2,147,483,648 to 2,147,483,647 |
| long int | 4 | -2,147,483,648 to 2,147,483,647 |
| unsigned long int | 8 | 0 to 4,294,967,295 |
| long long int | 8 | -(2^63) to (2^63)-1 |
| unsigned long long int | 8 | 0 to 18,446,744,073,709,551,615 |
| signed char | 1 | -128 to 127 |
| unsigned char | 1 | 0 to 255 |
| float | 4 | |
| double | 8 | |
| long double | 12 | |
| wchar_t | 2 or 4 | 1 wide character |

## C++ Variables

```
type variable = value;
```

- Variables are containers for storing data values.
  - •int - stores integers (whole numbers), without decimals, such as 123 or -123
  - •double - stores floating point numbers, with decimals, such as 19.99 or -19.99
  - •char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
  - •string - stores text, such as "Hello World". String values are surrounded by double quotes
  - •bool - stores values with two states: true or false

```
int myNum = 5;                // Integer (whole number)
float myFloatNum = 5.99;      // Floating point number
double myDoubleNum = 9.98;    // Floating point number
char myLetter = 'D';          // Character
bool myBoolean = true;        // Boolean
string myText = "Hello";      // String
```

## C++ Constants/Literals

- Constants refer to fixed values that the program may not alter and they are called **literals**.
- Constants can be of any of the basic data types and can be divided into Integer Numerals, Floating-Point Numerals, Characters, Strings and Boolean Values.
- Again, constants are treated just like regular variables except that their values cannot be modified after their definition.
- There are two simple ways in C++ to define constants −
- Using **#define** preprocessor. #define identifier value
- Using **const** keyword. const type variable = value;

## C++ Constants/Literals

```
#include <iostream>
using namespace std;
#define LENGTH 10
#define WIDTH 5
#define NEWLINE '\n'
int main()
{
  int area;
  area = LENGTH * WIDTH;
  cout << area;
  cout << NEWLINE;
  return 0;
  }
```

```
#include <iostream>
using namespace std;
int main()
{
   const int LENGTH = 10;
   const int WIDTH = 5;
   const char NEWLINE = '\n';
   int area;
   area = LENGTH * WIDTH;
   cout << area;
   cout << NEWLINE;
   return 0;
}
```

## Type Conversion in C++

- A type cast is basically a conversion from one type to another. There are two types of type conversion:
- Implicit Type Conversion/Automatic type conversion
  - Done by the compiler on its own, without any external trigger from the user.
  - Generally, takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.
  - All the data types of the variables are upgraded to the data type of the variable with largest data type.

  bool -> char -> short int -> int -> unsigned int -> long -> unsigned -> long long -> float -> double -> long double

  - It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

# Type Conversion in C++

**Explicit Type Conversion:** This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type. It can be done by two ways:

**Converting by assignment:** This is done by explicitly defining the required type in front of the expression in parenthesis. This can be also considered as forceful casting.

- Syntax: (type) expression

**Conversion using Cast operator:** A Cast operator is an **unary operator** which forces one data type to be converted into another data type.
C++ supports four types of casting:

- Static Cast
- Dynamic Cast
- Const Cast
- Reinterpret Cast

## Example

```
#include <iostream>
using namespace std;
int main()
{int x = 10; // integer x
char y = 'a'; // character c
// y implicitly converted to int. ASCII value of
'a' is 97
x = x + y; // x is implicitly converted to float
float z = x + 1.0;
cout << "x = " << x << endl<< "y = " << y <<
endl<< "z = " << z << endl;
return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
        double x = 1.2;
// Explicit conversion from double
to int
        int sum = (int)x + 1;
        cout << "Sum = " << sum;
        return 0;
}
```

## Operators

## Operators

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- sizeof operator

## Arithmetic Operators

| Operator | Description |
|---|---|
| + addition operator | Adds to operands. |
| - subtraction operator | Subtracts 2nd operand from 1st operand. |
| * multiplication operator | Multiplies 2 operands. |
| / division operator. | Divides numerator by denominator. |
| % modulus operator | Returns remainder after division. |
| ++ increment operator | Increases an integer value by 1. |
| -- decrement operator. | Decreases an integer value by 1. |

## Program to demonstrate Arithmetic Operators

```
#include <iostream>
using namespace std;
int main() {
        int a = 11;
        int b = 5;
        int c;
        cout << "a + b is :" << a+b << endl; //11+5
        cout << "a - b is :" << a-b << endl; //11-5
        cout << "a * b is :" << a*b << endl; //11*5
        cout << "a / b is :" << a/b << endl; //11/5
        cout << "a % b is :" << a%b << endl; //11%5
        cout << "a++ is :" << a++ << endl; //11++
        cout << "a-- is :" << a-- << endl; //12--
        return 0;
}
```

# Relational Operators

| **Operator** | **Description** |
|---|---|
| == equal to operator. | Checks equality of two operand values. |
| != not equal to operator | Checks equality of two operand values. |
| > great than operator | Checks whether value of left operand is greater than value of right operand. |
| < less than operator. | Checks whether value of left operand is less than value of right operand. |
| >= greater than or equal to operator | Checks whether value of left operand is greater than or equal to value of right operand. |
| <= less than or equal to operator. | Checks whether value of left operand is less than or equal to value of right operand. |

## Program to demonstrate Relational Operators

```
#include <iostream>
using namespace std;
int main() {
        int a = 11;
        int b = 5;
        cout << "a=11, b=5" << endl;
        if (a == b) {
        cout << "a == b is true" << endl;
        }
        else {
        cout << " a == b is false" << endl;
        }
```

```
if (a < b) {
        cout << "a < b is true" << endl;
        }
else {
        cout << "a < b is false" << endl;
        }
if (a > b) {
        cout << "a > b is true" << endl;
        }
else {  cout << "a > b is false" << endl;
        }
        return 0;
}
```

## Logical Operators

| Operator | Description |
|---|---|
| && logical AND operator. | The condition is true if both operands are not zero. |
| \|\| logical OR operator. | The condition is true if one of the operands is non-zero. |
| ! logical NOT operator. | It reverses operand's logical state. If the operand is true, the ! operator makes it false. |

## Program to demonstrate Relational Operators

```
#include <iostream>
using namespace std;
int main()
{
        int a = 5, b = 2, c = 6, d = 4;
        if (a == b && c > d)
                cout << "a equals to b AND c is greater than d\n";
        else
                cout << "AND operation returned false\n";
        if (a == b || c > d)
                cout << "a equals to b OR c is greater than d\n";
        else
                cout << "Neither a is equal to b nor c is greater than d\n";
        if (!b)
                cout << "b is zero\n";
        else
                cout << "b is not zero";
        return 0;
}
```

## Bitwise Operators

| Operator | Description |
|---|---|
| & (bitwise AND). | It takes 2 numbers (operands) then performs AND on each bit of two numbers. If both are 1, AND returns 1, otherwise 0. |
| \| (bitwise OR) | Takes 2 numbers (operands) then performs OR on every bit of two numbers. It returns 1 if one of the bits is 1. |
| ^ (the bitwise XOR) | Takes 2 numbers (operands) then performs XOR on every bit of 2 numbers. It returns 1 if both bits are different. |
| << (left shift) | Takes two numbers then left shifts the bits of the first operand. The second operand determines total places to shift. |
| >> (right shift) | Takes two numbers then right shifts the bits of the first operand. The second operand determines number of places to shift. |
| ~ (bitwise NOT). | Takes number then inverts all its bits. |

## Program to demonstrate Bitwise Operators

```
#include <iostream>
using namespace std;
int main() {
        unsigned int p = 60;     // 60 = 0011 1100
        unsigned int q = 13;     // 13 = 0000 1101
        int z = 0;
        z = p & q;
        cout << "p&q is : " << z << endl; // 12 = 0000 1100
        z = p | q;
        cout << "p|q is : " << z << endl; // 61 = 0011 1101
        z = p ^ q;
        cout << "p^q is : " << z << endl; // 49 = 0011 0001
        z = ~p;
        cout << "~p is : " << z << endl; // -61 = 1100 0011
        z = p << 2;
        cout << "p<<2 is: " << z << endl; // 240 = 1111 0000
        z = p >> 2;
        cout << "p>>2 is : " << z << endl; // 15 = 0000 1111
        return 0;
}
```

## Program to demonstrate sizeof operator

This operator determines a variable's size. Use sizeof operator to determine the size of a data type.

```
#include <iostream>
using namespace std;
int main() {
        cout<<"Size of int : "<< sizeof(int) << "\n";
        cout<<"Size of char : " << sizeof(char) << "\n";
        cout<<"Size of float : " << sizeof(float) << "\n";
        cout<<"Size of double : " << sizeof(double) << "\n";
        return 0;
}
```

## Program to demonstrate conditional/ternary operator

**Condition ? Expression1 : Expression2;**

- The Condition is the condition that is to be evaluated.
- Expression1 is the expression to be executed if condition is true.
- Expression2 is the expression to be executed if condition is false.
- This operator evaluates a condition and acts based on the outcome of the evaluation.

```
#include <iostream>
using namespace std;
int main() {
        int a = 1, b;
        b = (a < 10) ? 2 : 5;
        cout << "value of b: " << b << endl;
        return 0;
}
```