Imperative Paradigm: Data Abstraction in Object Orientation

Module 2

Basic Structure of a C++ program

```
/* My first program */

#include <iostream>

int main()

{

  std::cout << "Hello World!";

}
```

- Comments - // and /* */
- # preprocessor directive
- iostream - header not a header file
- Namespaces resolves identifier name conflicts.
- scope operator (::)
- stdcout - standard character output device
- insertion operator (<<)
-

Displaying the sum of two numbers

```
#include <iostream>

using namespace std;

int main()
{

    int a,b;

   std:cout<<"Enter 1st no"<<endl;
   std::cin>>a;

   std::cout<<"Enter 2nd no"<<endl;
   std::cin>>b;

   int sum= a+b;

   cout<<"The sum of"<<a<<"and"<<b<<"is<<sum;    //  like C print statement - printf("the sum of %d and %d is %d",a,b,sum);
   return 0;
}
```

- How variables are displayed
- Control structures used in C++ are same as the ones used in C

Try Yourself Write C++ programs to

1. Find the sum of Even and Odd number between 1 to 100
   (use looping control statements - for/while/do while)

2. Find if the input character is a vowel or not.
   (use switch control statment)

Modular programming and default arguments

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

int divide (int a, int b=2)
{
  int r;
  r=a/b;
  return (r);
}
int main ()
{

  cout << divide (20,4);
  cout << divide (12);
  cout << endl;
  return 0;
}
```

- C++ follows procedural programming
- Also object oriented style
- C++ is not a pure Object Oriented

Defining Class and Creating Objects

- OOP provides Encapsulation, Abstraction and Data Hiding

```
class ClassName
{    Access specifier:  Data members;

     Member Functions()
     {        // member function definition
     }
};
```

- class keyword
- ; at the end
- Encapsulation (state + behaviour: variables+ functions)

Access specifiers

- Data Hiding
- 3 types
  - Private (by default all members in a class are private)
  - public
  - protected

- Access specifiers - control where the function or data member can be used
  - Inside class only
  - Anywhere in the scope of program
  - Only in derived classes

```
#include <iostream>

using namespace std;


class Student
{         private: char name[20];
          int roll;

         public: void read();
          void disp();
};

void Student::read()
{
       cout<<"\nEnter the student Name :: ";
       cin>>name;
       cout<<"\nEnter the student roll no :: ";
       cin>>roll;

}
```

- Data members are declared in class
- Function members are declared in class, but can be defined inside or outside the scope of class
- Data members are used as normal variables in a scope of class
    - Object.variable ✗
    - variable ✓
- Private members can be access obly through public functions

```
 void Student :: disp()
{
   cout<<"\nThe Entered Student Details are shown below ::---------- \n";
      cout<<"\nStudent Name :: "<<name<<endl;
      cout<<"\nRoll no   is :: "<<roll<<endl;

}

int main()
{

   Student s1;

   s1.read();
   s1.disp();


   return 0;
}
```

- Objects allocate storage space base on the total size of all the data members
- Member function occupy common storage.
- No separate storage is allocated for all the objects for functions
- Member functions can be access by an object only ie an object is passed implicitly always (expectation friend/ static functions)

Try Yourself

Add a function to calculate percentage in the previous Student program

Demonstration of private and public access specifiers

```
#include <iostream>

using namespace std;

class Student
{   private:char name[20];
            int roll;

     private:void read()
            {
            cout<<"\nEnter the student Name :: ";
            cin>>name;
            cout<<"\nEnter the student roll no :: ";
            cin>>roll;

            }
    public:void disp();
};
```

- Private members are available only in the scope of class
- Public members can be used anywhere
- Private members can be access only through public functions

```
 void Student :: disp()
{       //to call private memebers we need public methods
        //implicit object calling it
        read();

        cout<<"\nThe Entered Student Details are shown below ::---------- \n";
        cout<<"\nStudent Name :: "<<name<<endl;
        cout<<"\nRoll no   is :: "<<roll<<endl;
 }

int main()
{
   // classname objajectname;
    Student s1;

    s1.disp();

    return 0;
}
```

Passing of arguments to functions

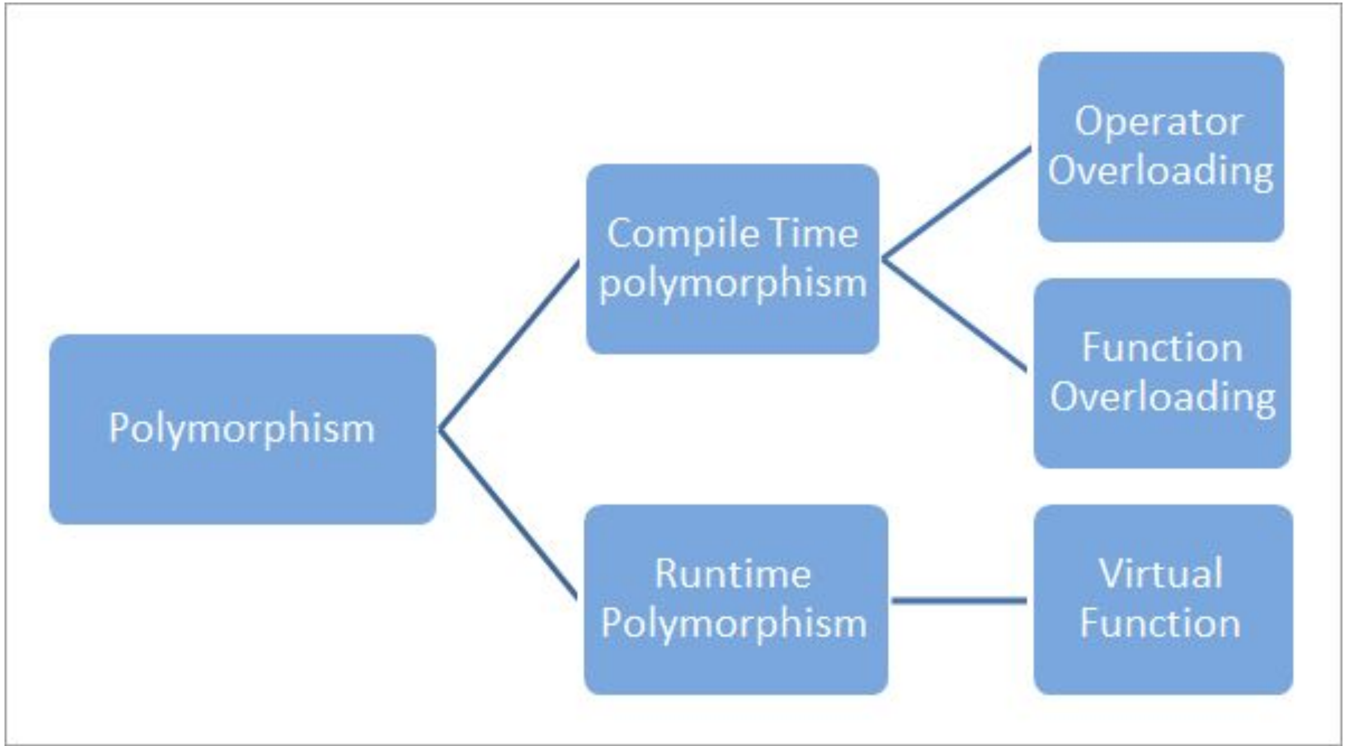
- Arguments can be passed to functions
- Both call by value and call by reference

```
Friend function
Class Complex
{ …….

    void display()
    {
        cout<<real<<"+"<<imag<<"i"<<endl;
    }

    friend Complex sum(Complex c1, Complex c2);

    };


  Complex sum(Complex c1,Complex c2)
    {
        Complex temp;
        temp.real=c1.real+c2.real;
        temp.imag=c1.imag+c2.imag;
        return(temp);
    }
```

- Friend function of a class is defined outside that class scope but it has the right to access all private and protected members of the class.
- prototypes for friend functions appear in the class definition
- Friend functions are not member functions
- Friend functions are used when you don’t want to call with objects or the function involves 2 or more objects of different type
-

Polymorphism

- Polymorphism means having multiple forms of one thing



Polymorphism

- poly" means many and morphs means forms.

    Compile Time Polymorphism

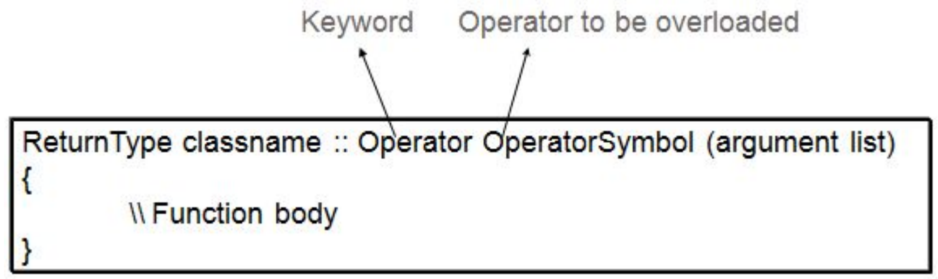
    - Binding takes place at compile time

    Runtime Polymorphism

    - Binding takes place a run time

https://www.youtube.com/watch?v=_aaXc-096tg
6:04 - compile and run time
9:13 - virtual functions
11:10 pure virtual function

Operator overloading



```
ReturnType classname :: Operator OperatorSymbol (argument list)
{
        \\ Function body
}
```

- It is a type of polymorphism in which an operator is overloaded to give user defined meaning to it.
- Overloaded operator is used to perform operation on user-defined data type.
- For example '+' operator can be overloaded to perform addition on various data types, like for Integer, String(concatenation) etc
- Operator overloading function can be a member function if the Left operand is an Object of that class, but if the Left operand is different, then Operator overloading function must be a non-member function or friend

- **Function overloading**

- Function Overloading is defined as the process of having two or more function with the same name
- In function overloading, the function is redefined by using either different types of arguments, sequence of arguments or a different number of arguments.
- Compiler can differentiate between the functions.
- Functions having the same name and same parameter list but different return type is not an overloaded function

Constructor

- special class functions which performs initialization of every object.
- called whenever an object is created.
- No return value
- Same name as class name
- Constructor overloading

Destructor

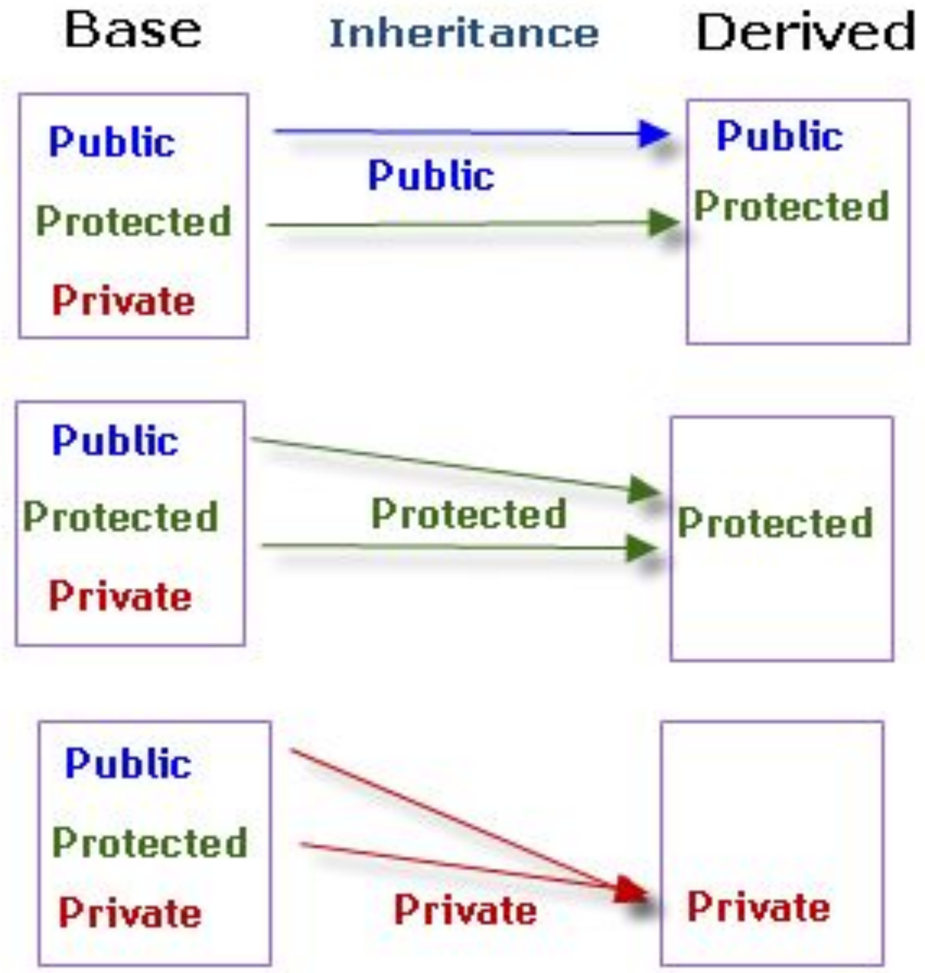
- used to destroy the object.
- Called when the object is destroyed or scope of object is exited
- Same name of class preceded by ~

Inheritance

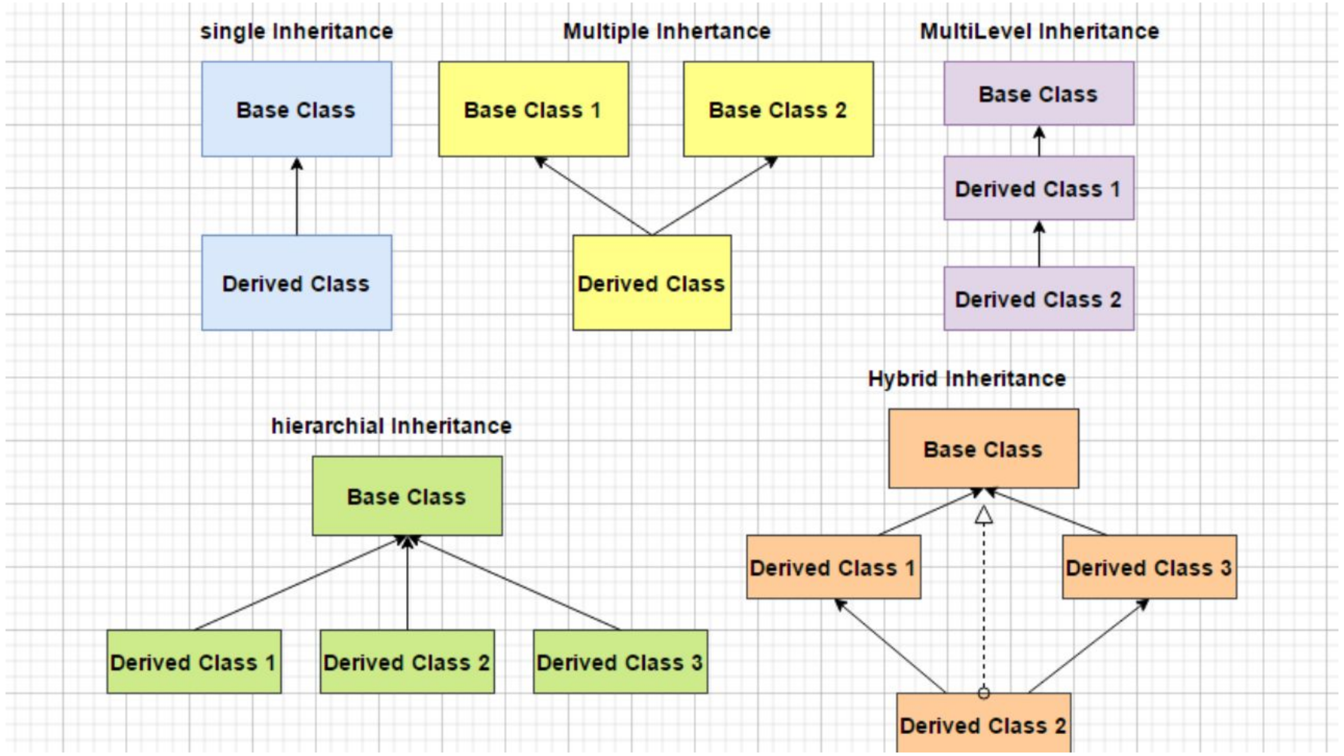
- The capability of a class to derive properties and characteristics from another class is called **Inheritance**.
- Encourages **Reusability**
- The class that inherits properties from another class - Derived class, Sub class, Child class (specialized class)
- The inheriting class - Base class, Parent class (Generalized class)
- IS A relationship

Syntax:

```
class parent_class
{
    //Body of parent class
};
class child_class : access_modifier parent_class
{
   //Body of child class
}
```


Base
Inheritance
Derived
Public
Protected
Private
Public
Public
Protected
Public
Protected
Private
Protected
Protected
Public
Protected
Private
Private
Private

Types of Inheritance



Calling of constructors in inheritance

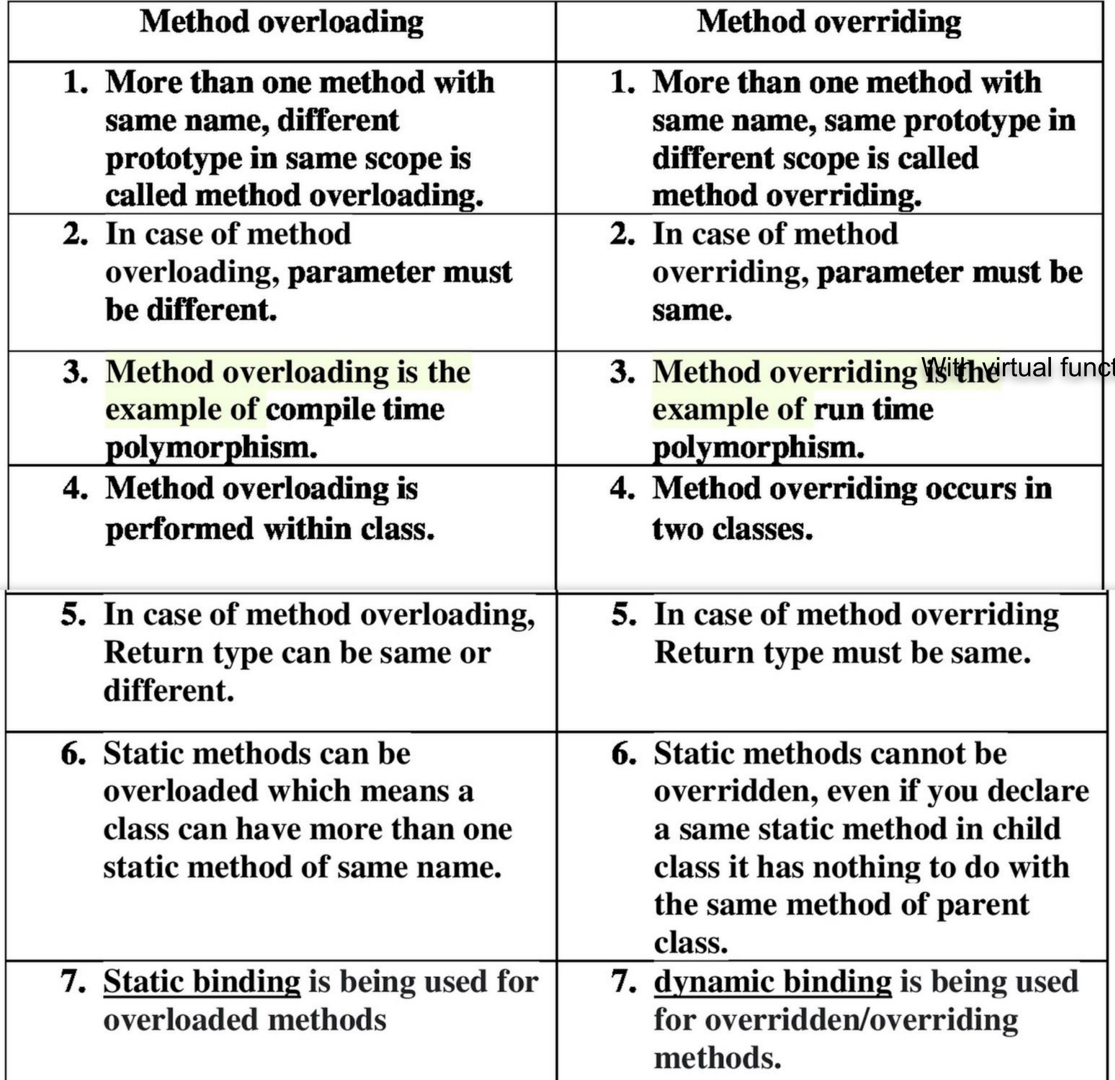
- Base class **constructors** are always called in the derived class **constructors**.
- Whenever you create derived class object, first the base class default **constructor** is executed and then the derived class's **constructor** finishes execution

Overriding

- derived class defines same function as defined in its base class

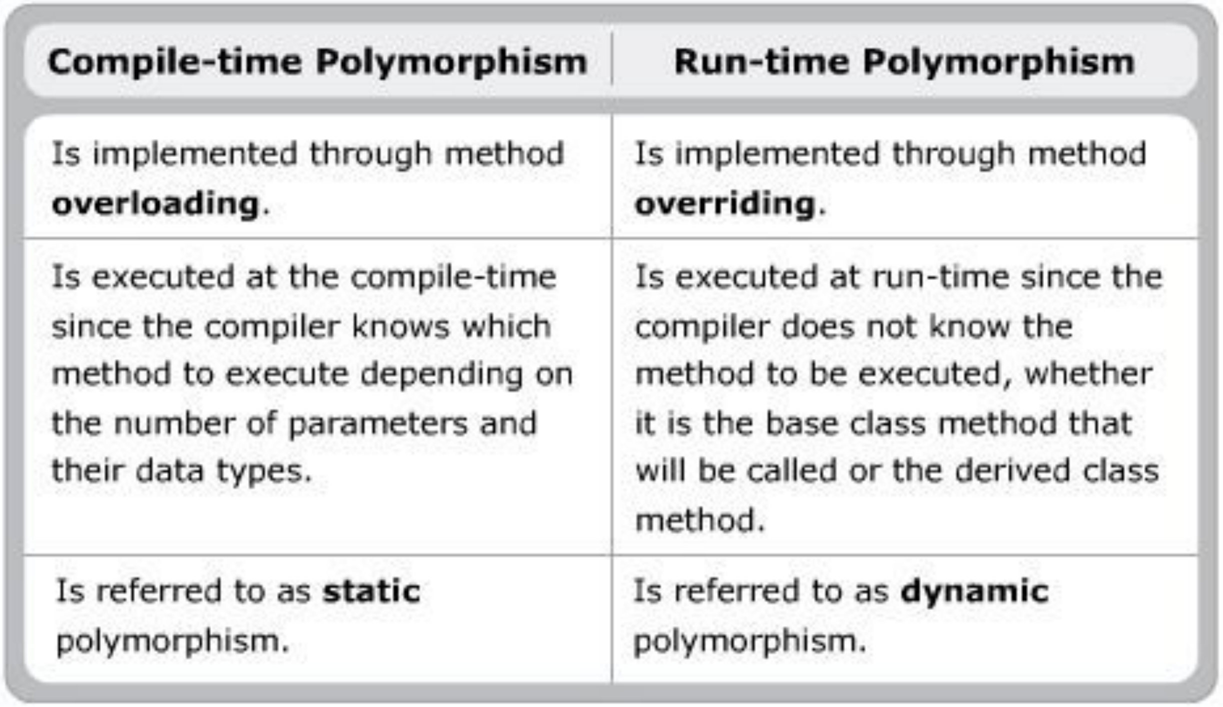
Virtual functions

- Is a member function in the base class that you redefine in a derived class.
- tells the compiler to perform dynamic linkage or late binding on the function.
- Use the single pointer to refer to all the objects of the different classes. So, we create the pointer to the base class that refers to all the derived objects. But, when base class pointer contains the address of the derived class object, always executes the base class function. This issue can only be resolved by using the 'virtual' function.
- A 'virtual' is a keyword preceding the normal declaration of a function.
- When the function is made virtual, C++ determines which function is to be invoked at the runtime based on the type of the object pointed by the base class pointer.
- Pure virtual functions
  - function declared in the base class that has no definition relative to the base class.
  - A class containing the pure virtual function cannot be used to declare the objects of its own
  - provides the traits to the derived classes and to create the base pointer used for achieving the runtime polymorphism.



Method overloading	Method overriding
1. More than one method with same name, different prototype in same scope is called method overloading.	1. More than one method with same name, same prototype in different scope is called method overriding.
2. In case of method overloading, parameter must be different.	2. In case of method overriding, parameter must be same.
3. Method overloading is the example of compile time polymorphism.	3. Method overriding is the example of run time polymorphism. With virtual function
4. Method overloading is performed within class.	4. Method overriding occurs in two classes.
5. In case of method overloading, Return type can be same or different.	5. In case of method overriding Return type must be same.
6. Static methods can be overloaded which means a class can have more than one static method of same name.	6. Static methods cannot be overridden, even if you declare a same static method in child class it has nothing to do with the same method of parent class.
7. Static binding is being used for overloaded methods	7. dynamic binding is being used for overridden/overriding methods.

Difference between Compile time and Runtime



Compile-time Polymorphism	Run-time Polymorphism
Is implemented through method **overloading**.	Is implemented through method **overriding**.
Is executed at the compile-time since the compiler knows which method to execute depending on the number of parameters and their data types.	Is executed at run-time since the compiler does not know the method to be executed, whether it is the base class method that will be called or the derived class method.
Is referred to as **static** polymorphism.	Is referred to as **dynamic** polymorphism.

Pure virtual function

- Pure virtual Functions are virtual functions with no definition.
- They start with virtual keyword and ends with = 0

Abstract Classes

- Abstract Class is a class which contains atleast one Pure Virtual function in it.
- Abstract classes are used to provide an Interface for its sub classes
- cannot be instantiated
- Used for upcasting

Self Study topics

- Default Arguments
  - https://www.youtube.com/watch?time_continue=1443&v=uJGmGAShHeU&feature=emb_title
  - Watch 5:50 -24:00
- Copy Constructors
  - https://www.youtube.com/watch?v=jXTTOZUT1iU&list=PL0gIV7t6l2iIsR55zsSgeiOw9Bd_IUTbY&index=27
  - Watch 22:15 -30:13
- Static Data member and member function
  - https://www.youtube.com/watch?v=l1JYbPhh9Vw&list=PL0gIV7t6l2iIsR55zsSgeiOw9Bd_IUTbY&index=32
  - Watch 00:00 - 27:16

Text Book :

Scott M L, Programming Language Pragmatics, 3rd Edn., Morgan Kaufmann Publishers, 2009 (Chapter No. 9)

Online resources

- https://nptel.ac.in/courses/106/105/106105151/
- https://www.javatpoint.com/cpp-tutorial

Online editor for C++

https://www.tutorialspoint.com/compile_cpp_online.php

https://www.onlinegdb.com/online_c++_compiler