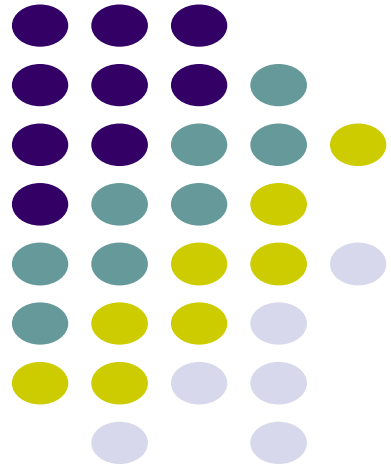
Object Oriented Programming in C++

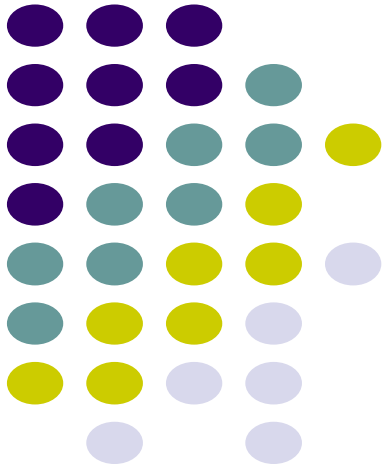
Presented by Bhimashankar T

E-Mail: bhima.t@gmail.com

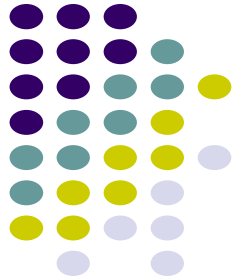
PhNo:+91 9980156833 / 6363863430



DAY 03

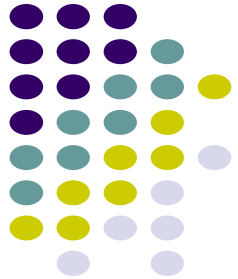


Inheritance – walk through the codes

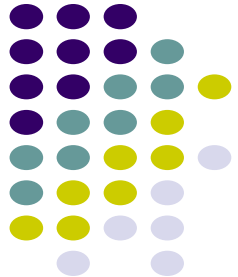


Codes for Inheritance (click here)

Passing values to base class – walk through the codes



Passing values to base class (click here)



The new operator

- allocates memory on the heap at runtime for an object, or a primitive data type, and returns a pointer to the object or the primitive data type thus allocated.

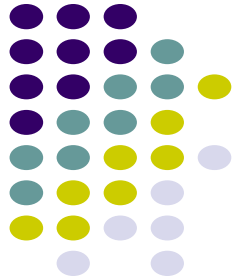
Eg:

```
int *p = new int;

    int *pia = new int[4]; //  allocates an array of four integer
```

elements.

- In addition to allocating memory, **new** also creates an object by calling the object's constructor.
- The object actually allocated on the free store is un-initialized. We can specify an initial value by: int *pi=new int(1024);



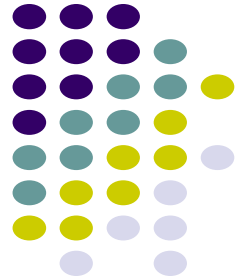
The delete operator

Frees the memory occupied by an object on the heap previously allocated to it using **new**.

```
delete p;      //deletes a single object
delete [] pia;     //deletes an array of objects
```

In addition to de-allocating memory occupied by an object, **delete** also destroys the object by calling the object's destructor.

Memory allocated using new should be freed only using delete.

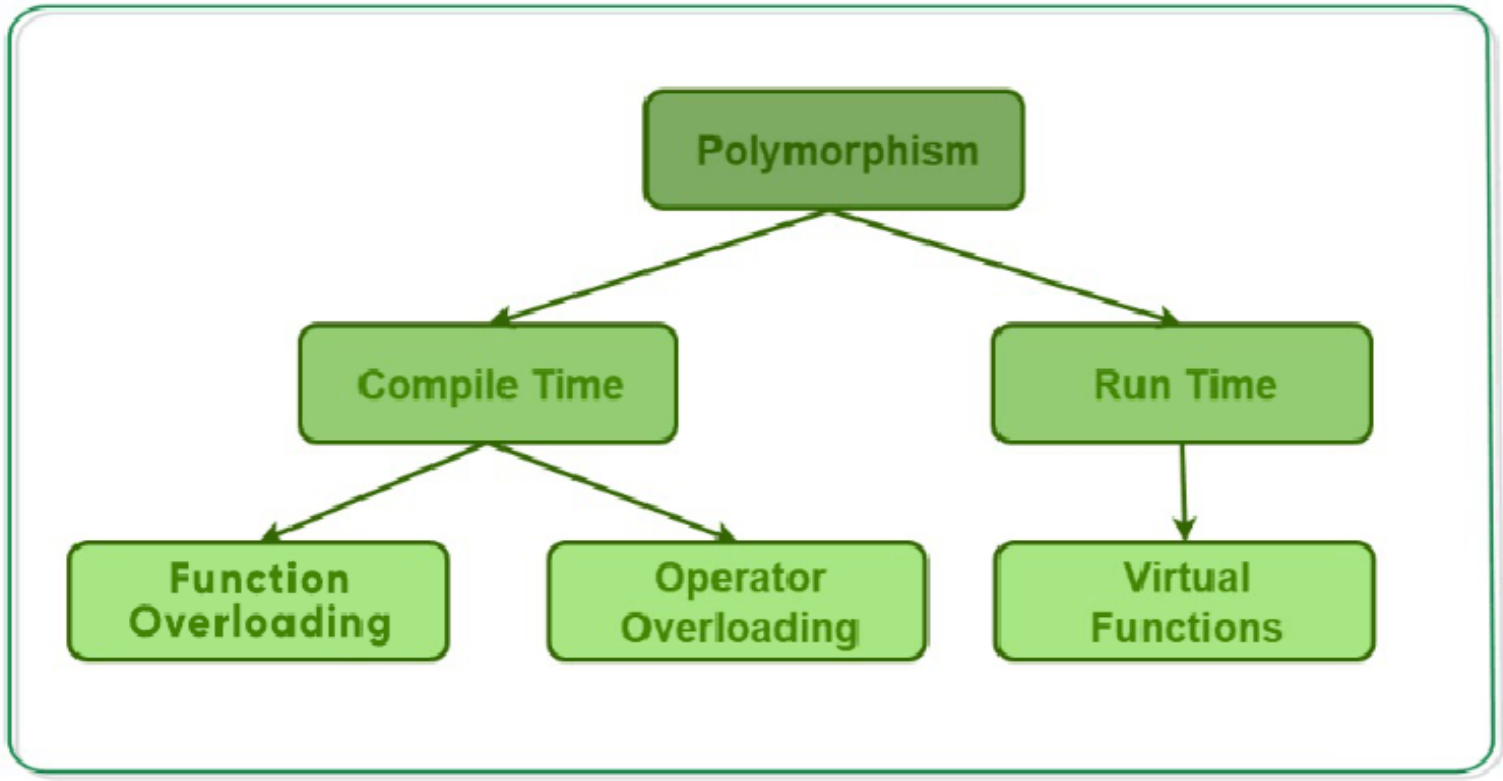


C++ Polymorphism

The word “polymorphism” means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. A real-life example of polymorphism is a person who at the same time can have different characteristics. A man at the same time is a father, a husband, and an employee. So the same person exhibits different behavior in different situations. This is called polymorphism. Polymorphism is considered one of the important features of Object-Oriented Programming.

Types of Polymorphism

- Compile-time Polymorphism
- Runtime Polymorphism



1. Compile-Time Polymorphism

This type of polymorphism is achieved by function overloading or operator overloading.

A. Function Overloading

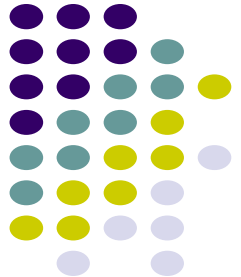
When there are multiple functions with the same name but different parameters, then the functions are said to be **overloaded,** hence this is known as Function Overloading. Functions can be overloaded by **changing the number of arguments** or/and **changing the type of arguments**. In simple terms, it is a feature of object-oriented programming providing many functions that have the same name but distinct parameters when numerous tasks are listed under one function name. There are certain Rules of Function Overloading that should be followed while overloading a function.

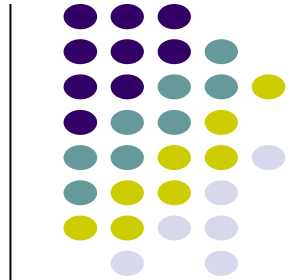
Below is the C++ program to show function overloading or compile-time polymorphism:

```
// C++ program to demonstrate function overloading or Compile-time Polymorphism
class Test {
public:
        // Function with 1 int parameter
        void func(int x)
        {
                cout << "value of x is " << x << endl;
        }

        // Function with same name but
        // 1 double parameter
        void func(double x)
        {
                cout << "value of x is " << x << endl;
        }

        // Function with same name and
        // 2 int parameters
        void func(int x, int y)
        {
                cout << "value of x and y is " << x << ", " << y << endl;
        }
};
```





```
// Driver code
int main()
{
        Test obj1;

        // Function being called depends
        // on the parameters passed
        // func() is called with int value
        obj1.func(7);

        // func() is called with double value
        obj1.func(9.132);

        // func() is called with 2 int values
        obj1.func(85, 64);
        return 0;
}
```