Lecture review

Polymorphims means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

Real world example

- A real – life example of polymorphism is that a person at the same time can have different characteristics. A man at the same time is a father, a husband, an employee, so the same person possesses different behavior in different situations. This is called as polymorphism.

TYPES OF POLYMORPHISM

In C++ polymorphism is mainly divided into two types:

- Compile time Polymorphism
- Runtime Polymorphism

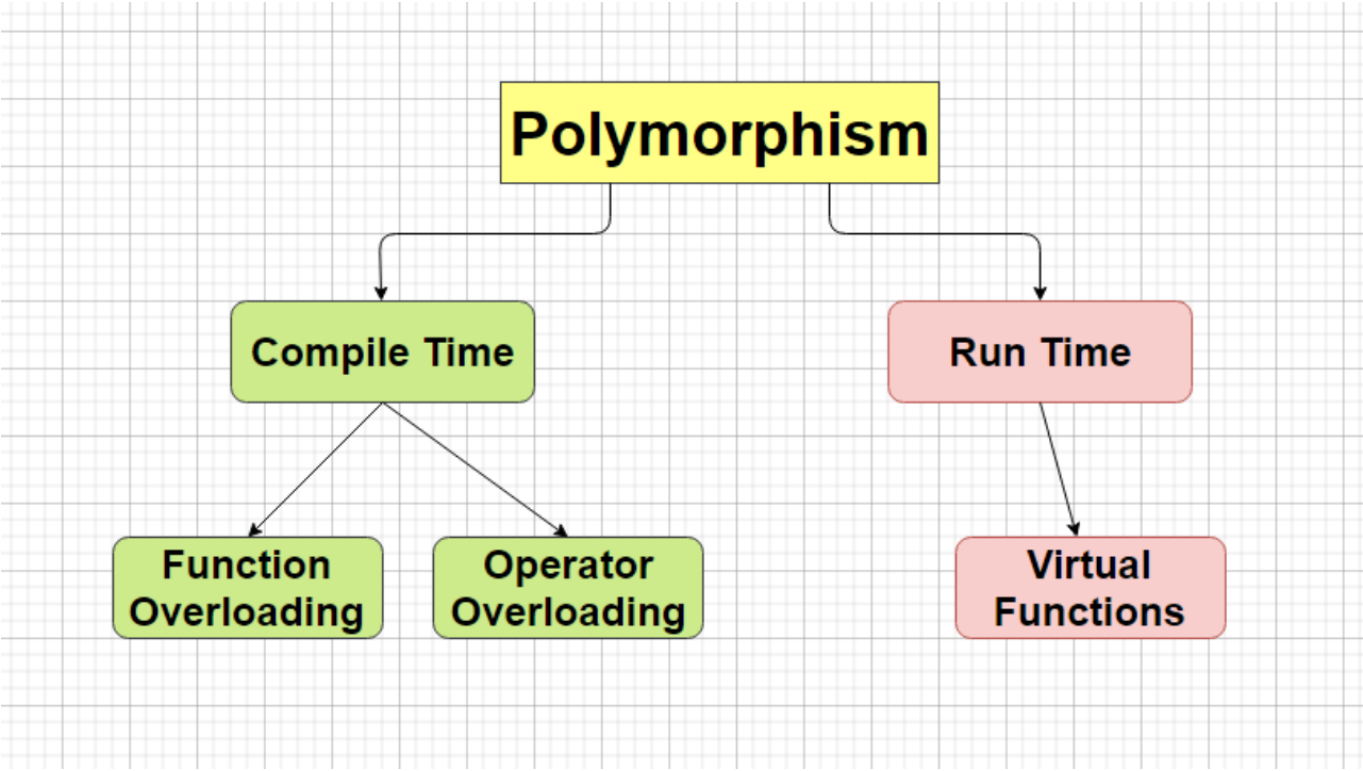

Figure 1: Types of Polymorphism

Compile time Polymorphism

This type of polymorphism is achieved by function overloading or operator overloading.
Function overloading When there are multiple functions with same name but different parameters then these functions are said to be overloaded. Functions can be overloaded by a change in the number of arguments or/and change in the type of arguments.

```
// C++ program for function overloading
#include <bits/stdc++.h>

using namespace std;
class Geeks {
    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;
    }
};

int main() {

    Geeks obj1;

    // Which function is called will depend on the parameters passed
    // The first 'func' is called
    obj1.func(7);

    // The second 'func' is called
    obj1.func(9.132);

    // The third 'func' is called
    obj1.func(85, 64);
    return 0;
}

Sample Run:
value of x is 7
value of x is 9.132
value of x and y is 85, 64

In the above example, a single
function named func acts differently in three different situations which is the property of polymorphism.
```

Operator Overloading

In C++, the purpose of operator overloading for an operator is to specify more than one meaning in one scope. It gives special meaning of an operator for a user-defined data type. You can redefine most of the C++ operators with the help of operator overloading. To perform multiple operations using one operator, you can also use operator overloading. Syntax of operator overloading:
To overload a C++ operator, you should define a special function inside the Class as follows:

```
    class class_name
{
    ... .. ...
    public
       return_type operator symbol (argument(s))
       {
           ... .. ...
       }
    ... .. ...
};
```

Here is an explanation for the above syntax:

- The return_type is the return type for the function.
- Next, you mention the operator keyword.
- The symbol denotes the operator symbol to be overloaded. For example, +, -, <, ++.
- The argument(s) can be passed to the operator function in the same way as functions.

Example:

```
#include <iostream>

using namespace std;
class TestClass {
    private: int count;
    public: TestClass() {
        count = 5;
    }
    void operator--() {
        count = count - 3;
    }
    void Display() {
        cout << "Count: " << count;
    }
};
int main() {
    TestClass tc;
    --tc;
    tc.Display();
    return 0;
```

```
}
output:
Count = 2
```

C++ Operators that cannot be Overloaded

+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	-
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []

Table 1: C++ Operators and Symbols that can not be overloaded

Unary Operators Overloading

The unary operators operate on a single operand and following are the examples of Unary operators.

- The increment (++) and decrement (-) operators.
- The unary minus (-) operator.
- The logical not (!) operator.

The unary operators operate on the object for which they were called and normally, this operator appears on the left side of the object, as in !obj, -obj, and ++obj but sometime they can be used as postfix as well like obj++ or obj-. Following example explain how minus (-) operator can be overloaded for prefix as well as postfix usage.

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

class Distance {
   private:
      int feet;             // 0 to infinite
      int inches;           // 0 to 12

   public:
      // required constructors
      Distance() {
         feet = 0;
         inches = 0;
      }
      Distance(int f, int i) {
         feet = f;
         inches = i;
      }

      // method to display distance
```

```
      void displayDistance() {
         cout << "F: " << feet << " I:" << inches <<endl;
      }

      // overloaded minus (-) operator
      Distance operator- () {
         feet = -feet;
         inches = -inches;
         return Distance(feet, inches);
      }
};

int main() {
   Distance D1(11, 10), D2(-5, 11);

   -D1;                    // apply negation
   D1.displayDistance();   // display D1

   -D2;                    // apply negation
   D2.displayDistance();   // display D2

   return 0;
}

Sample output:
F: -11 I: -10
F: 5   I: -11
```

The increment (++) and decrement (-) operators are two important unary operators available in C++. Following example explain how increment (++) operator can be overloaded for prefix as well as postfix usage. Similar way, you can overload operator (-).

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

class Time {
   private:
      int hours;             // 0 to 23
      int minutes;           // 0 to 59

   public:
      // required constructors
      Time() {
         hours = 0;
         minutes = 0;
      }
      Time(int h, int m) {
         hours = h;
         minutes = m;
      }

      // method to display time
      void displayTime() {
```

```
         cout << "H: " << hours << " M:" << minutes <<endl;
      }

      // overloaded prefix ++ operator
      Time operator++ () {
         ++minutes;          // increment this object
         if(minutes >= 60) {
            ++hours;
            minutes -= 60;
         }
         return Time(hours, minutes);
      }

      // overloaded postfix ++ operator
      Time operator++( int ) {

         // save the original value
         Time T(hours, minutes);

         // increment this object
         ++minutes;

         if(minutes >= 60) {
            ++hours;
            minutes -= 60;
         }

         // return old original value
         return T;
      }
};

int main() {
   Time T1(11, 59), T2(10,40);

   ++T1;                    // increment T1
   T1.displayTime();        // display T1
   ++T1;                    // increment T1 again
   T1.displayTime();        // display T1

   T2++;                    // increment T2
   T2.displayTime();        // display T2
   T2++;                    // increment T2 again
   T2.displayTime();        // display T2
   return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
H: 12 M:0
H: 12 M:1
H: 10 M:41
H: 10 M:42
```

Figure 2: Sample output

Binary Operators Overloading

The binary operators take two arguments and following are the examples of Binary operators. You use binary operators very frequently like addition (+) operator, subtraction (-) operator and division (/) operator. Following example explains how addition (+) operator can be overloaded. Similar way, you can overload subtraction (-) and division (/) operators.

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

class Box {
   double length;      // Length of a box
   double breadth;     // Breadth of a box
   double height;      // Height of a box

   public:

   double getVolume(void) {
      return length * breadth * height;
   }

   void setLength( double len ) {
      length = len;
   }

   void setBreadth( double bre ) {
      breadth = bre;
   }

   void setHeight( double hei ) {
      height = hei;
   }

   // Overload + operator to add two Box objects.
   Box operator+(const Box& b) {
      Box box;
      box.length = this->length + b.length;
      box.breadth = this->breadth + b.breadth;
      box.height = this->height + b.height;
      return box;
   }
```

```
};

// Main function for the program
int main() {
   Box Box1;                // Declare Box1 of type Box
   Box Box2;                // Declare Box2 of type Box
   Box Box3;                // Declare Box3 of type Box
   double volume = 0.0;     // Store the volume of a box here

   // box 1 specification
   Box1.setLength(6.0);
   Box1.setBreadth(7.0);
   Box1.setHeight(5.0);

   // box 2 specification
   Box2.setLength(12.0);
   Box2.setBreadth(13.0);
   Box2.setHeight(10.0);

   // volume of box 1
   volume = Box1.getVolume();
   cout << "Volume of Box1 : " << volume <<endl;

   // volume of box 2
   volume = Box2.getVolume();
   cout << "Volume of Box2 : " << volume <<endl;

   // Add two object as follows:
   Box3 = Box1 + Box2;

   // volume of box 3
   volume = Box3.getVolume();
   cout << "Volume of Box3 : " << volume <<endl;

   return 0;
}

/*Sample output*/
Volume of Box1-------210
Volume of Box2-------1560
Volume of Box3-------5400
```

Relational Operators Overloading

There are various relational operators supported by C++ language like (`<`, `>`, `<=`, `>=`, `==`, etc.) which can be used to compare C++ built-in data types. You can overload any of these operators, which can be used to compare the objects of a class. Following example explains how a < operator can be overloaded and similar way you can overload other relational operators.

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

```
class Distance {
   private:
      int feet;             // 0 to infinite
      int inches;           // 0 to 12

   public:
      // required constructors
      Distance() {
         feet = 0;
         inches = 0;
      }
      Distance(int f, int i) {
         feet = f;
         inches = i;
      }

      // method to display distance
      void displayDistance() {
         cout << "F: " << feet << " I:" << inches <<endl;
      }


      // overloaded < operator
      bool operator <(const Distance& d) {
         if(feet < d.feet) {
            return true;
         }
         if(feet == d.feet && inches < d.inches) {
            return true;
         }

         return false;
      }
};

int main() {
   Distance D1(11, 10), D2(5, 11);

   if( D1 < D2 ) {
      cout << "D1 is less than D2 " << endl;
   } else {
      cout << "D2 is less than D1 " << endl;
   }

   return 0;
}
```

Assignment Operator Overloading

You can overload the assignment operator (=) just as you can other operators and it can be used to create an object just like the copy constructor. Following example explains how an assignment operator can be

overloaded.

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

class Distance {
   private:
      int feet;             // 0 to infinite
      int inches;           // 0 to 12

   public:
      // required constructors
      Distance() {
         feet = 0;
         inches = 0;
      }
      Distance(int f, int i) {
         feet = f;
         inches = i;
      }
      void operator = (const Distance &D ) {
         feet = D.feet;
         inches = D.inches;
      }

      // method to display distance
      void displayDistance() {
         cout << "F: " << feet <<  " I:" <<  inches << endl;
      }
};

int main() {
   Distance D1(11, 10), D2(5, 11);

   cout << "First Distance : ";
   D1.displayDistance();
   cout << "Second Distance :";
   D2.displayDistance();

   // use assignment operator
   D1 = D2;
   cout << "First Distance :";
   D1.displayDistance();

   return 0;
}
```

Function Call() Operator Overloading

The function call operator `()` can be overloaded for objects of class type. When you overload `()` , you are not creating a new way to call a function. Rather, you are creating an operator function that can be passed an arbitrary number of parameters. Following example explains how a function call operator `()` can be

overloaded.

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

class Distance {
   private:
      int feet;             // 0 to infinite
      int inches;           // 0 to 12

   public:
      // required constructors
      Distance() {
         feet = 0;
         inches = 0;
      }
      Distance(int f, int i) {
         feet = f;
         inches = i;
      }

      // overload function call
      Distance operator()(int a, int b, int c) {
         Distance D;

         // just put random calculation
         D.feet = a + c + 10;
         D.inches = b + c + 100 ;
         return D;
      }

      // method to display distance
      void displayDistance() {
         cout << "F: " << feet << " I:" << inches << endl;
      }
};

int main() {
   Distance D1(11, 10), D2;

   cout << "First Distance : ";
   D1.displayDistance();

   D2 = D1(10, 10, 10); // invoke operator()
   cout << "Second Distance :";
   D2.displayDistance();

   return 0;
}
Sample output
First  Distance F11 I:10
Second Distance F30 I:120
```

Things to remember for Operator Overloading

Here are rules for Operator Overloading:

- At least one of the operands in overloaded operators must be user-defined, which means we cannot overload the minus operator to work with one integer and one double. However, you could overload the minus operator to work with an integer and a mystring.
- We can only overload the operators that exist and cannot create new operators or rename existing operators.
- You can make the operator overloading function a friend function if it needs to access the private and protected class members.
- Some operators cannot be overloaded using a friend function. However, such operators can be overloaded using member function. Eg. Assignment overloading operator = and function call() operator etc.

Run time Polymorphism

This type of polymorphism is achieved by Function Overriding.

Function Overriding

Function overriding is a feature that allows us to have a same function in child class which is already present in the parent class.

- A child class inherits the data members and member functions of parent class, but when you want to override a functionality in the child class then you can use function overriding. It is like creating a new version of an old function, in the child class.
- To override a function you must have the same signature in the child class.

Syntax for Function Overriding:

```
public class Parent{
access_modifier:
return_type method_name(){}
};

public class child : public Parent {
access_modifier:
return_type method_name(){}
};
```

Example Code for Function Overriding:

```
#include <iostream>
using namespace std;
class BaseClass {
public:
   void disp(){
```

```
      cout<<"Function of Parent Class";
   }
};
class DerivedClass: public BaseClass{
public:
   void disp() {
      cout<<"Function of Child Class";
   }
};
int main() {
   DerivedClass obj = DerivedClass();
   obj.disp();
   return 0;
}


Sample Run:
Function of Child Class
```

Lab exercises

Exercise 1

Write a program to overload decrement operator - in such a way that when it is used as a prefix, it multiplies a number by 4 and when it is used as a postfix then it divides the number by 4.

Exercise 2

Write a program that will apply the concept of operator overloading on + operator to add the areas of $shape_1$ and $shape_2$. Name of class is "shape" while $shape_1$ and $shape_2$ are the objects of class shape. Use the same Area() function for both objects.

Exercise 3

You are assigned a task in IBA sports day which is to find out the area of football ground, cricket ground, and a place for robot playing. The mentioned places are square, rectangle and circular in shape. You are given only one side length for football ground which can be user input by using constructor only and is private. Then there is a constraint that the width of cricket ground is equal to football side but the length is twice, while the robotic play ground radius is equal to the given football side. There is another constraint that you can only use the football given side for finding out the area of cricket ground, and robotic ground along with football ground.

Exercise 4

A school teacher is going to teach her students the different sounds of certain animals. You need to implement a program that does the following as shown in the figure:

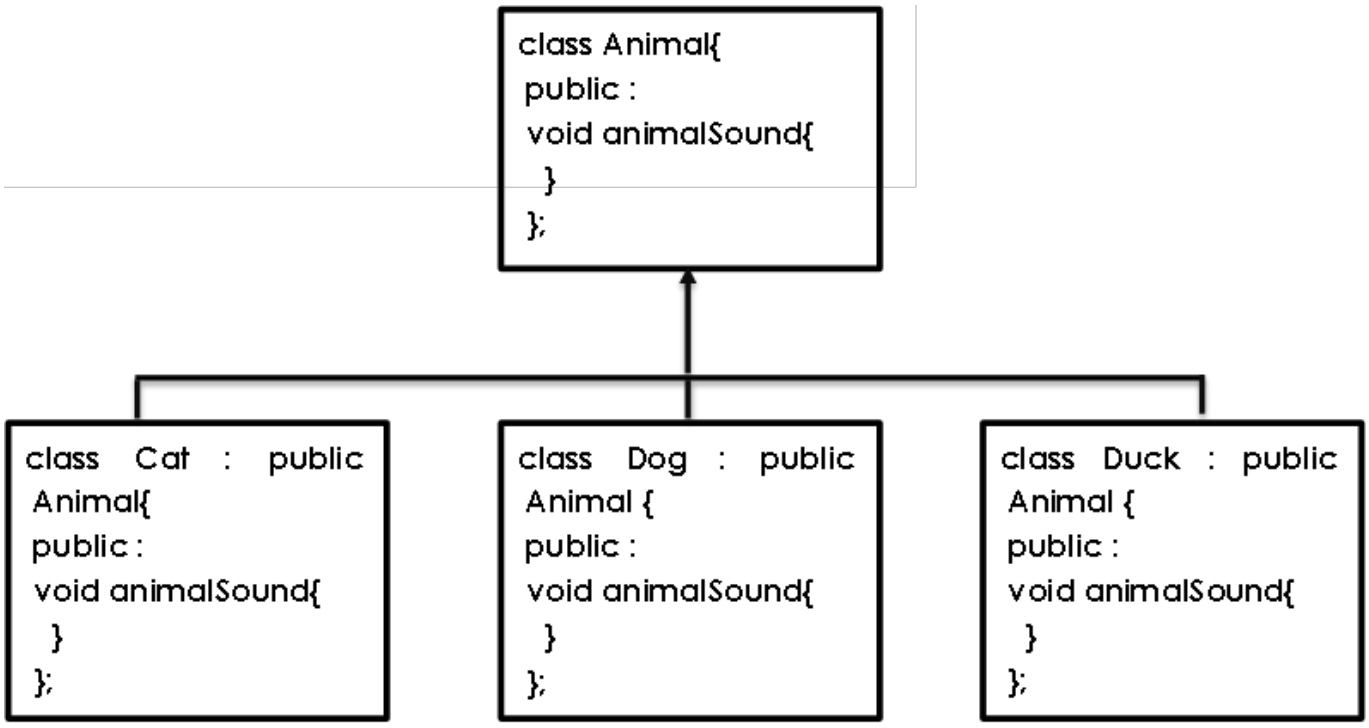

Figure 3: Class diagram

Note the following:

- The animalSound function in the Animal class displays the text as follows: "The animal makes a sound".
- The animalSound function in the Cat class displays the text as follows: "The cat says meow".
- The animalSound function in the Dog class displays the text as follows: "The dog says bow wow".
- The animalSound function in the Duck class displays the text as follows: "The duck says quack quack".
- In the main function create objects for each of the classes and display the text for each of them.

Exercise 5

You are required to create a program that allows students to calculate the volume of a cube, cylinder and a rectangular box. You will need to create three separate functions each having the same name that performs the calculations. You then need to display all the results of the calculations performed.
Cube = $(\text{side})^3$
Cylinder = $3.14 * \text{radius} * \text{radius} * \text{height}$
Rectangular box = $\text{Length} * \text{Breadth} * \text{Height}$

Exercise 6

A company wants to calculate the bonuses of each of the employees that work in a particular department. Your services are required as a programmer to develop an automated program that will allow the company to perform their calculations. You need to create a base class called **Person** and derive two other classes named as **Admin** and **Accounts**.

- The base class will have the member functions getData, displayData and the derived class will have the member functions getData, displayData and bonus.
- The Person class will contain a data member that will store the Employee's ID.
- The Admin and Accounts contain data members that include the name of the employee and their monthly income. The bonus function will calculate the bonus of the employees. According to the company's policy each employee is awarded an annual bonus of 5
- Display each employee's information that includes the Employee's ID, their name, their monthly income and the bonus each one received.

Exercise 7

You are required to develop a program that allows students at a school to carry out multiplication operations with ease. To do that you need to create four different functions having the same name of your choice.

- The first function will take two integer values and return the result after multiplying them.
- The second function will take three integer values and return the result after multiplying them.
- The third function will take two decimal values and return the result after multiplying them.
- The fourth function will take three decimal values and return the result after multiplying them.
- Display the results for all the four functions to the user.

****