Date No.

Polymorphism in C++

Polymorphism in C++ is very important feature in OOP which is used to define a message or function in many forms.

(*) Types:

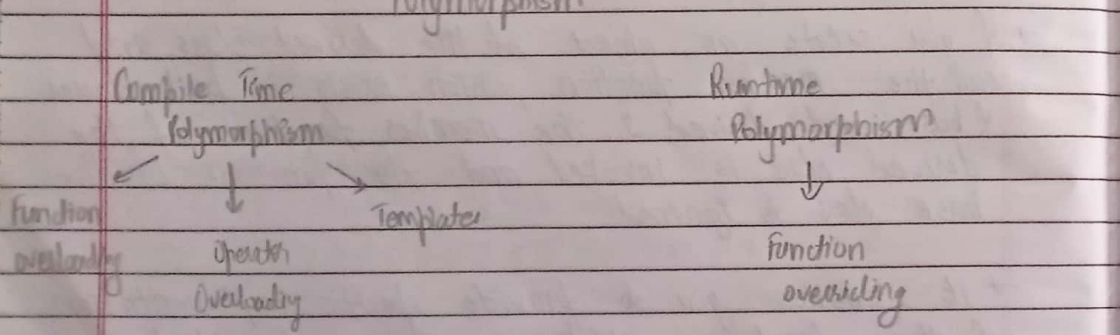
In C++, there are two types of polymorphism.

(i): Compile time Polymorphism | Static Polymorphism | Early Binding

An object is bound to its function call at compile time.

(ii) Run time | Dynamic Polymorphism | Late binding

The member function is selected during runtime and is possible using virtual function.



Date No.

(*) Pointers to Objects:

Syntax: class name * pointername;

- Used to create objects at runtime.

Eg: item * ptr; ptr = &x;

We can refer to their members functions in two ways.

Syntax: i) Object-name . function-name (arguments)

Eg: x . show();

ii) pointer-name -> functionname (arguments)

Eg. ptr -> show();

(iii) (* pointer-name) . function_name().

Eg. (*ptr) . show();

We can create new objects using pointers and 'new' operator.

Eg: i) item *ptr = new item; { allocate enough memory for data members in object structure and assigns memory address to ptr. }

ii) item *ptr = new item [10] // array of 10 objects.

(*) Pointer to Derived Class.

We can use the pointer of base class to point to objects of derived class.

During this process, the members present in the base class can only be accessed.

Virtual functions are used for this purpose.

(*) Virtual functions in C++

A virtual function is a member function which is declared within base class and is re-defined by derived class.

Thus, while using to refer to derived class object using a pointer or a reference to base class, you can call virtual function for that object and execute the derived class version of the function.

→ Importance:

i) Ensures correct function is called for an object regardless of the type of reference used.
ii) Used to achieve Runtime polymorphism.
iii) Declared using 'virtual' keyword in base class.
iv) Resolving function call is done at run-time.

(*) Rules:

(i): They must be member of base class and cannot be static.

(ii) It can be accessed by object pointers and can be friend of another class.

(iii) It must be defined in base class.

(iv) No virtual constructors but can have virtual destructors.

(v) : Only base class pointer can point to derive class and not vice-versa.

(*) Abstract class:

Sometimes implementation of all function cannot be provided in base class because we don't know the implementation and such class is called abstract class.

→ Not used to create objects
→ Only acts as base class.
→ can't be instantiated but pointers can be created.
→ must contain atleast one virtual function.

(*) Pure virtual function:

A pure virtual function in C++ is a virtual function for which we don't have implementation, we only declare it.

⇒ A pure virtual function is a special kind of virtual function that doesn't have any body and acts as a placeholder meant to redefined by derived class.

A pure virtual function is declared by assigning 0 in declaration.

*) Note:

i) A class is abstract if it has atleast one pure virtual functions.
ii) If we don't override the pure virtual function in derived class, then derived class also becomes abstract class.

(*) Interface Class

An interface class is a class that has no member variables, and where all of the functions are pure virtual.

Interfaces are useful when you want to define the functionality that derived classes must implement, but leave the details of how the derived class implements that functionality entirely up to the derived class.

(*) Dynamic casting:

A base pointer can point to any of the derived object, but the reverse is not true.

To convert base-class pointers into derived class pointers, we use a casting operator named dynamic_cast.

Done when we need access to something that is derived class specific.