object-oriented programming

(OOP)

KIAN _ ACADEMY

Quick Review On Pointer

Example :

```
#include <bits/stdc++.h>
using namespace std;
class Person
{
public:
    void print()
    {
        cout << "Person\n";
    }
    ~Person()
    {
        cout << "~Person\n";
    }
};
```

```
class Student: public
Person
{
public:
    void print()
    {
        cout << "Student\n";
    }
    ~Student()
    {
        cout <<
"~Student\n";
    }
};
```

```
int main()
{
    Person person;      // first object
    Student stud;       // 2nd object
     person.print();
    stud.print();
    Person* per_ptr = &person;
    Student* stud_ptr1 = &stud;
    Person* stud_ptr2 = &stud;
    Person* stud_ptr3 = new Student();      // 3rd object
    per_ptr->print();
    stud_ptr1->print();
    stud_ptr2->print();
    stud_ptr3->print();
```

```
delete stud_ptr3;       // Memory leak
   return 0;
}
```

The output :

1-Person

2-Student

3-Person

4-Student

5-Person

6-Person

7-~Person from 3rd object: memory leak: student is not cleaned

8-~Student from 2nd object: derived

9-~Person from 2nd object: base

10-~Person from first object

Virtual Functions

Example :

```
#include <bits/stdc++.h>
using namespace std;
class Person
{
public:
    virtual void print()
    {
        cout << "Person\n";
    }
    virtual ~Person()
    {
        cout << "~Person\n";
    }
};
```

```
class Student: public Person
{
public:
    void print()
    {
        cout << "Student\n";
    }   ~Student()
    {
        cout << "~Student\n";
    }
};
```

```
int main()
{
    Person* stud_ptr3 = new Student();//first object
    stud_ptr3->print();
    stud_ptr3->Person::print();
    delete stud_ptr3;
    return 0;
}
```

The output :

```
Student
Person
~Student
~Person
```

Virtual Functions :

- A virtual function is a member function which is declared within a base class and is redefined(Overriden) by a derived class . When you refer to a derived class object using a pointer to the base class, you can call a virtual function for that object and execute the derived class's version of the function

****

Another Example on Virtual Functions :

```
#include <bits/stdc++.h>
using namespace std;
class Shape
{
private:
    string name;
public:
    Shape(string name):name(name)
    {
    }
    virtual int Area()
    {
        cout<<"Not implemented. Do override"<<endl;
    }
    string GetShapeName()
    {
        return name;
    }
    virtual ~Shape()
    {
    }
};
class Rectangle: public Shape
{
    int wid;
    int height;
public:
    Rectangle(string name, int wid, int height): Shape(name)
    {
        this->wid=wid;
        this->height=height;
    }
    int Area()
    {
        return wid * height;
    }
};
```

```
void process(Shape* shape)
{
  // This function knows nothing about children!
 // Compile time determined
 cout << "This shape's name is: " << shape->GetShapeName()<<endl;
 // Run Time determined
    cout << "Its area: " << shape->Area() << "\n";
}
```

```
int main()
{
    Rectangle r("Nice Rect", 4, 5);
    process(&r);
   return 0;
}
```

the output is :

This shape's name is: Nice Rect

Its area: 20

Another Example on Virtual Functions :

```
#include <bits/stdc++.h>
using namespace std;
class A
{
public:
    virtual void f1()
    {
        cout << "A::f1\n";    // virtual
    }
    void f2()
    {
        cout << "A::f2\n";
    }
    void f3()
    {
        cout << "A::f3\n";
    }
};
class B: public A
{
public:
    void f1()
    {
        cout << "B::f1\n";    // virtual
    }
    virtual void f2()
    {
        cout << "B::f2\n";          // virtual
    }
    void f3()
    {
        cout << "B::f3\n";
    }
};
```

```
class C: public B
{
public:
    void f1()
    {
        cout << "C::f1\n";     // virtual
    }
    void f2()
    {
        cout << "C::f2\n";     // virtual
    }
    virtual void f3()
    {
        cout << "C::f3\n";
    }
};
```

```
class D: public C
{
public:
    void f1()
    {
        cout << "D::f1\n";     // virtual
    }
    void f2()
    {
        cout << "D::f2\n";     // virtual
    }
    void f3()
    {
        cout << "D::f3\n";     // virtual
    }
};
```

```
int main()
{
    B* d1 = new D();
    d1->f1();
    d1->f2();
    d1->f3();
    cout<<"\n";
    A* d2 = new D();
    d2->f1();
    d2->f2();
    d2->f3();
    return 0;
}
```

The output is :

D::f1

D::f2

B::f3

D::f1

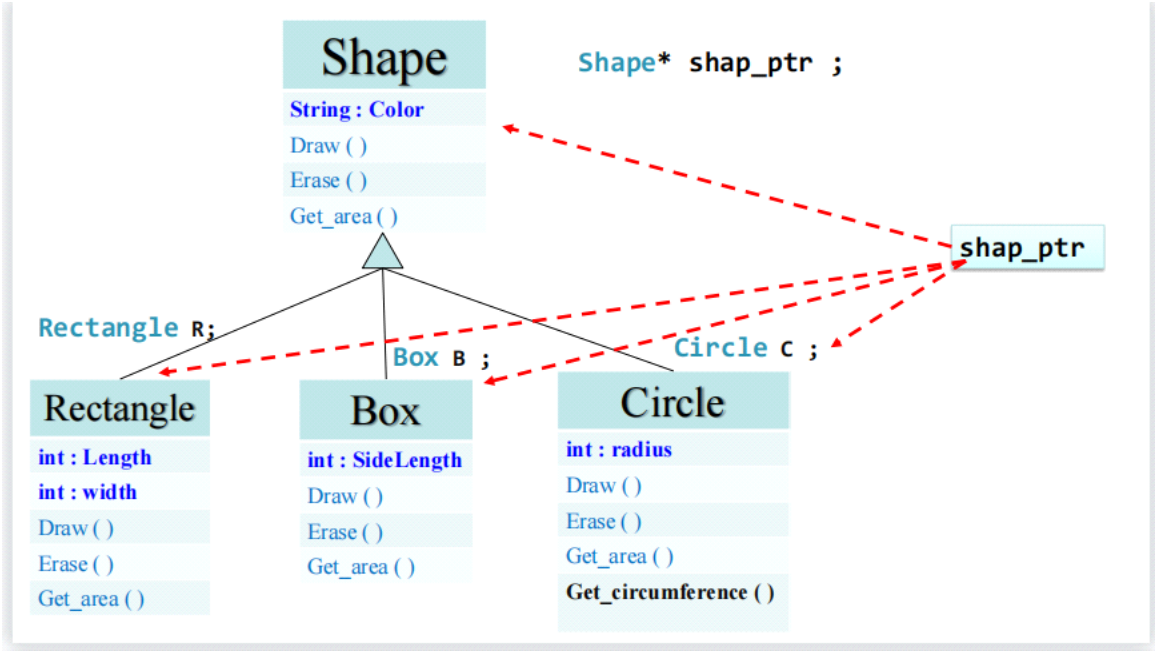
A::f2

A::f3

from understand the above :

What is Polymorphism ?

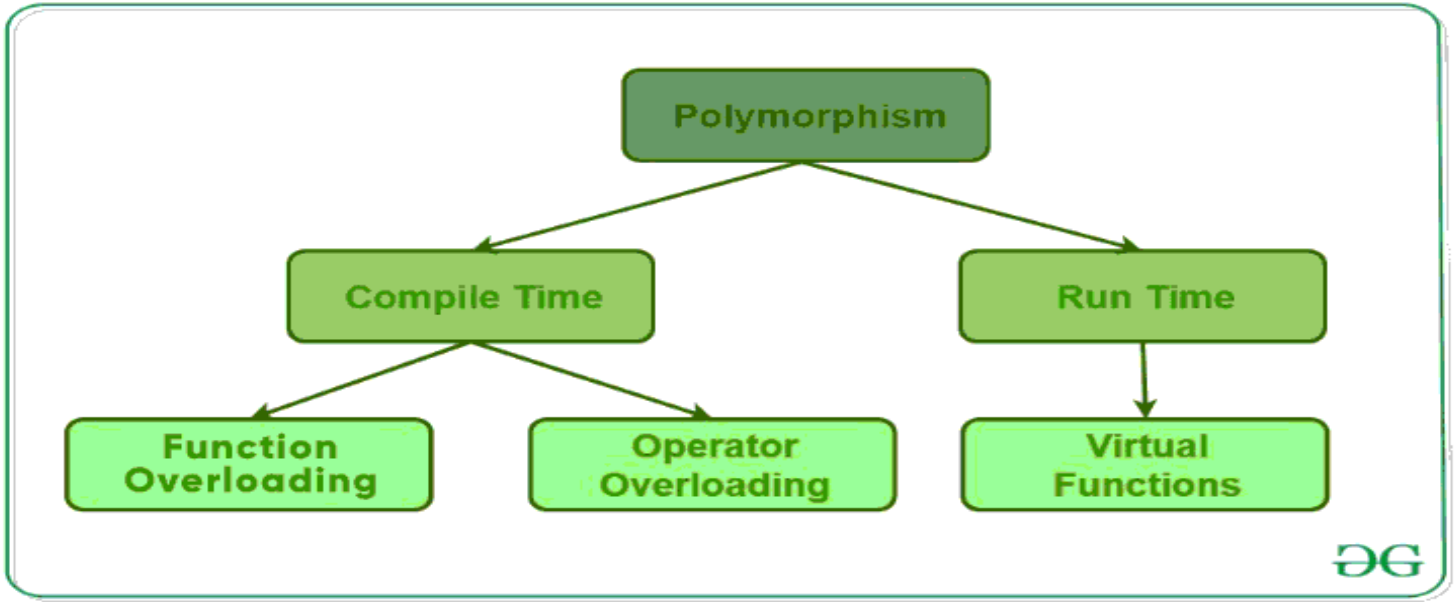
Polymorphism : the ability of a variable, function or object to take on multiple forms.



Types of Polymorphism :

Note:

- compile time is Static polymorphism
- Run time is Dynamic polymorphism



Exam questions:

- **(1)** Which of the following statements about virtual base classes is correct ?

- A) It is used to provide multiple inheritance.
- B) It is used to avoid multiple copies of base class in derived class.
- C) It is used to allow multiple copies of base class in a derived class.
- D) It allows private members of the base class to be inherited in the derived class.

- **(2)** Which of the following is true about virtual functions in C++.

- A) Virtual functions are functions that can be overridden in derived class with the same signature.
- B) Virtual functions enable run-time polymorphism in a inheritance hierarchy.
- C) If a function is 'virtual' in the base class, the most-derived class's implementation of the function is called according to the actual type of the object referred to, regardless of the declared type of the pointer or reference. In non-virtual functions, the functions are called according to the type of reference or pointer.
- D) All of the above

3)

```
#include<iostream>
using namespace std;
class Base
{
public:
    virtual void show()
    {
        cout<<" In Base \n";
    }
};
class Derived: public Base
{
public:
    void show()
    {
        cout<<"In Derived \n";
    }
};
int main()
{
    Base *bp = new Derived;
    bp->show();
    Base &br = *bp;
    br.show();
    return 0;
}
```

- From The Last Question what is the output :

- A) In Base , In Base
- B) In Base , In Derived
- C) In Derived , In Derived
- D) In Derived , In Base

- **(4)** Which of the following are the types of polymorphism?

  A-User define polymorphism

  B-System define polymorphism

  C-Static polymorphism

  D-Dynamic polymorphism

- Options:
- A) A and B
- B) C and D
- C) A, C, and D
- D) A, B, C, and D

Answer the questions :-

- 1 - b
- 2 - d
- 3 - c
- 4 - b