OOP: Polymorphism in C++

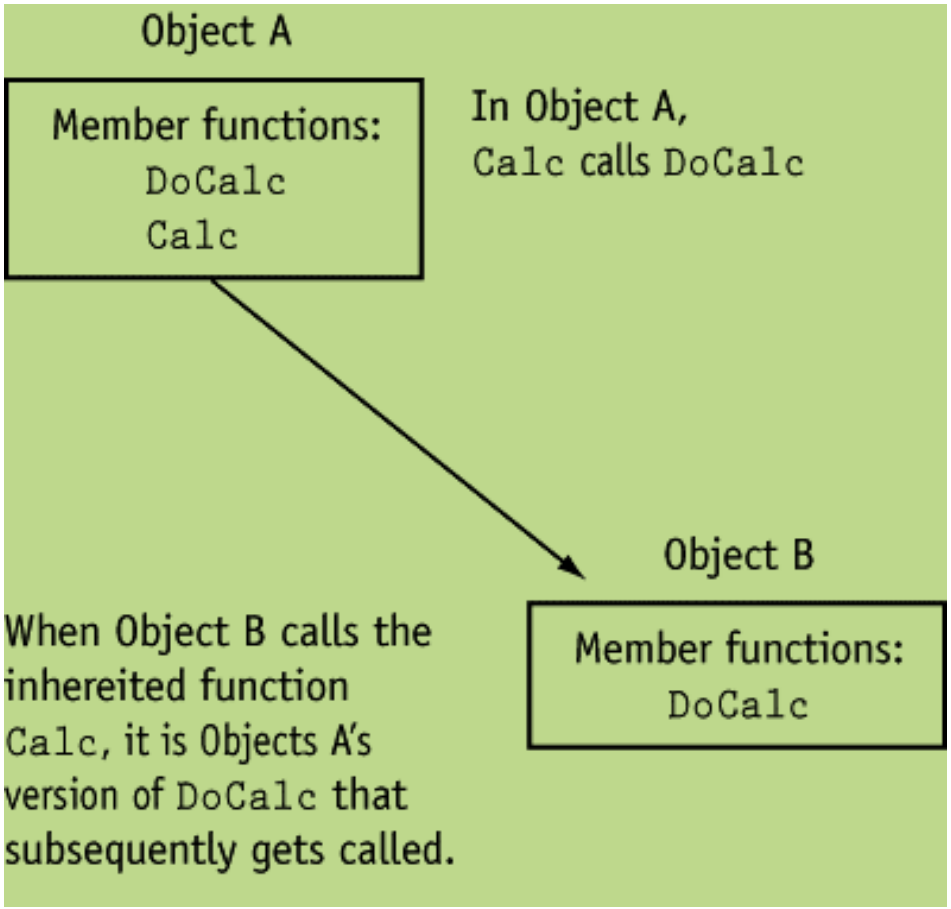
Part 2: Last

Contents

- Different Ways of Polymorphisms
- Examples and Exercise
- Virtual Destructors
- Introduction to Multiple Inheritance

Polymorphism and Virtual Member Functions

- A virtual member function in a base class expects to be overridden in a derived class



Polymorphism: Base Class Pointers

```
#include <iostream>

using namespace std;


class A
{
protected:
    int testValue;

public:
    virtual void testFunction(int a)=0;
};


class B: public A
{
public:

    void testFunction(int a)
    {
      testValue=a;
      cout << "Test Value from class B: " << testValue << endl;
    };
};
```

```
class C: public A
{
public:

    void testFunction(int a)
    {
      testValue=a;
      cout << "Test Value from class C: " << testValue<< endl
    };
};


int main()
{
    B objb;
    C objc;

    A *ptr1=&objb;
    A *ptr2=&objc;

    ptr1->testFunction(33);
    ptr2->testFunction(99);
}
```

Polymorphism through function overlaoding

```
#include<string>
#include<iostream>

using namespace std;

class Player {
    string mName;
    int mAge;
    string mGameType;

public:
    void setData(string str)
    {
        mName = str;
        cout << "Name :" << mName << endl;
    }

    void setData(string str, int age)
    {
        mAge = age;
        mName = str;
        cout << "Name :" << mName << " " << "Age :" << mAge << endl;
    }

    void setData(string str, int age, string game)
    {
        mAge = age;
        mName = str;
        mGameType = game;
        cout << "Name :" << mName  << " " << "Age :" << mAge << " " << "Game Type:" << mGameType << endl;
    }
};
```

```
int main() {

    Player p1;

    p1.setData("Sakib Hasan");
    p1.setData("John",25);
    p1.setData("Hyeryon Yoon",38,"Golf");

    return 0;
}
```

```cpp
//Polymorphism through Virtual Function
// BY: S.M. Riazul Islam

#include<iostream>
#include<string>

using namespace std;

class Player
{
public:
    virtual void showData()
    {
        cout << "Player class Data" << endl;
    }
};

class Footballer : public Player {
public :
    void showData()
    {
        cout << "Footballer class Data" << endl;
    }
};

class Cricketer : public Player {
public :
    void showData()
    {
        cout << "Cricketer class Data" << endl;
    }
};
```

```cpp
int main()
{
    Footballer f1;
    Cricketer c1;

    Player *ptr1, p1;

    ptr1=&p1;
    ptr1->showData();

    ptr1=&f1;
    ptr1->showData();

    ptr1=&c1;
    ptr1->showData();  //run time resolution
    //dynamic or late binding


    //c1.showData(); (Compile time resolution)
    //not polymorphic behavior
    // called static binding


    return 0;
}
```

Polymorphism Exercise 1

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

class First
{
protected:
    int a;
public:
    First(int x = 1)
        { a = x; }

    int getVal()
        { return a; }
};

class Second : public First
{
private:
    int b;
public:
    Second(int y = 5)
        { b = y; }
    int getVal()
        { return b; }
};
```

Find the O/P

```
int main()
{
    First object1;
    Second object2;

    cout << object1.getVal() << endl;
    cout << object2.getVal() << endl;
    return 0;
}
```

Polymorphism Exercise 2

Find the O/P

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

class First
{
protected:
   int a;
public:
   First(int x = 1)
      { a = x; }

   void twist()
      { a *= 2; }
   int getVal()
      { twist(); return a; }
};

class Second : public First
{
private:
   int b;
public:
   Second(int y = 5)
      { b = y; }

   void twist()
      { b *= 10; }
};
```

```
int main()
{
   First object1;
   Second object2;

   cout << object1.getVal() << endl;
   cout << object2.getVal() << endl;
   return 0;
}
```

Polymorphism Exercise 3

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

class First
{
protected:
    int a;
public:
    First(int x = 1)
        { a = x; }

    virtual void twist()
        { a *= 2; }

    int getVal()
        { twist(); return a; }
};

class Second : public First
{
private:
    int b;
public:
    Second(int y = 5)
        { b = y; }
    virtual void twist()
        { b *= 10; }
};
```

Find the O/P

```
int main()
{
    First object1;
    Second object2;

    cout << object1.getVal() << endl;
    cout << object2.getVal() << endl;
    return 0;
}
```

Polymorphism Exercise 4

Find the O/P

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

class Base
{
protected:
    int baseVar;
public:
    Base(int val = 2)
        { baseVar = val; }

    int getVar()
        { return baseVar; }
};

class Derived : public Base
{
private:
    int derivedVar;
public:
    Derived(int val = 100)
        { derivedVar = val; }
    int getVar()
        { return derivedVar; }
};
```

```
int main()
{
    Base *optr;
    Derived object;

    optr = &object;
    cout << optr->getVar() << endl;
    return 0;
}
```

Virtual Destructors

```
class Base
{
    // some virtual methods
};

class Derived : public Base
{
    ~Derived()
    {
        // Do some important cleanup
    }
}
```

```
Base *b = new Derived();
// use b
delete b; // Here's the problem!
```

Undefined Behavior

```cpp
#include <iostream>

using namespace std;

class Base
{
public:

    Base()
    {
        cout << "Base()" << endl;
    }

    ~Base()
    {
        cout << "~Base()" << endl;
    }


    // see 5 (after see 4)
    //virtual ~Base()
    //{
    //    cout << "~Base()" << endl;
    //}
};
```

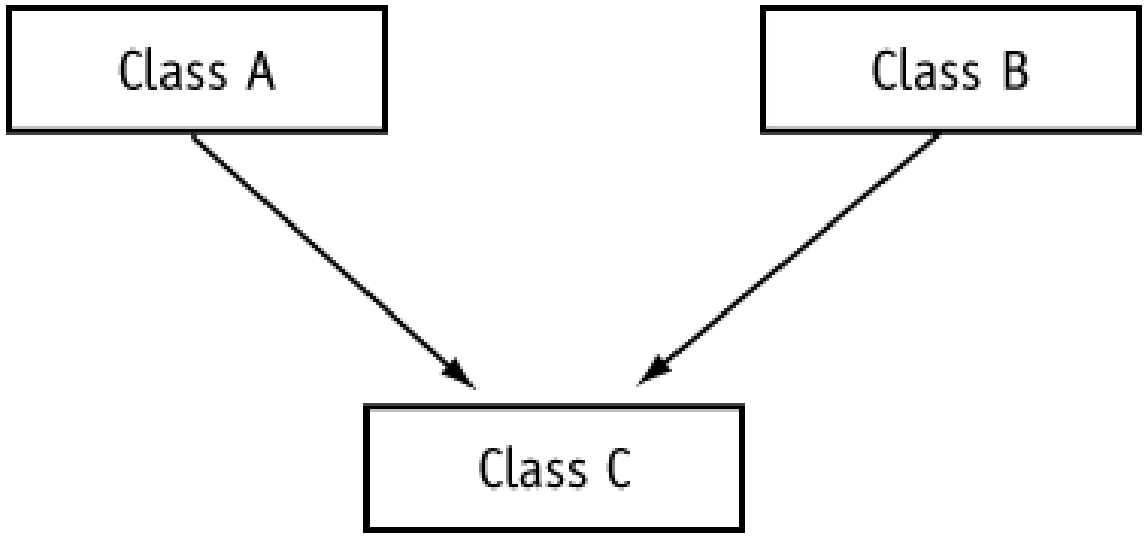
```cpp
class Derived: public Base
{
public:

    Derived()
    {
        cout << "Derived()" << endl;
    }

    ~Derived()
    {
        cout << "~Derived()" << endl;
    }
};
```

```cpp
int main()
{
    // see 1
    Derived derived;


    //see 2
    //Derived *derived=new Derived;


    //see 3
    //Derived *derived=new Derived;
    //delete derived;



    //see 4
    //Base *base=new Derived;
    //delete base;
}
```

Virtual Destructors

Multiple Inheritance

- Multiple inheritance: when a derived class has two or more base classes



class Date

```
//date.h
#ifndef DATE_H
#define DATE_H
class Date
{
   protected:
        int day;
        int month;
        int year;
   public:
        Date(int d, int m, int Y)
          { day = d; month = m; year = Y; }
        int getDay(void) { return day; }
        int getMonth(void) { return month; }
        int getYear(void) { return year; }
};
#endif
```

class Time

//time.h

```
#ifndef TIME_H
#define TIME_H
class Time
{
   protected:
        int hour;
        int min;
        int sec;
   public:
        Time(int h, int m, int s)
            { hour = h; min = m; sec = s; }
        int getHour(void) { return hour; }
        int getMin(void) { return min; }
        int getSec(void) { return sec; }
};
#endif
```

class DateTime

```
// datetime.h
#ifndef DATETIME_H
#define DATETIME_H
#include "date.h" // For Date class declaration
#include "time.h" // For Time class declaration

class DateTime : public Date, public Time
{
   protected:
        char dTString[20];
public:
   DateTime(int, int, int, int, int, int);
   void getDateTime(char *str) { strcpy(str, dTString); }
};
#endif
```

class DateTime

```
// datetime.cpp
#include “datetime.h”
#include <string.h> // For strcpy and strcat
#include <stdlib.h> // For itoa
void DateTime::DateTime(int dy, int mon, int yr, int hr, int mt,
   int sc) :
        Date(dy, mon, yr), Time(hr, mt, sc)
{
   char temp[10]; // Temporary work area for itoa()

   // Store the date in dTString, in the form MM/DD/YY
   strcpy(dTString, itoa(getMonth(), temp, 10));
   strcat(dTString, “/”);
   strcat(dTString, itoa(getDay(), temp, 10));
   strcat(dTString, “/”);
   strcat(dTString, itoa(getYear(), temp, 10));
   strcat(dTString, “ “);
```

class DateTime

```
   // Store the time in dtString, in the form HH:MM:SS
   strcpy(dTString, itoa(getHour(), temp, 10));
   strcat(dTString, ":");
   strcat(dTString, itoa(getMin(), temp, 10));
   strcat(dTString, ":");
   strcat(dTString, itoa(getSec(), temp, 10));
}
```

Multiple Inheritance Class Testing

```
#include <iostream>
#include "DateTime.h"
using namespace std;

int main()
{
   // Define a DateTime object and use the default
   // constructor to initialize it.
   DateTime emptyDay;

   // Display the object's date and time.
   cout << emptyDay.getDateTime() << endl;

   // Define a DateTime object and initialize it
   // with the date 2/4/60 and the time 5:32:27.
   DateTime pastDay(2, 4, 60, 5, 32, 27);

   // Display the object's date and time.
   cout << pastDay.getDateTime() << endl;
   return 0;
}
```

Program Output

```
1/1/1900 0:0:0
4/2/60 5:32:27
```

Virtual Destructors w. Multiple Inheritance

```cpp
#include <iostream>

using namespace std;

class A
{
public:

    virtual ~A()
    {
        cout << "~A" << endl;
    }
};

class B
{
public:

    virtual ~B()
    {
        cout << "~B" << endl;
    }
};

class AB : public A, public B
{
public:

    virtual ~AB()
    {
        cout << "~AB" << endl;
    }
};
```

```cpp
int main()
{
    AB* ab1 = new AB();
    AB* ab2 = new AB();

    A* a = ab1;
    B* b = ab2;

    delete a;
    delete b;
}
```

Try Yourself

An automatic checkout system for a supermarket chain needs to be completed.

- Declare the virtual methods `scanner()` and `printer()` in the base class `Product`. Also define a virtual destructor.
- Write the `record()` function, which registers and lists products purchased in the store in a program loop.

  The function creates an array of 100 pointers to the base class, `Product`. The checkout assistant is prompted to state whether a prepacked or fresh food item is to be scanned next. Memory for each product scanned is allocated dynamically and referenced by the next pointer in the array. After scanning all the available items, a sequential list is displayed. The prices of all the items are added and the total is output at the end.
- Now create an application program to simulate a supermarket checkout. The checkout assistant is prompted in a loop to state whether to define a new customer. If so, the `record()` function is called; if not, the program terminates.