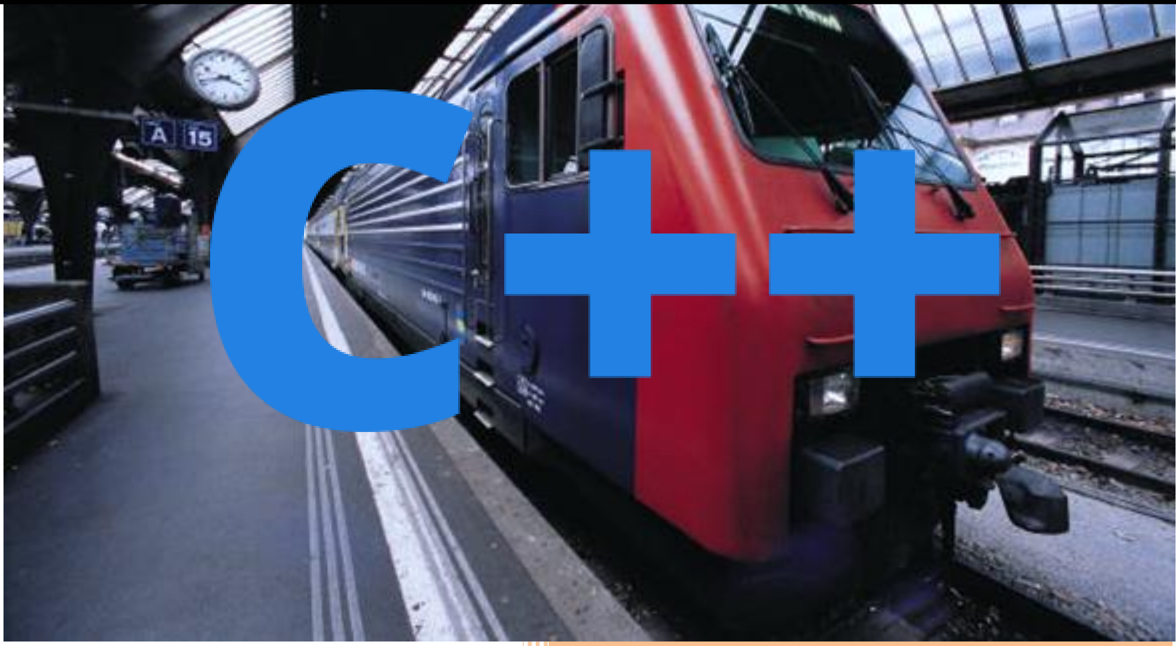
ETEC Center

Build Your IT Skill

2020

ណែនាំស្គាល់ពី Polymorphism Of OOP

Overloading Template Early/Late Binding



ណែនាំអោយស្គាល់ពី

Polymorphism

I. អ្វីទៅដែលហៅថា Polymorphism?

ពាក្យថា Polymorphism គឺមកពីភាសាក្រិចដែលហៅថា (Poly+Mophism: ទំរង់ច្រើន) ចង់សំដៅ លើចំនុច សំខាន់ពី លើ Method និង object របស់ Class។

នៅក្នុងចំនុចនេះអ្នកនិងសិក្សាលើ ៣ចំនុចសំខាន់ដូចជា៖

១) Overloading Function , Constructor & Operator

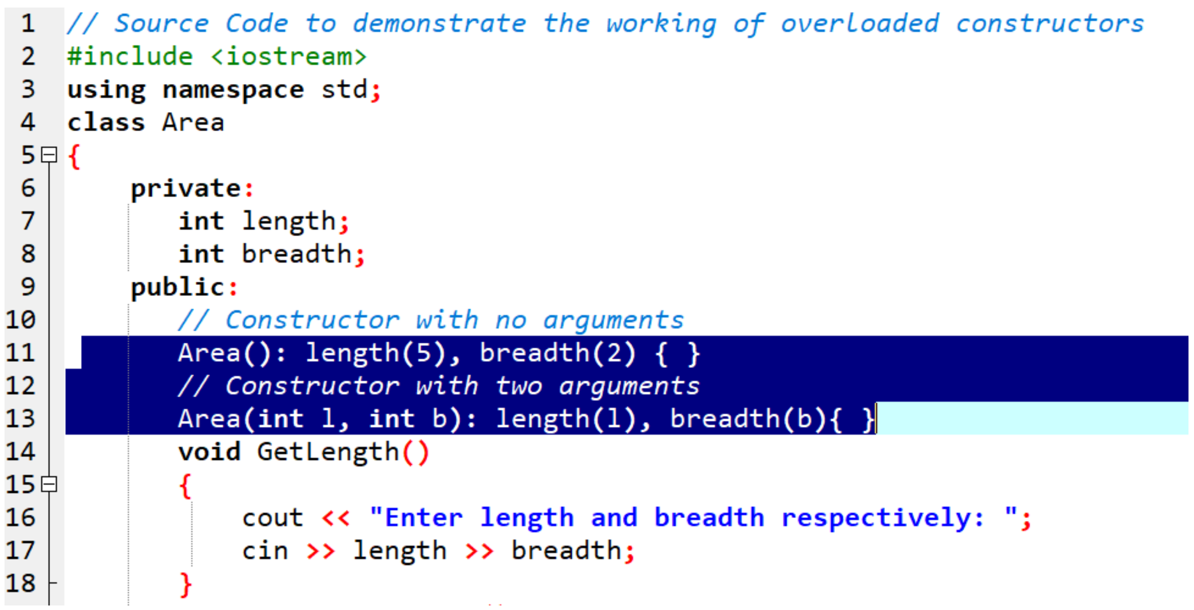
២) Overriding Methods/Abstract Class

៣) Template Function & Template Class

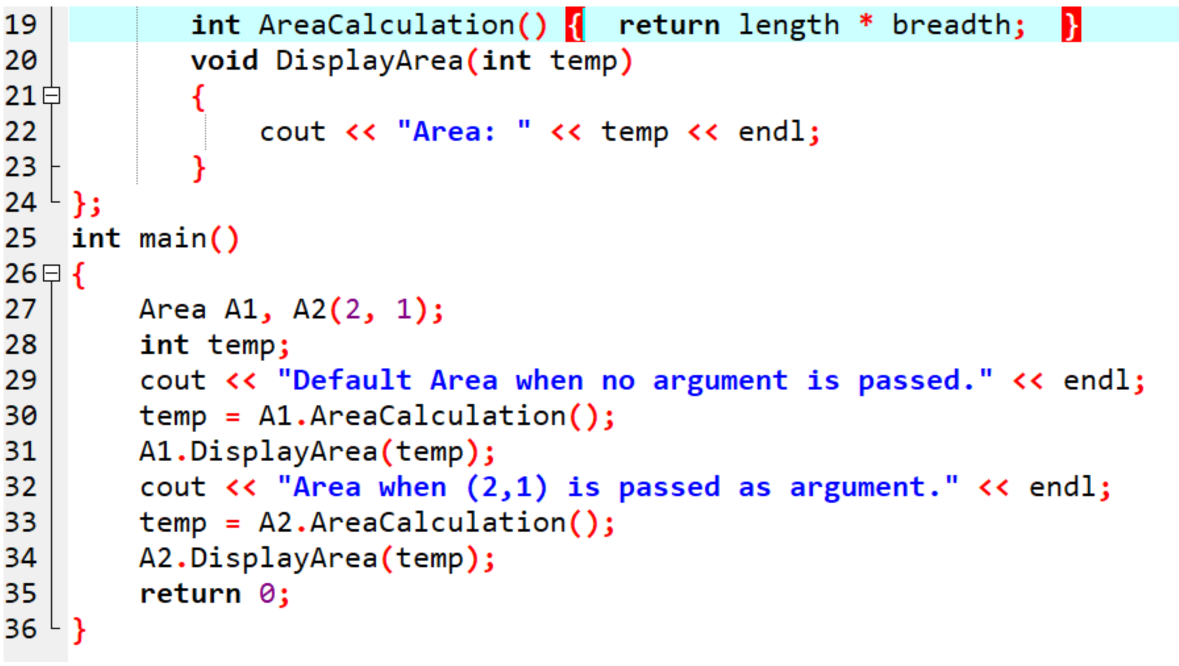
៤) Early Binding/Compile Time Binding និង Late Binding/Run Time Binding

1. Overloading Constructor: គឺជាការបង្កើតនូវ Constructor មានចាប់ពីរឡើងទៅ ដែលខុសគ្នាត្រង់ចំនួន Parameter របស់ function

ឧទាហរណ៏ ១៖

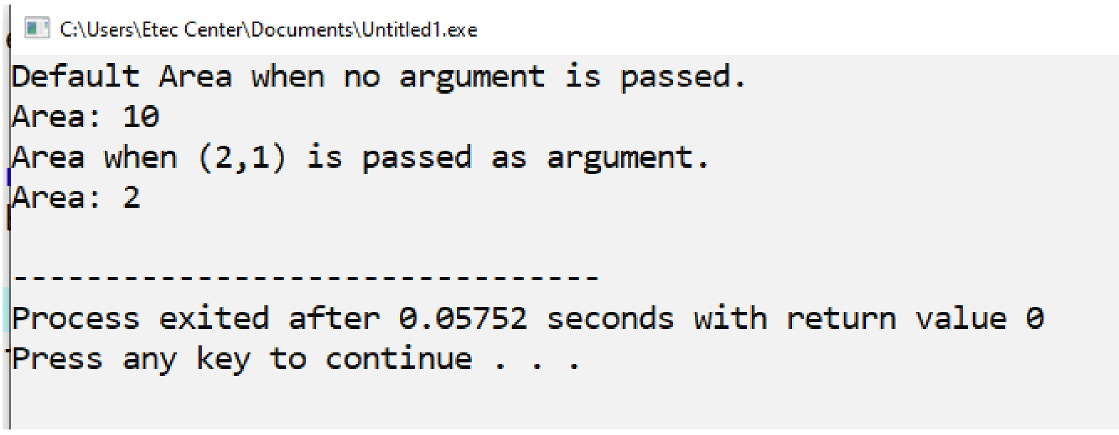


```
// Source Code to demonstrate the working of overloaded constructors
#include <iostream>
using namespace std;
class Area
{
    private:
       int length;
       int breadth;
    public:
       // Constructor with no arguments
       Area(): length(5), breadth(2) { }
       // Constructor with two arguments
       Area(int l, int b): length(l), breadth(b){ }
       void GetLength()
       {
           cout << "Enter length and breadth respectively: ";
           cin >> length >> breadth;
       }
```



```
        int AreaCalculation() {  return length * breadth;  }
        void DisplayArea(int temp)
        {
            cout << "Area: " << temp << endl;
        }
};
int main()
{
    Area A1, A2(2, 1);
    int temp;
    cout << "Default Area when no argument is passed." << endl;
    temp = A1.AreaCalculation();
    A1.DisplayArea(temp);
    cout << "Area when (2,1) is passed as argument." << endl;
    temp = A2.AreaCalculation();
    A2.DisplayArea(temp);
    return 0;
}
```

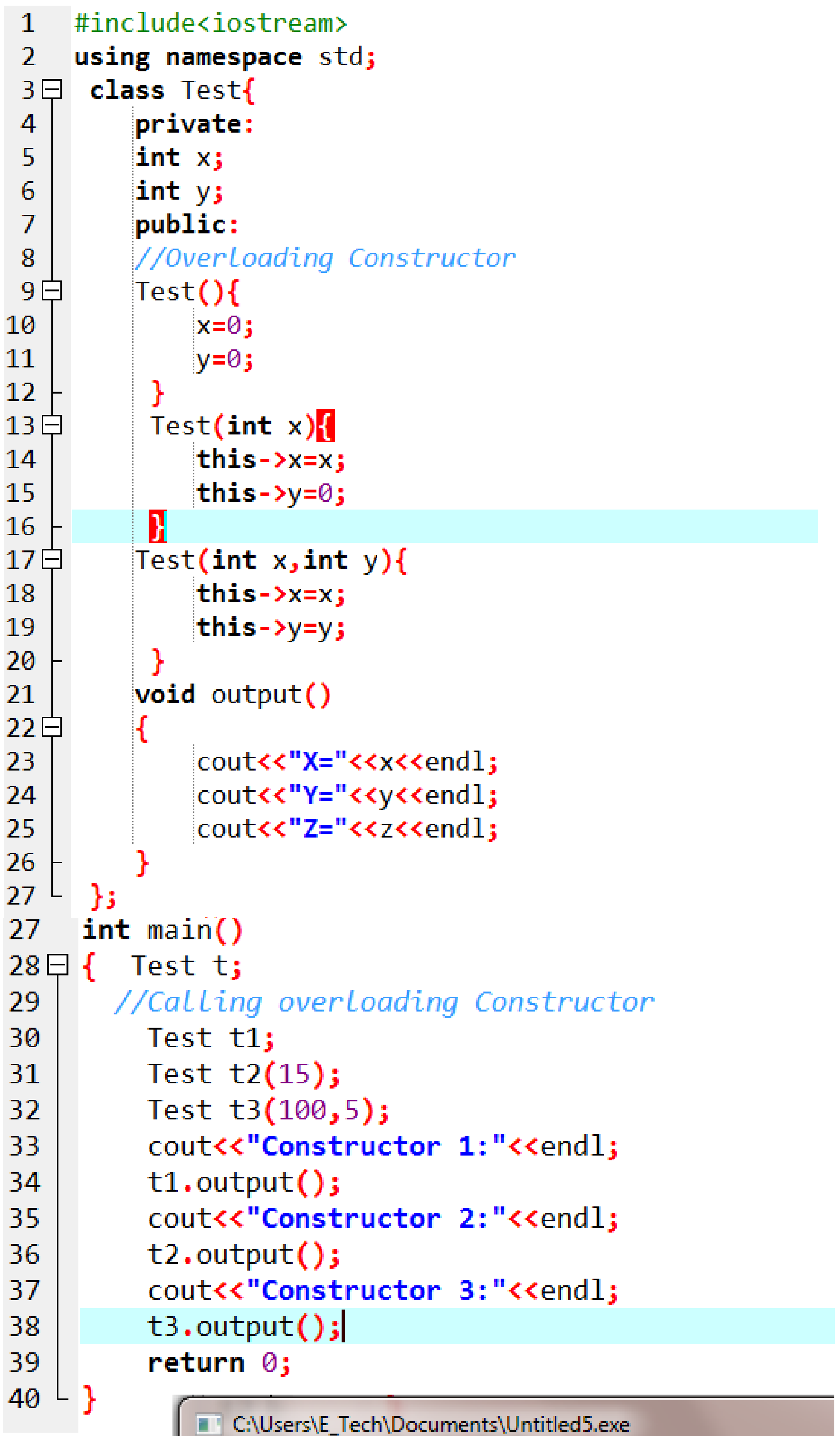
លទ្ធផលទទួលបាន៖



C:\Users\Etec Center\Documents\Untitled1.exe

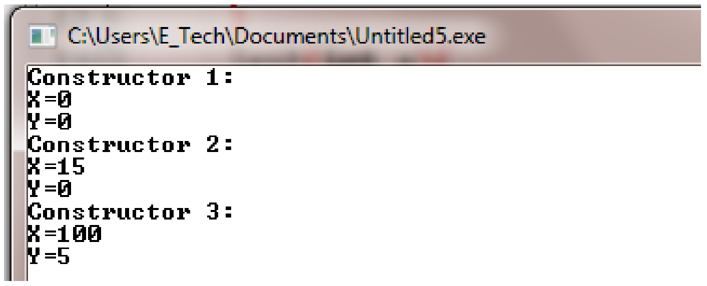
```
Default Area when no argument is passed.
Area: 10
Area when (2,1) is passed as argument.
Area: 2

--------------------------------
Process exited after 0.05752 seconds with return value 0
Press any key to continue . . .
```

ឧទាហរណ៍ ២ ៖



```
#include<iostream>
using namespace std;
 class Test{
    private:
    int x;
    int y;
    public:
    //Overloading Constructor
    Test(){
        x=0;
        y=0;
     }
     Test(int x){
        this->x=x;
        this->y=0;
     }
    Test(int x,int y){
        this->x=x;
        this->y=y;
     }
    void output()
    {
        cout<<"X="<<x<<endl;
        cout<<"Y="<<y<<endl;
        cout<<"Z="<<z<<endl;
    }
 };
int main()
{   Test t;
  //Calling overloading Constructor
    Test t1;
    Test t2(15);
    Test t3(100,5);
    cout<<"Constructor 1:"<<endl;
    t1.output();
    cout<<"Constructor 2:"<<endl;
    t2.output();
    cout<<"Constructor 3:"<<endl;
    t3.output();
    return 0;
}
```

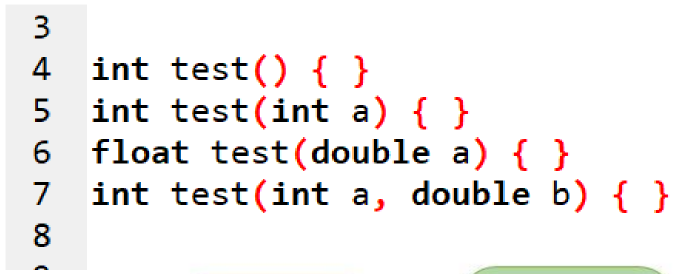
លទ្ធផលទទួលបាន៖



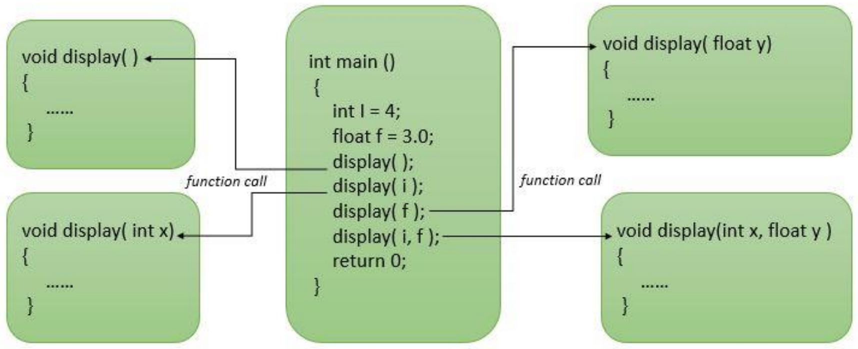
C:\Users\E_Tech\Documents\Untitled5.exe

```
Constructor 1:
X=0
Y=0
Constructor 2:
X=15
Y=0
Constructor 3:
X=100
Y=5
```

1.2. Overloading Function គឺជាប្រភេទ Function ដែលបង្កើតឡើងដោយមានឈ្មោះដូចគ្នាចាប់ពីរឡើងទៅ តែត្រូវតែមានភាពខុសគ្នាត្រង់ចំនួន Parameter របស់ Function ។

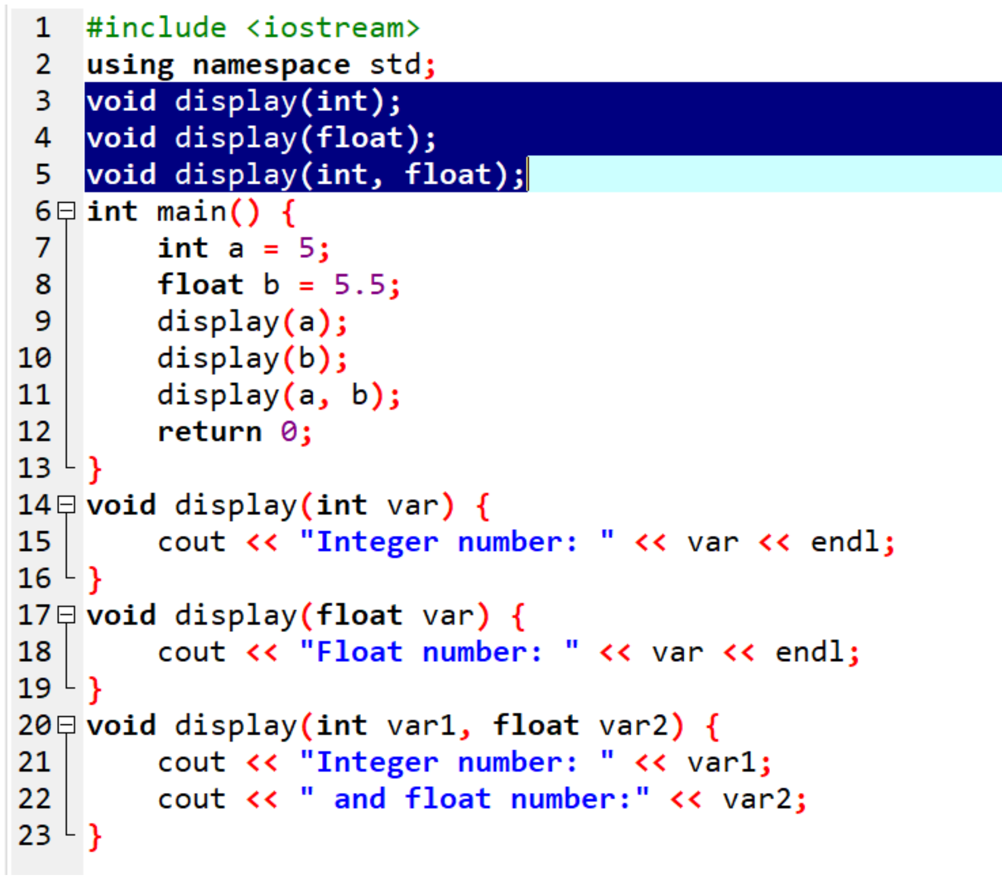


```cpp

int test() { }
int test(int a) { }
float test(double a) { }
int test(int a, double b) { }

```



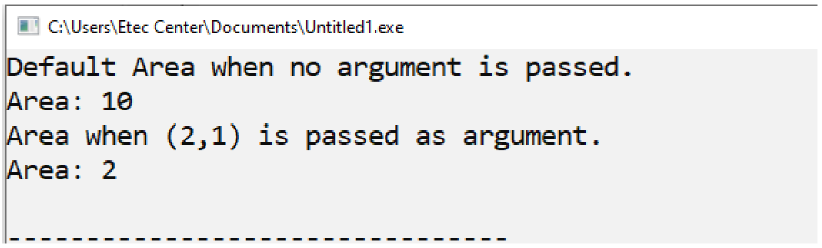
```
void display( )
{
    ......
}

void display( int x)
{
    ......
}

int main ()
{
    int I = 4;
    float f = 3.0;
    display( );
    display( i );
    display( f );
    display( i, f );
    return 0;
}

void display( float y)
{
    ......
}

void display(int x, float y )
{
    ......
}
```

ឧទាហរណ៍ ១៖



```cpp
#include <iostream>
using namespace std;
void display(int);
void display(float);
void display(int, float);
int main() {
    int a = 5;
    float b = 5.5;
    display(a);
    display(b);
    display(a, b);
    return 0;
}
void display(int var) {
    cout << "Integer number: " << var << endl;
}
void display(float var) {
    cout << "Float number: " << var << endl;
}
void display(int var1, float var2) {
    cout << "Integer number: " << var1;
    cout << " and float number:" << var2;
}
```

លទ្ធផលទទួលបាន៖

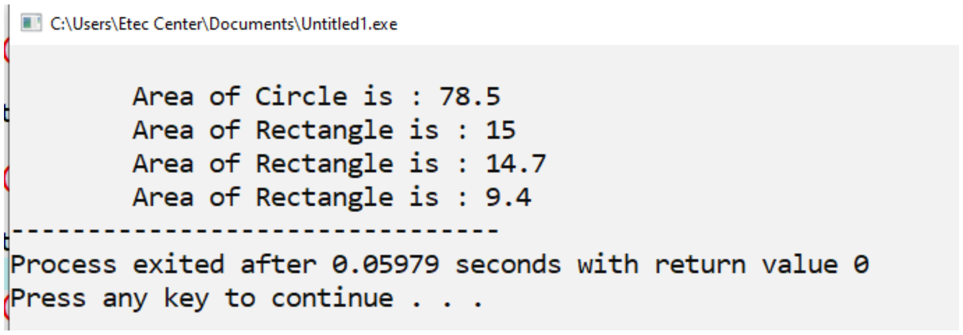


```
C:\Users\Etec Center\Documents\Untitled1.exe
Default Area when no argument is passed.
Area: 10
Area when (2,1) is passed as argument.
Area: 2

--------------------------------
```

ឧទាហរណ៏ ២៖

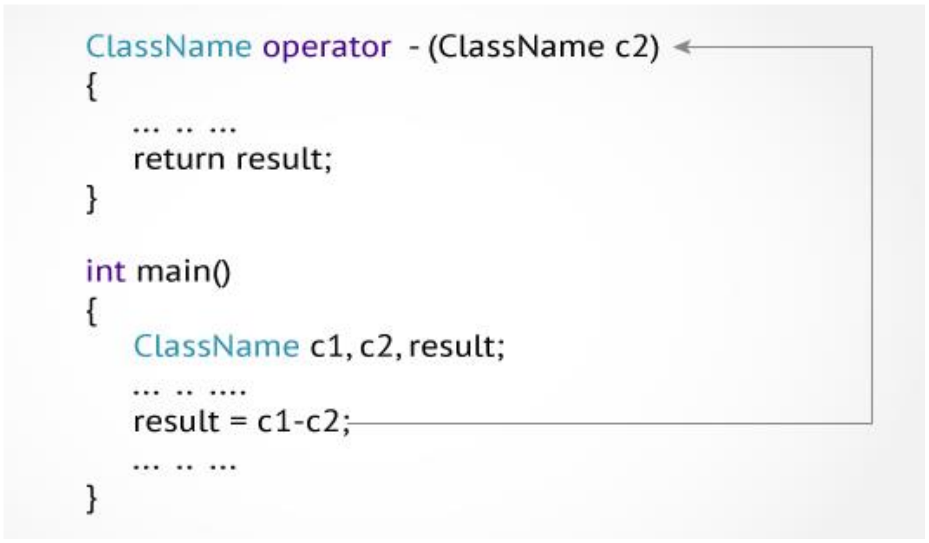
```cpp
#include<iostream>
#include<conio.h>
using namespace std;
		class CalculateArea
		{
			
			public:
			void Area(int r)		//Overloaded Function 1
			{
				cout<<"\n\tArea of Circle is : "<<3.14*r*r;
			}
			void Area(int l,int b)		//Overloaded Function 2
			{
				cout<<"\n\tArea of Rectangle is : "<<l*b;
			}
			void Area(float l,int b)		//Overloaded Function 3
			{
				cout<<"\n\tArea of Rectangle is : "<<l*b;
			}
			void Area(int l,float b)		//Overloaded Function 4
			{
				cout<<"\n\tArea of Rectangle is : "<<l*b;
			}
		};
		int main()
		{
			CalculateArea C;
			C.Area(5);		//Statement 1
			C.Area(5,3);		//Statement 2
			C.Area(7,2.1f);		//Statement 3
			C.Area(4.7f,2);		//Statement 4
		}
```

លទ្ធផលទទួលបាន៖



```
C:\Users\Etec Center\Documents\Untitled1.exe

        Area of Circle is : 78.5
        Area of Rectangle is : 15
        Area of Rectangle is : 14.7
        Area of Rectangle is : 9.4
--------------------------------
Process exited after 0.05979 seconds with return value 0
Press any key to continue . . .
```

1.3. Overloading Operator: គឺជាប្រភេទ Overload ដែលអាចផ្តល់លទ្ធភាពអាចអោយគេអាច គណនានូវ Object ឬ យើងអាចប្រើប្រាស់នូវសញ្ញាណដូចជា +, -, *, /, %, >, <, =, ==, -ល-។

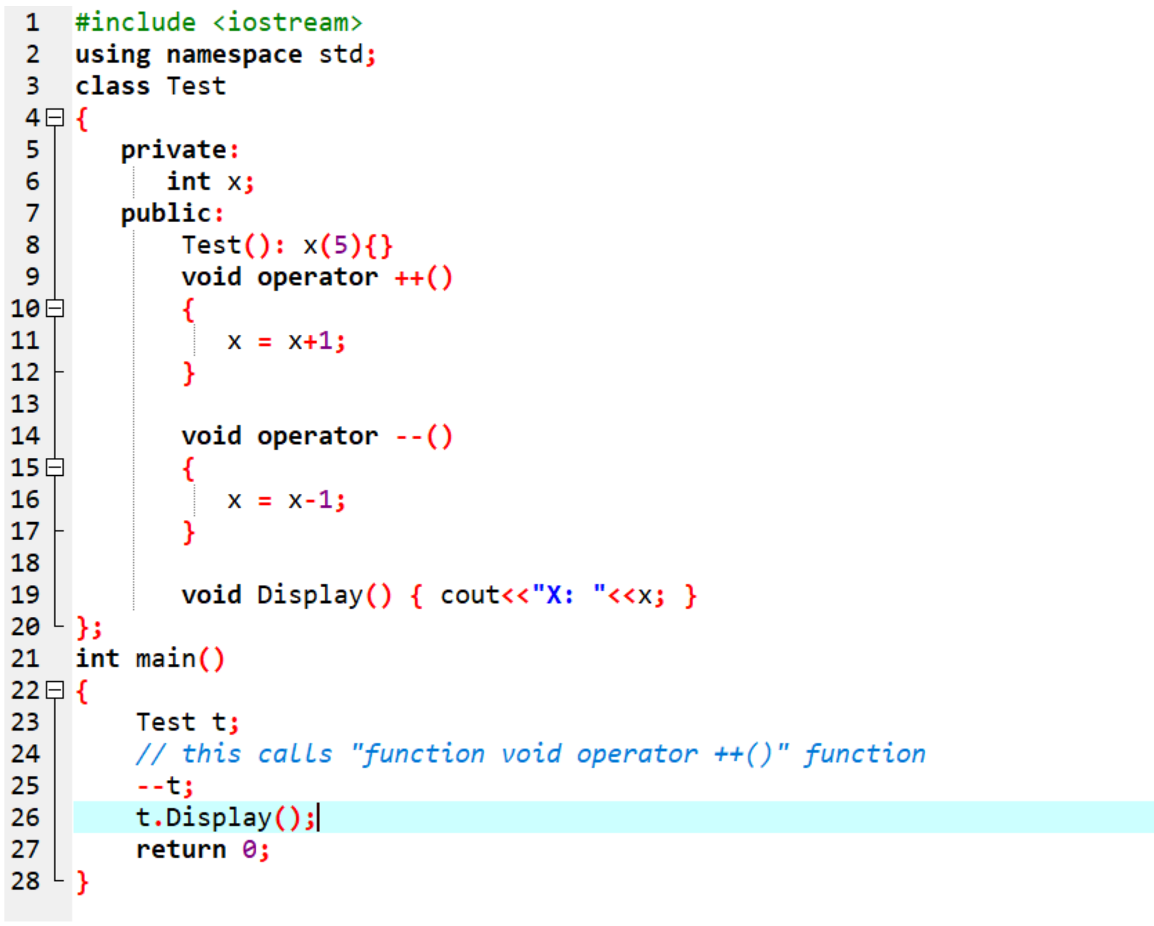


```
ClassName operator - (ClassName c2)
{
    ... .. ...
    return result;
}

int main()
{
    ClassName c1, c2, result;
    ... .. ....
    result = c1-c2;
    ... .. ...
}
```

ទំរង់ទូរទៅ៖

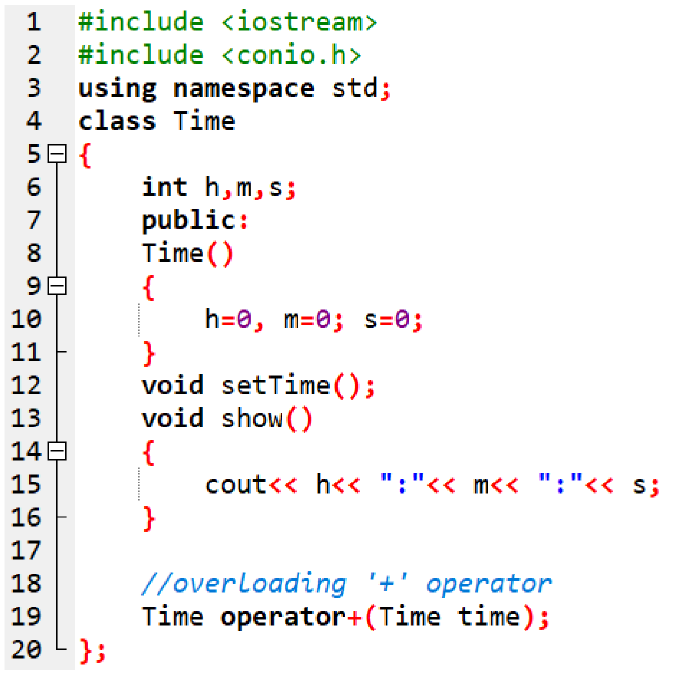
```
Syntax:
class className
{
      ... .. ...
      public
         returnType operator symbol (arguments)
         {
               ... .. ...
         }
      ... .. ...
};
```

ឧទាហរណ៍ ១៖

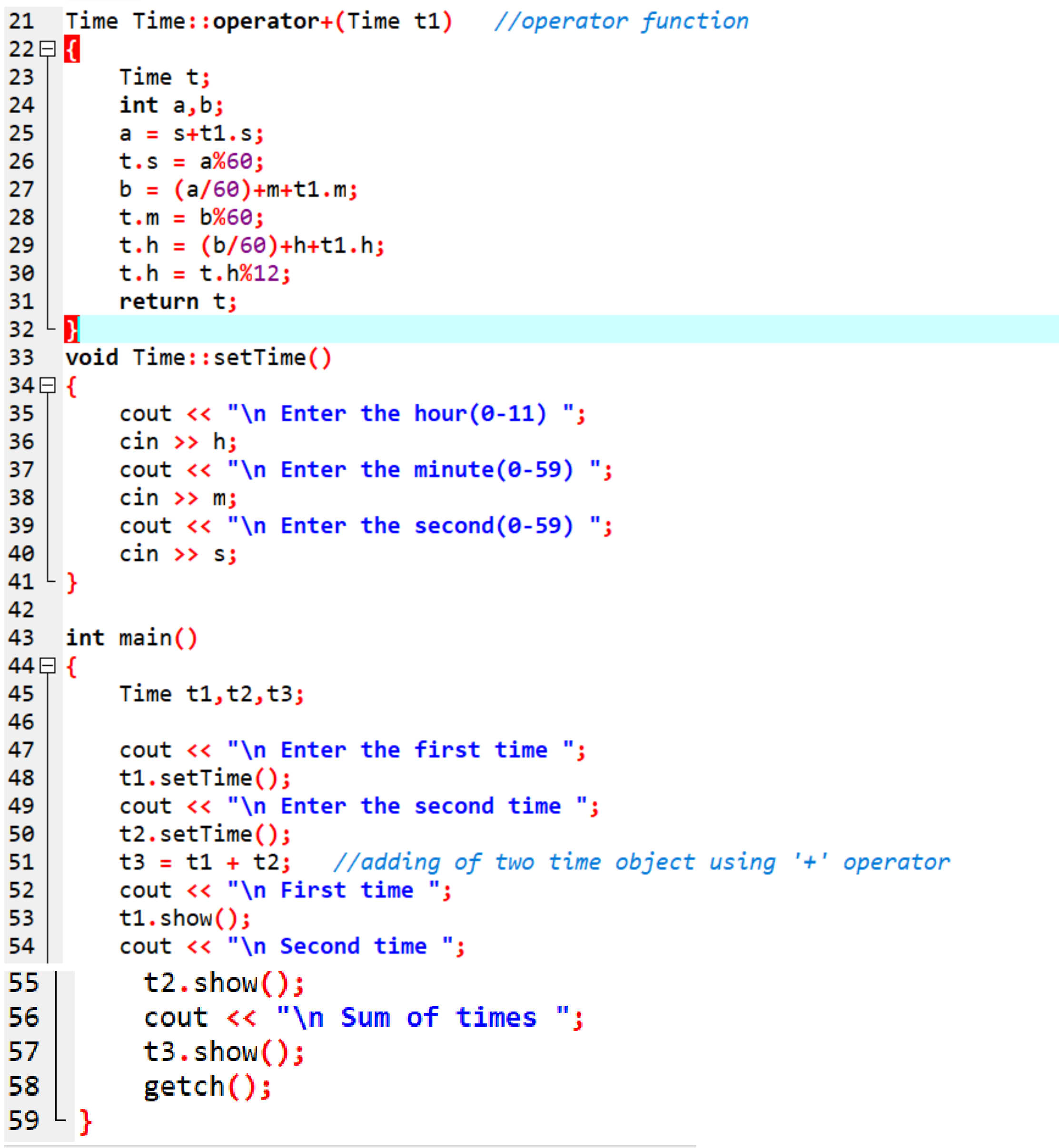


```
#include <iostream>
using namespace std;
class Test
{
    private:
        int x;
    public:
        Test(): x(5){}
        void operator ++()
        {
            x = x+1;
        }

        void operator --()
        {
            x = x-1;
        }

        void Display() { cout<<"X: "<<x; }
};
int main()
{
    Test t;
    // this calls "function void operator ++()" function
    --t;
    t.Display();
    return 0;
}
```

ឧទាហរណ៍ ២៖

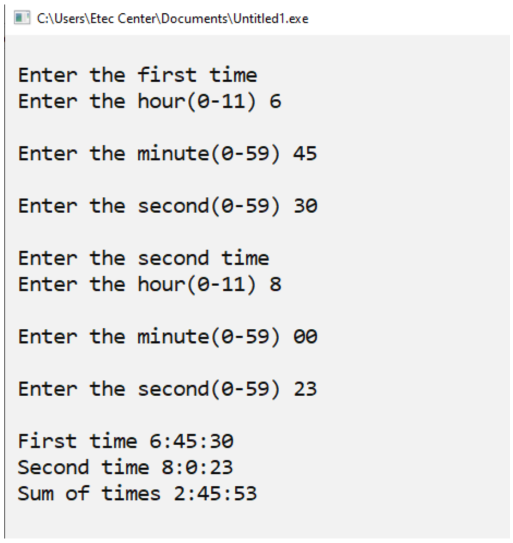


```
#include <iostream>
#include <conio.h>
using namespace std;
class Time
{
    int h,m,s;
    public:
    Time()
    {
        h=0, m=0; s=0;
    }
    void setTime();
    void show()
    {
        cout<< h<< ":"<< m<< ":"<< s;
    }

    //overloading '+' operator
    Time operator+(Time time);
};
```



```cpp
Time Time::operator+(Time t1)    //operator function
{
    Time t;
    int a,b;
    a = s+t1.s;
    t.s = a%60;
    b = (a/60)+m+t1.m;
    t.m = b%60;
    t.h = (b/60)+h+t1.h;
    t.h = t.h%12;
    return t;
}
void Time::setTime()
{
    cout << "\n Enter the hour(0-11) ";
    cin >> h;
    cout << "\n Enter the minute(0-59) ";
    cin >> m;
    cout << "\n Enter the second(0-59) ";
    cin >> s;
}

int main()
{
    Time t1,t2,t3;

    cout << "\n Enter the first time ";
    t1.setTime();
    cout << "\n Enter the second time ";
    t2.setTime();
    t3 = t1 + t2;    //adding of two time object using '+' operator
    cout << "\n First time ";
    t1.show();
    cout << "\n Second time ";
    t2.show();
    cout << "\n Sum of times ";
    t3.show();
    getch();
}
```

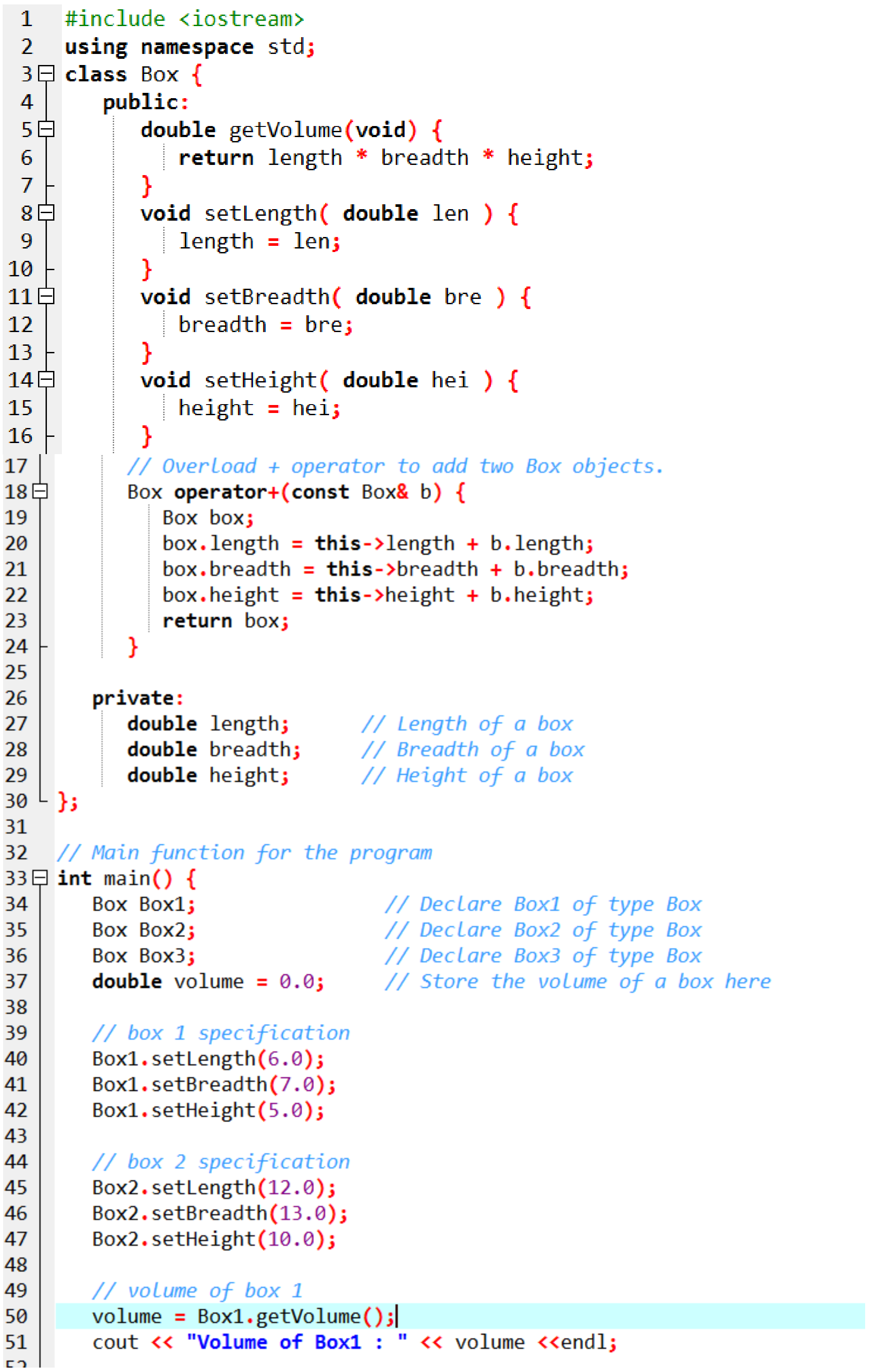
លទ្ធផលទទួលបាន៖



C:\Users\Etec Center\Documents\Untitled1.exe

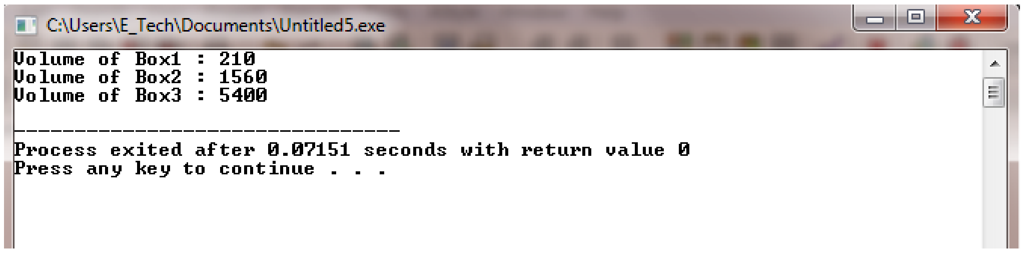
```

 Enter the first time
 Enter the hour(0-11) 6

 Enter the minute(0-59) 45

 Enter the second(0-59) 30

 Enter the second time
 Enter the hour(0-11) 8

 Enter the minute(0-59) 00

 Enter the second(0-59) 23

 First time 6:45:30
 Second time 8:0:23
 Sum of times 2:45:53
```

ឧទាហរណ៍ ៣៖ ចូរបង្កើតនូវ Overloading Operator + សំរាប់អោយគេអាចយក obj1=obj2+obj3 បន្ទាប់មកបង្ហាញលទ្ធផលមកក្រៅវិញ(លទ្ធផលបោះមកក្រៅជាប្រភេទ object class)



```cpp
#include <iostream>
using namespace std;
class 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;
      }

   private:
      double length;      // Length of a box
      double breadth;     // Breadth of a box
      double height;      // Height of a 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;
}
```

លទ្ធផលទទួលបាន៖

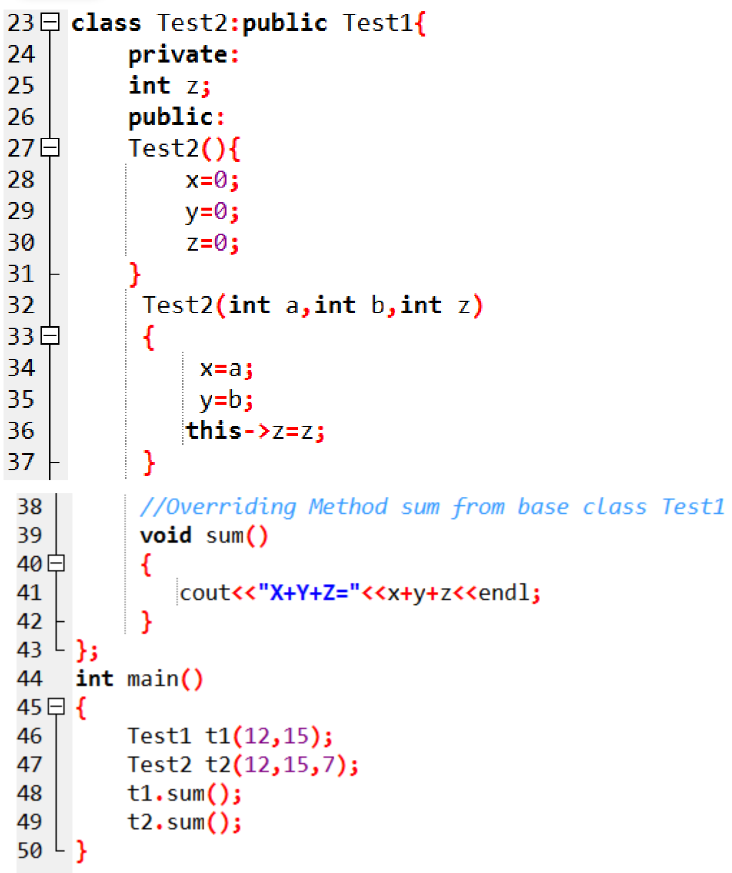


```
C:\Users\E_Tech\Documents\Untitled5.exe
Volume of Box1 : 210
Volume of Box2 : 1560
Volume of Box3 : 5400

--------------------------------
Process exited after 0.07151 seconds with return value 0
Press any key to continue . . .
```

៤) **overriding Method**: គឺជាប្រភេទ Method ដែលមានឈ្មោះដូចគ្នាទាំងក្នុង Base Class និង Sub Class ដែល អាចអោយ Sub Class Overriding នូវ Method ទាំងនោះបាន

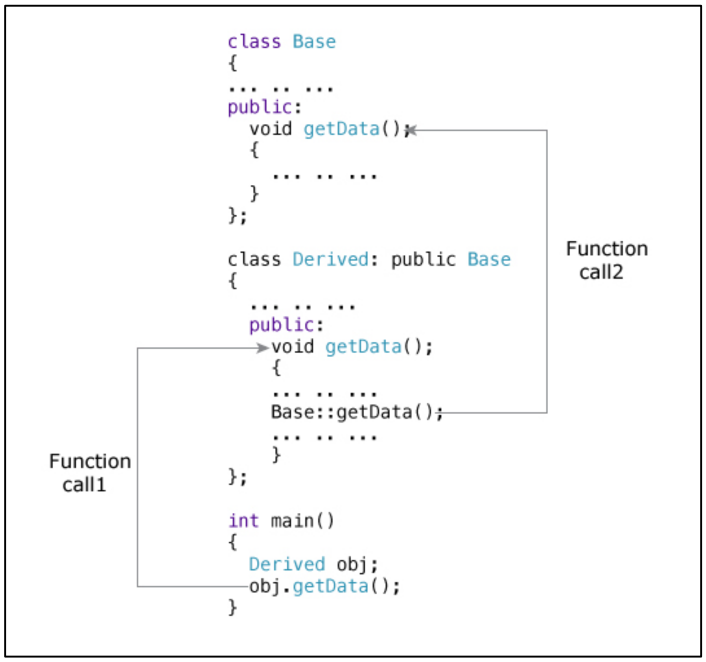
ឧទាហរណ៍ ១៖

```
#include<iostream>
using namespace std;
class Test1{
    protected:
    int x;
    int y;
    public:
     Test1()
     {
        x=0;
        y=0;
     }
     Test1(int x,int y)
     {
        this->x=x;
        this->y=y;
     }
     void sum()
     {
        cout<<"X+Y="<<x+y<<endl;
     }
};
```



```
class Test2:public Test1{
    private:
    int z;
    public:
    Test2(){
        x=0;
        y=0;
        z=0;
    }
     Test2(int a,int b,int z)
     {
         x=a;
         y=b;
        this->z=z;
     }
     //Overriding Method sum from base class Test1
     void sum()
     {
        cout<<"X+Y+Z="<<x+y+z<<endl;
     }
};
int main()
{
    Test1 t1(12,15);
    Test2 t2(12,15,7);
    t1.sum();
    t2.sum();
}
```

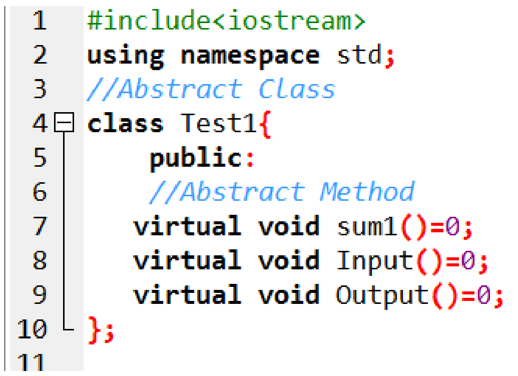
ឧទាហរណ៍ ២៖ ចូរធ្វើការ Design នូវ Class ដូចខាងក្រោម និង Overriding លើ Method ដូចខាងក្រោម៖



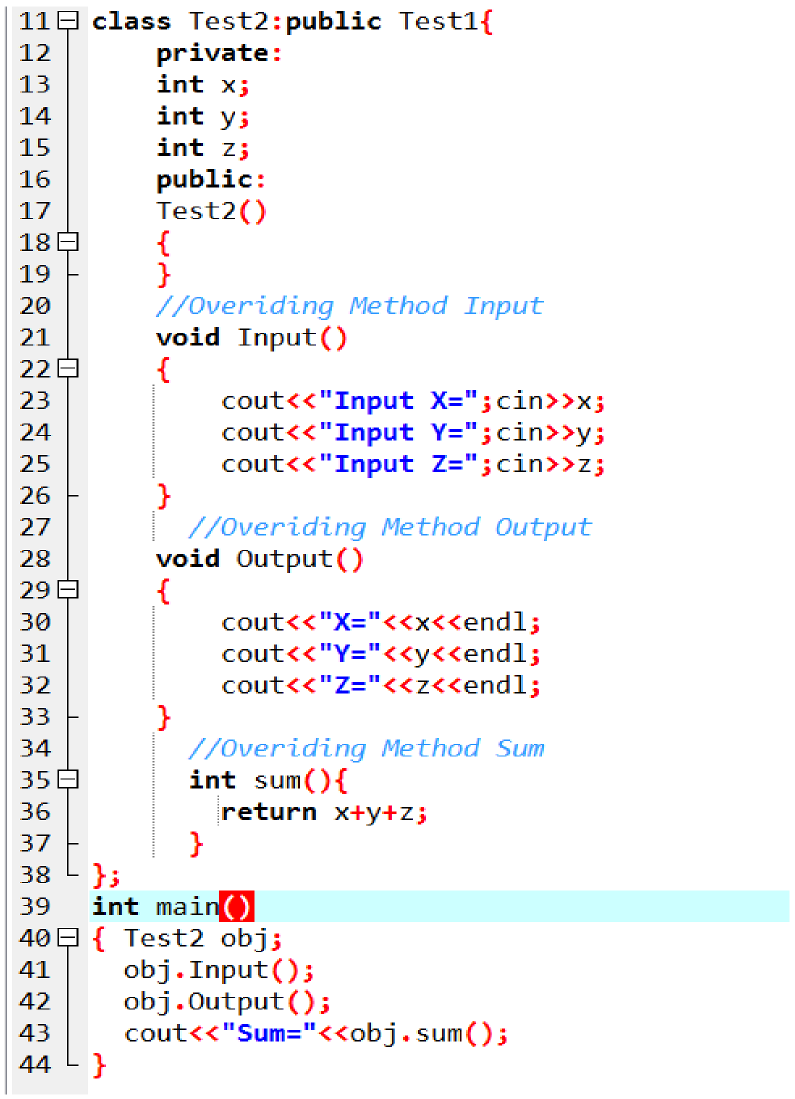
```
class Base
{
... .. ...
public:
  void getData();
  {
    ... .. ...
  }
};

class Derived: public Base
{
  ... .. ...
  public:
    void getData();
    {
    ... .. ...
    Base::getData();
    ... .. ...
    }
};

int main()
{
  Derived obj;
  obj.getData();
}
```

៥) Abstract Class: គឺជាប្រភេទ Class ដែលមាននូវ Method Abstract មួយយ៉ាងតិច។ Method Abstract គឺជាប្រភេទ Method ដែលមានតែប្រកាសតែគ្មានខ្លួន។ ការបង្កើតឡើងនូវ Method Abstract ឡើងគឺ ផ្តល់លទ្ធភាពអោយគេអាច Overriding ទៅលើ Method ដែលមាន ស្រាប់នោះនៅក្នុង Sub Class។

ឧទាហរណ៍៖

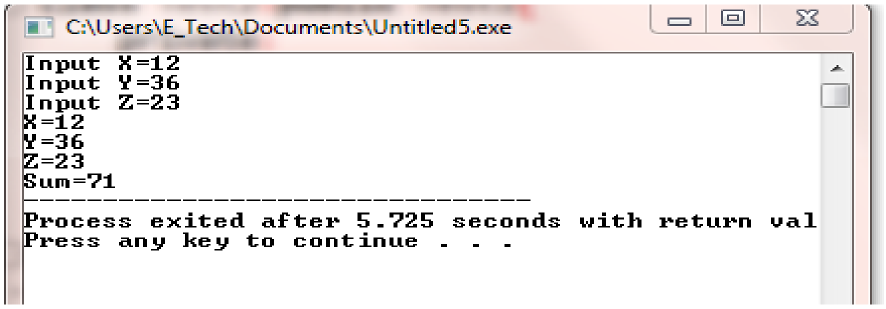


```
#include<iostream>
using namespace std;
//Abstract Class
class Test1{
     public:
     //Abstract Method
    virtual void sum1()=0;
    virtual void Input()=0;
    virtual void Output()=0;
};

```



```
class Test2:public Test1{
     private:
     int x;
     int y;
     int z;
     public:
     Test2()
     {
     }
     //Overiding Method Input
     void Input()
     {
         cout<<"Input X=";cin>>x;
         cout<<"Input Y=";cin>>y;
         cout<<"Input Z=";cin>>z;
     }
       //Overiding Method Output
     void Output()
     {
         cout<<"X="<<x<<endl;
         cout<<"Y="<<y<<endl;
         cout<<"Z="<<z<<endl;
     }
       //Overiding Method Sum
       int sum(){
         return x+y+z;
       }
};
int main()
{ Test2 obj;
  obj.Input();
  obj.Output();
  cout<<"Sum="<<obj.sum();
}
```

លទ្ធផលទទួលបាន៖

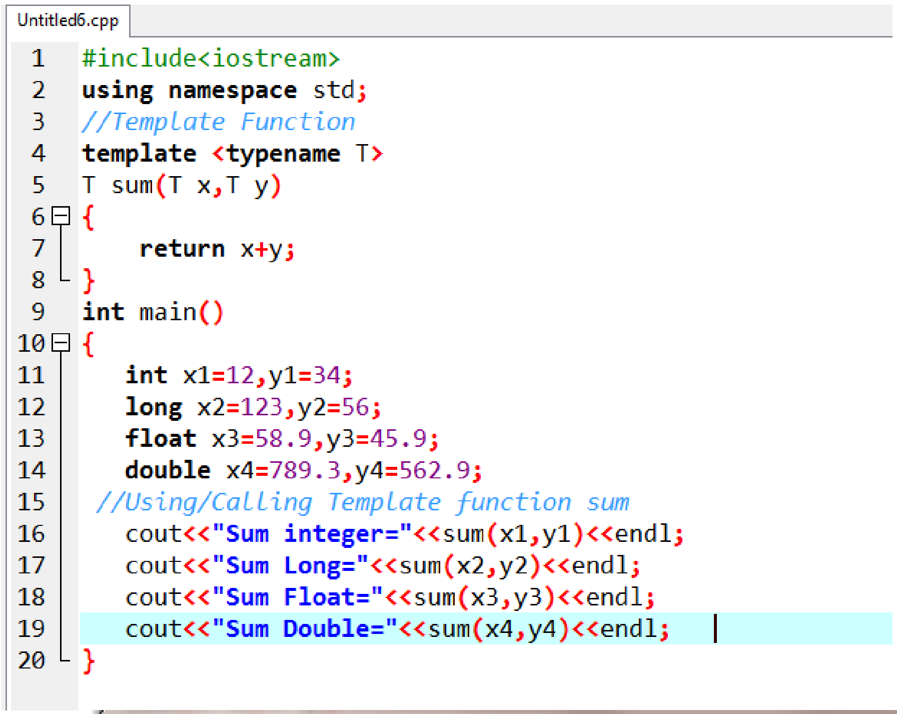


```
C:\Users\E_Tech\Documents\Untitled5.exe
Input X=12
Input Y=36
Input Z=23
X=12
Y=36
Z=23
Sum=71
--------------------------------
Process exited after 5.725 seconds with return val
Press any key to continue . . .
```

៦) Template: គឺជាសំដៅលើការបង្កើតនូវគំរូមួយទៅលើ Class និង Method ដែលគេ ប្រើប្រាស់សំរាប់កាត់បន្ថយនូវការសរសេរកូដច្រំដែរច្រើនដង ឬ ការបង្កើតនូវ Function ច្រើនក្នុងពេលតែមួយ។ នៅក្នុងចំនុចនេះគេបែងចែក Template ជា ពីរប្រភេទគឺ៖

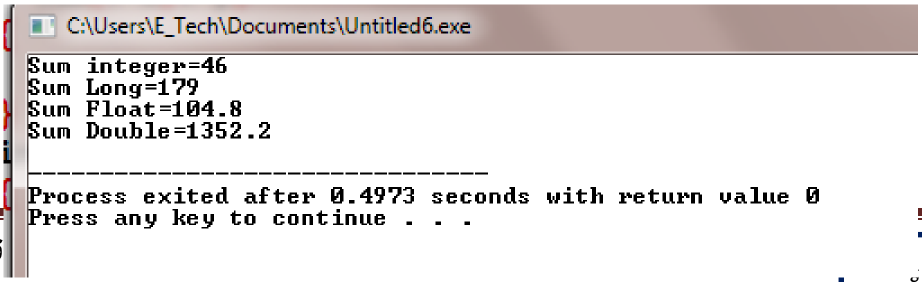
៦.១. Function Template: គឺជាប្រភេទ នៃ Template ដែលគេអាចបង្កើតនូវ Function គំរូ មួយ ដែលអាចអោយគេប្រើប្រាស់វាជារួមបាន។

ឧទាហរណ៍ ១៖



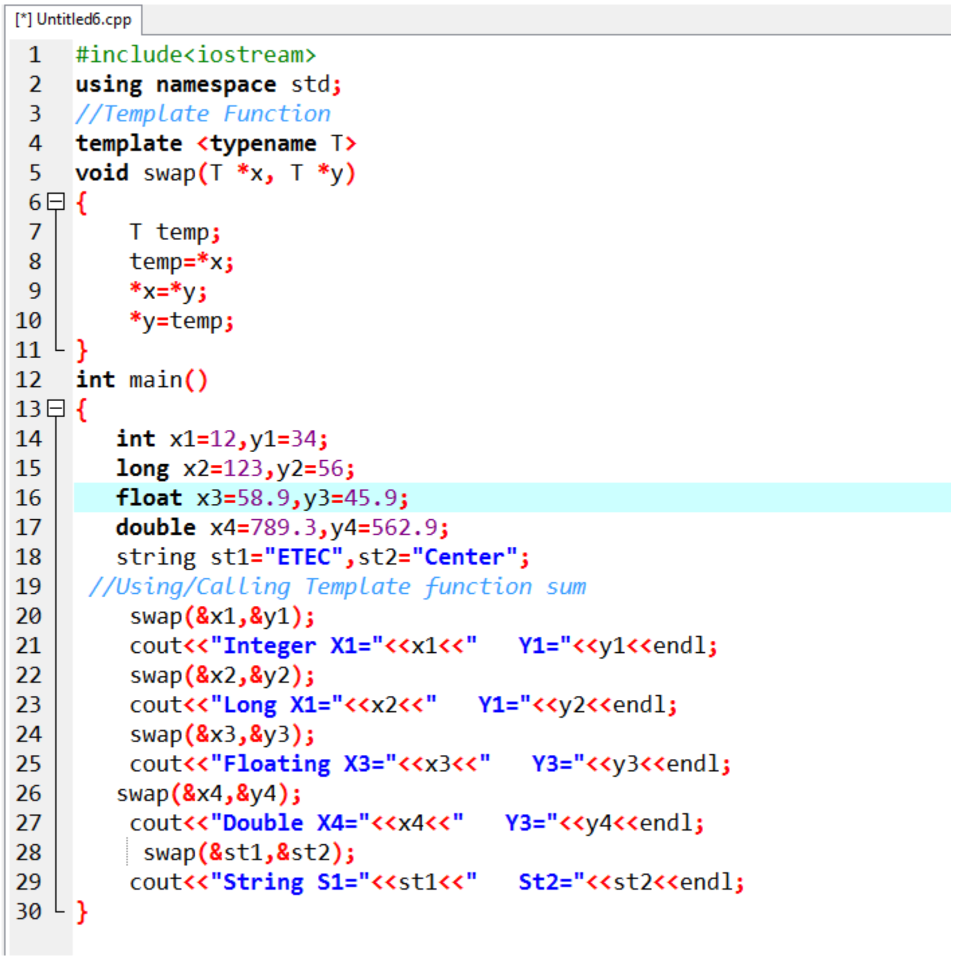
```cpp
#include<iostream>
using namespace std;
//Template Function
template <typename T>
T sum(T x,T y)
{
    return x+y;
}
int main()
{
   int x1=12,y1=34;
   long x2=123,y2=56;
   float x3=58.9,y3=45.9;
   double x4=789.3,y4=562.9;
 //Using/Calling Template function sum
   cout<<"Sum integer="<<sum(x1,y1)<<endl;
   cout<<"Sum Long="<<sum(x2,y2)<<endl;
   cout<<"Sum Float="<<sum(x3,y3)<<endl;
   cout<<"Sum Double="<<sum(x4,y4)<<endl;
}
```

លទ្ធផលទទួលបាន៖



```
C:\Users\E_Tech\Documents\Untitled6.exe
Sum integer=46
Sum Long=179
Sum Float=104.8
Sum Double=1352.2
--------------------------------
Process exited after 0.4973 seconds with return value 0
Press any key to continue . . .
```

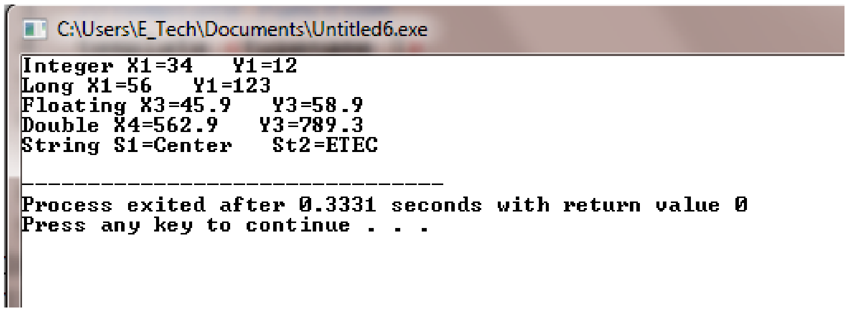
ឧទាហរណ៏ ២៖



[*] Untitled6.cpp

```cpp
#include<iostream>
using namespace std;
//Template Function
template <typename T>
void swap(T *x, T *y)
{
    T temp;
    temp=*x;
    *x=*y;
    *y=temp;
}
int main()
{
   int x1=12,y1=34;
   long x2=123,y2=56;
   float x3=58.9,y3=45.9;
   double x4=789.3,y4=562.9;
   string st1="ETEC",st2="Center";
 //Using/Calling Template function sum
    swap(&x1,&y1);
    cout<<"Integer X1="<<x1<<"   Y1="<<y1<<endl;
    swap(&x2,&y2);
    cout<<"Long X1="<<x2<<"   Y1="<<y2<<endl;
    swap(&x3,&y3);
    cout<<"Floating X3="<<x3<<"   Y3="<<y3<<endl;
   swap(&x4,&y4);
    cout<<"Double X4="<<x4<<"   Y3="<<y4<<endl;
     swap(&st1,&st2);
    cout<<"String S1="<<st1<<"   St2="<<st2<<endl;
}
```

លទ្ធផលទទួលបាន៖

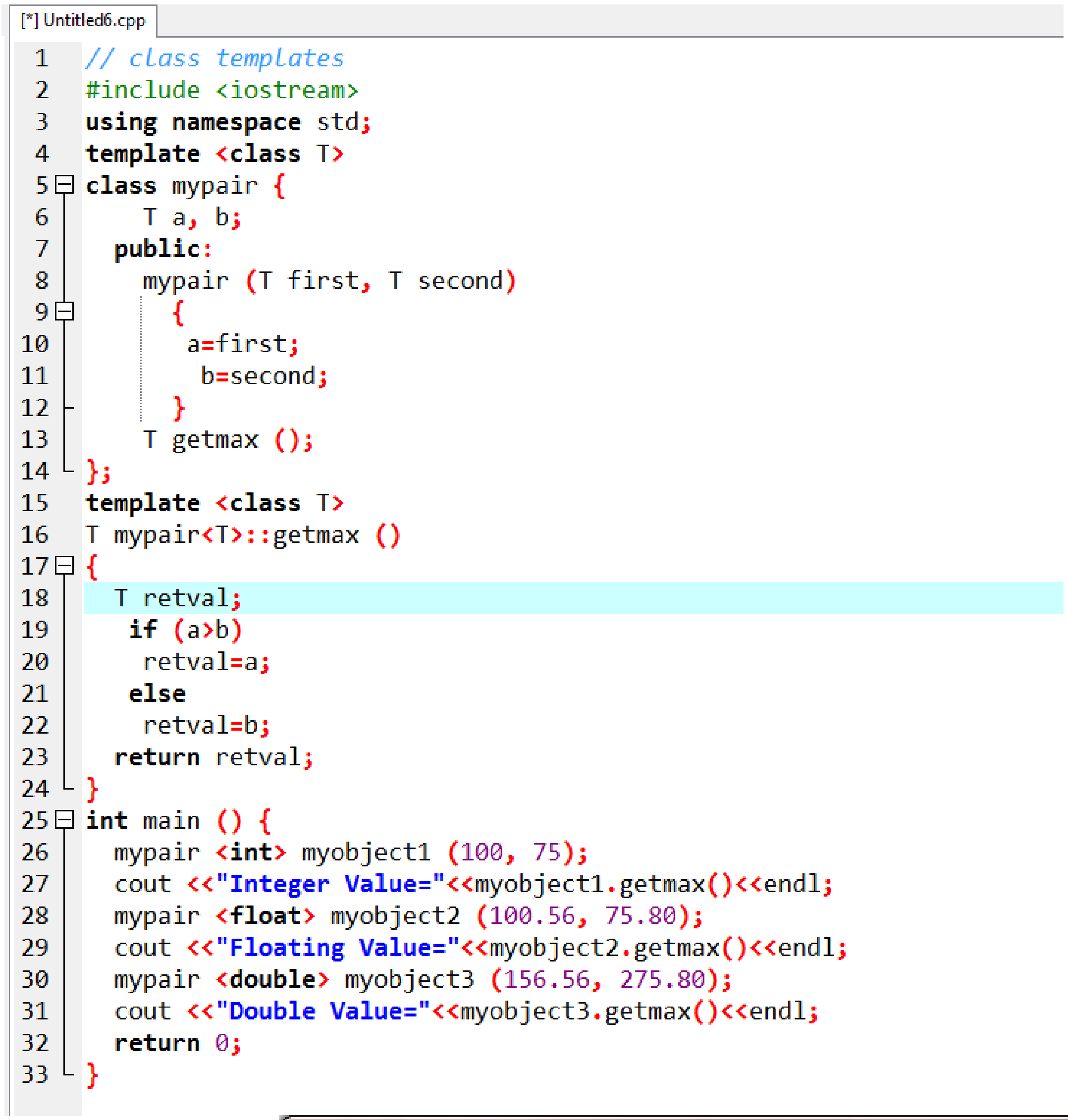


C:\Users\E_Tech\Documents\Untitled6.exe

```
Integer X1=34   Y1=12
Long X1=56   Y1=123
Floating X3=45.9   Y3=58.9
Double X4=562.9   Y3=789.3
String S1=Center   St2=ETEC

--------------------------------
Process exited after 0.3331 seconds with return value 0
Press any key to continue . . .
```

៦.១. Class Template: គឺជាប្រភេទ Class មួយដែលគេអាចបង្កើតវាជាគំរូមួយ សំរាប់អោយគេអាចប្រើប្រាស់នូវ Data Member និង Function Member របស់ វាច្រើនទំរង់។

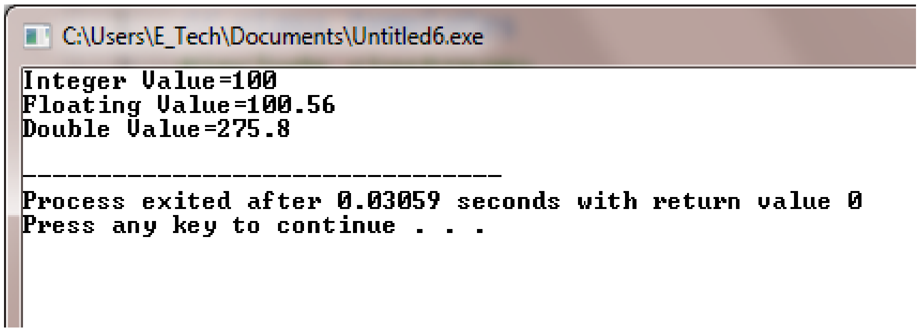
ឧទាហរណ៏១៖



[*] Untitled6.cpp

```cpp
// class templates
#include <iostream>
using namespace std;
template <class T>
class mypair {
    T a, b;
  public:
    mypair (T first, T second)
      {
        a=first;
         b=second;
      }
    T getmax ();
};
template <class T>
T mypair<T>::getmax ()
{
  T retval;
   if (a>b)
    retval=a;
   else
    retval=b;
  return retval;
}
int main () {
  mypair <int> myobject1 (100, 75);
  cout <<"Integer Value="<<myobject1.getmax()<<endl;
  mypair <float> myobject2 (100.56, 75.80);
  cout <<"Floating Value="<<myobject2.getmax()<<endl;
  mypair <double> myobject3 (156.56, 275.80);
  cout <<"Double Value="<<myobject3.getmax()<<endl;
  return 0;
}
```

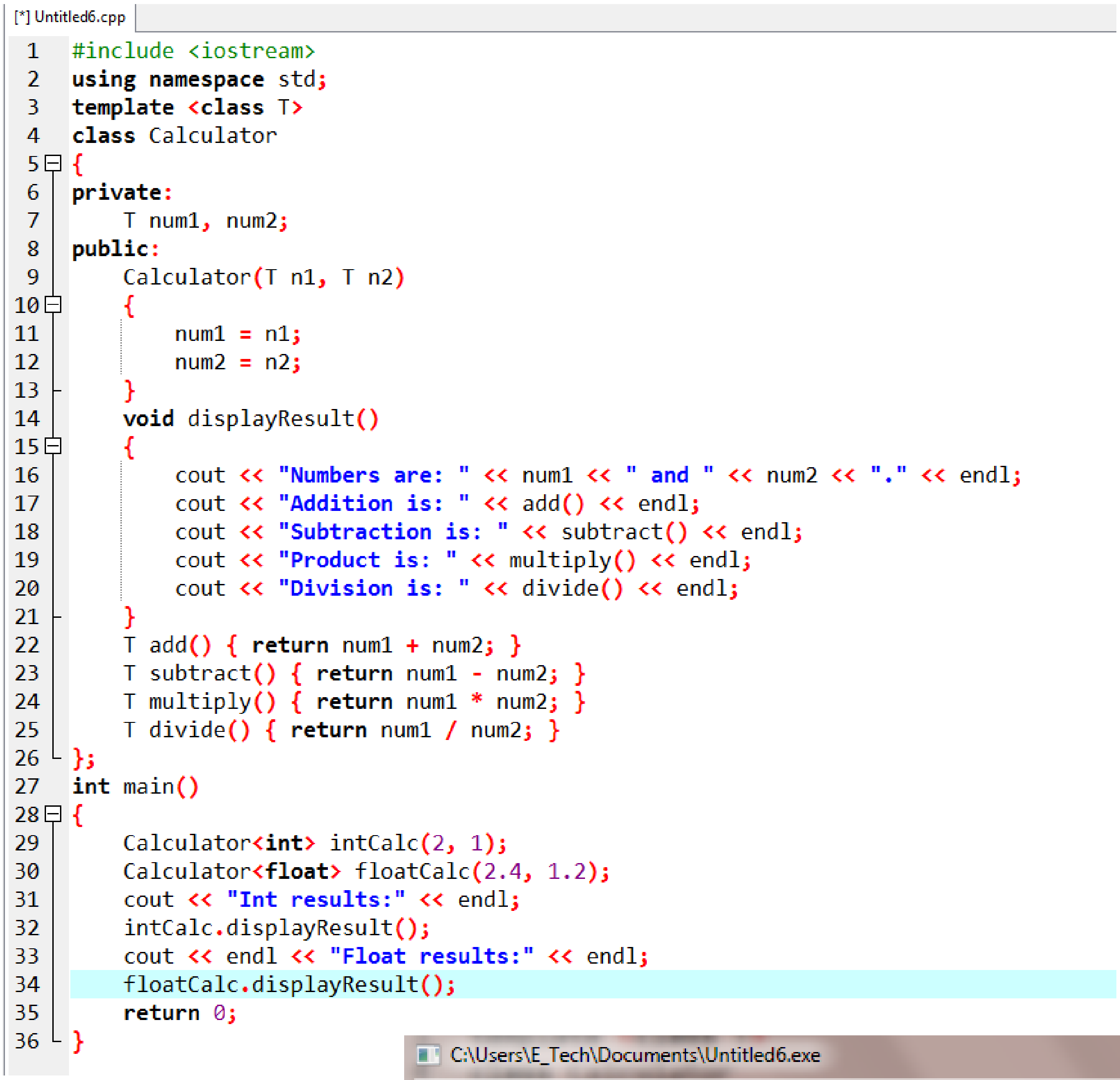
លទ្ធផលទទួលបាន៖



C:\Users\E_Tech\Documents\Untitled6.exe

```
Integer Value=100
Floating Value=100.56
Double Value=275.8

--------------------------------
Process exited after 0.03059 seconds with return value 0
Press any key to continue . . .
```

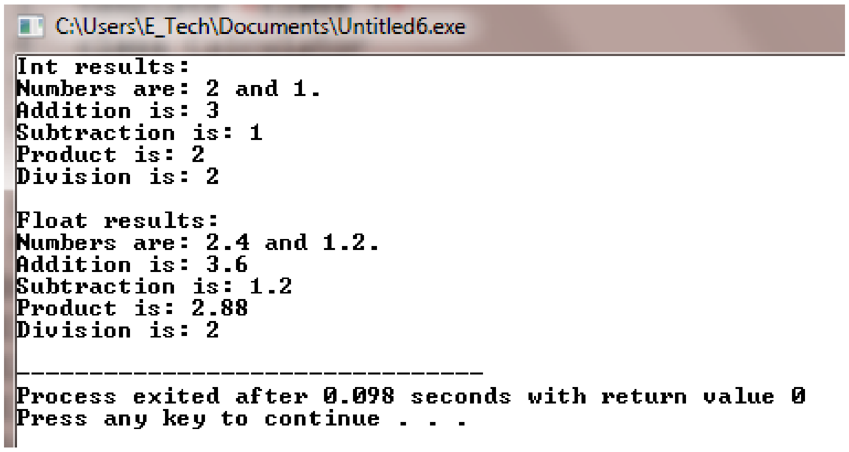
ឧទាហរណ៍ ២៖



[*] Untitled6.cpp

```cpp
#include <iostream>
using namespace std;
template <class T>
class Calculator
{
private:
    T num1, num2;
public:
    Calculator(T n1, T n2)
    {
        num1 = n1;
        num2 = n2;
    }
    void displayResult()
    {
        cout << "Numbers are: " << num1 << " and " << num2 << "." << endl;
        cout << "Addition is: " << add() << endl;
        cout << "Subtraction is: " << subtract() << endl;
        cout << "Product is: " << multiply() << endl;
        cout << "Division is: " << divide() << endl;
    }
    T add() { return num1 + num2; }
    T subtract() { return num1 - num2; }
    T multiply() { return num1 * num2; }
    T divide() { return num1 / num2; }
};
int main()
{
    Calculator<int> intCalc(2, 1);
    Calculator<float> floatCalc(2.4, 1.2);
    cout << "Int results:" << endl;
    intCalc.displayResult();
    cout << endl << "Float results:" << endl;
    floatCalc.displayResult();
    return 0;
}
```

លទ្ធផលទទួលបាន៖



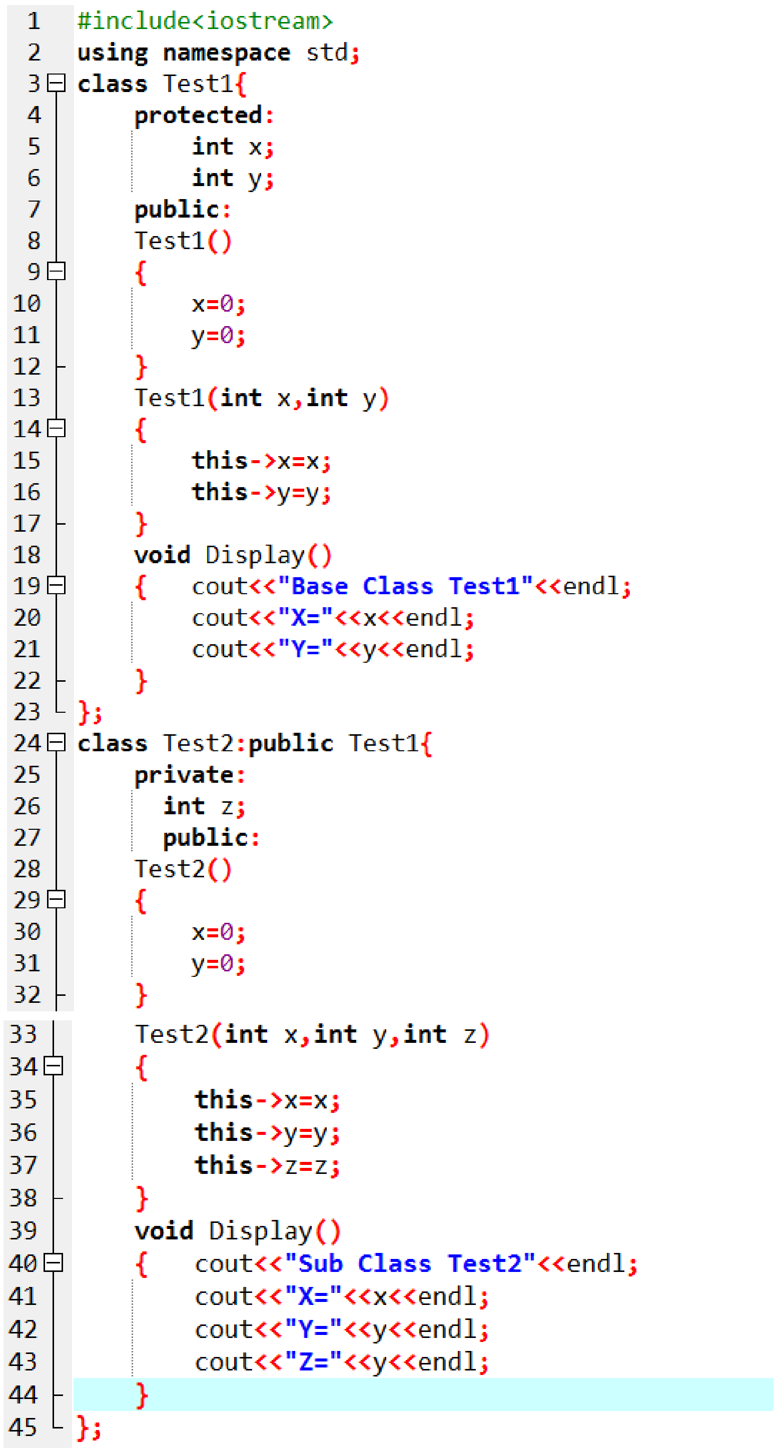
C:\Users\E_Tech\Documents\Untitled6.exe

```
Int results:
Numbers are: 2 and 1.
Addition is: 3
Subtraction is: 1
Product is: 2
Division is: 2

Float results:
Numbers are: 2.4 and 1.2.
Addition is: 3.6
Subtraction is: 1.2
Product is: 2.88
Division is: 2

--------------------------------
Process exited after 0.098 seconds with return value 0
Press any key to continue . . .
```

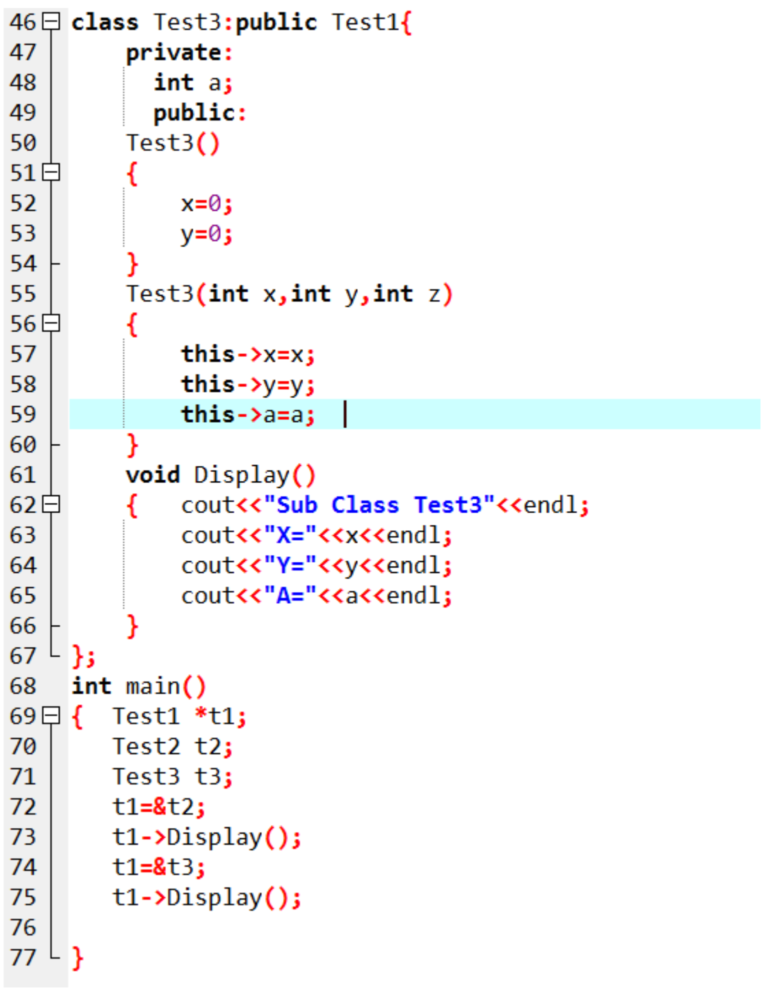
៧). Early Binding/ Late Binding របស់ Polymorphism

៧.១. Early Binding: គឺជាប្រភេទ Concept របស់ Polymorphism ដែលវាដំណើរ ជា លើក ដំបូង ក្នុងពេល Compile Code ដែលវាត្រូវជ្រើសរើសនូវ Function ណាមួយដែលត្រូវ ដំណើរ ការនៅពេល Object របស់ Base Class ចង្អុលទៅកាន់ Object របស់ Derived Class។

ឧទាហរណ៍ ១៖

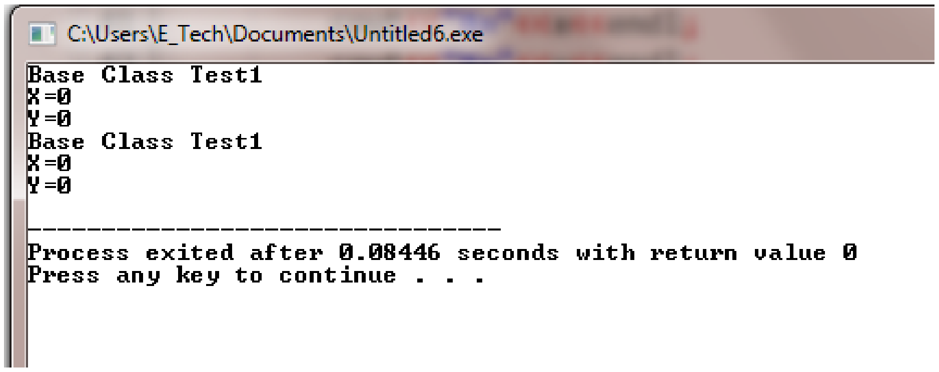


```
#include<iostream>
using namespace std;
class Test1{
    protected:
        int x;
        int y;
    public:
    Test1()
    {
        x=0;
        y=0;
    }
    Test1(int x,int y)
    {
        this->x=x;
        this->y=y;
    }
    void Display()
    {   cout<<"Base Class Test1"<<endl;
        cout<<"X="<<x<<endl;
        cout<<"Y="<<y<<endl;
    }
};
class Test2:public Test1{
    private:
      int z;
      public:
    Test2()
    {
        x=0;
        y=0;
    }
    Test2(int x,int y,int z)
    {
        this->x=x;
        this->y=y;
        this->z=z;
    }
    void Display()
    {   cout<<"Sub Class Test2"<<endl;
        cout<<"X="<<x<<endl;
        cout<<"Y="<<y<<endl;
        cout<<"Z="<<y<<endl;
    }
};
```



```
class Test3:public Test1{
    private:
      int a;
      public:
    Test3()
    {
        x=0;
        y=0;
    }
    Test3(int x,int y,int z)
    {
        this->x=x;
        this->y=y;
        this->a=a;
    }
    void Display()
    {   cout<<"Sub Class Test3"<<endl;
        cout<<"X="<<x<<endl;
        cout<<"Y="<<y<<endl;
        cout<<"A="<<a<<endl;
    }
};
int main()
{  Test1 *t1;
   Test2 t2;
   Test3 t3;
   t1=&t2;
   t1->Display();
   t1=&t3;
   t1->Display();

}
```

លទ្ធផលទទួលបាន៖

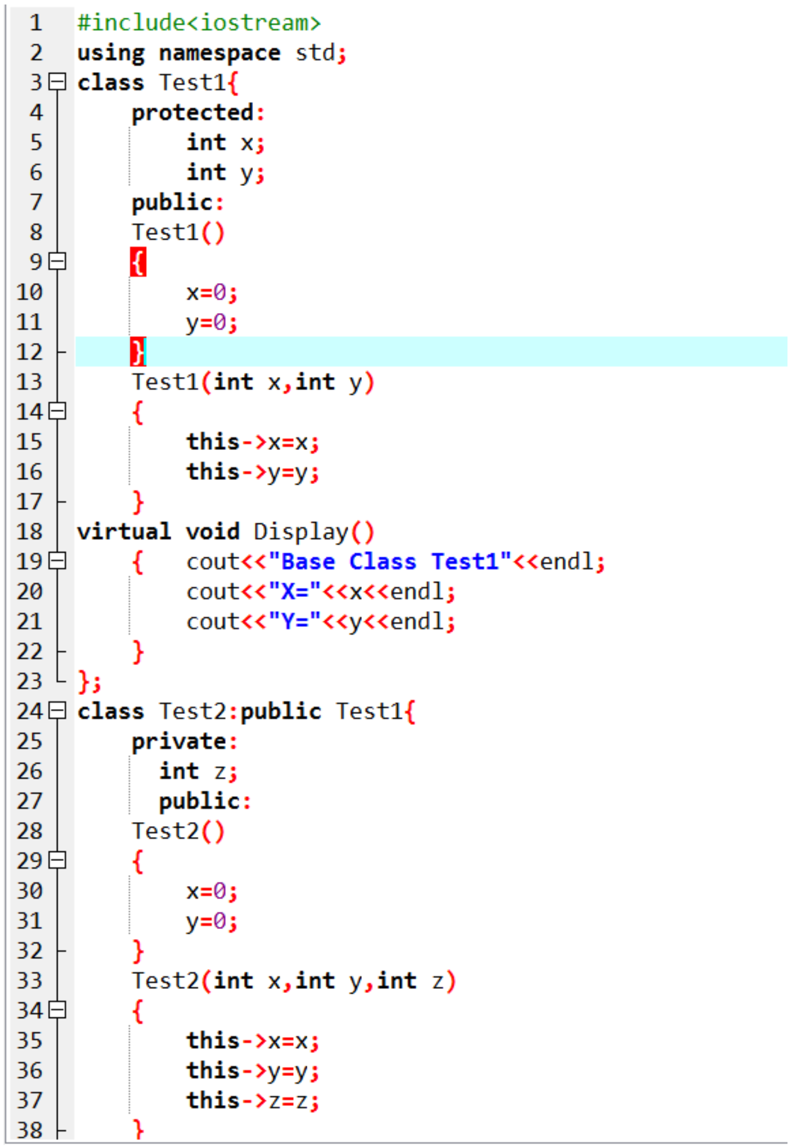


```
C:\Users\E_Tech\Documents\Untitled6.exe
Base Class Test1
X=0
Y=0
Base Class Test1
X=0
Y=0

--------------------------------
Process exited after 0.08446 seconds with return value 0
Press any key to continue . . .
```

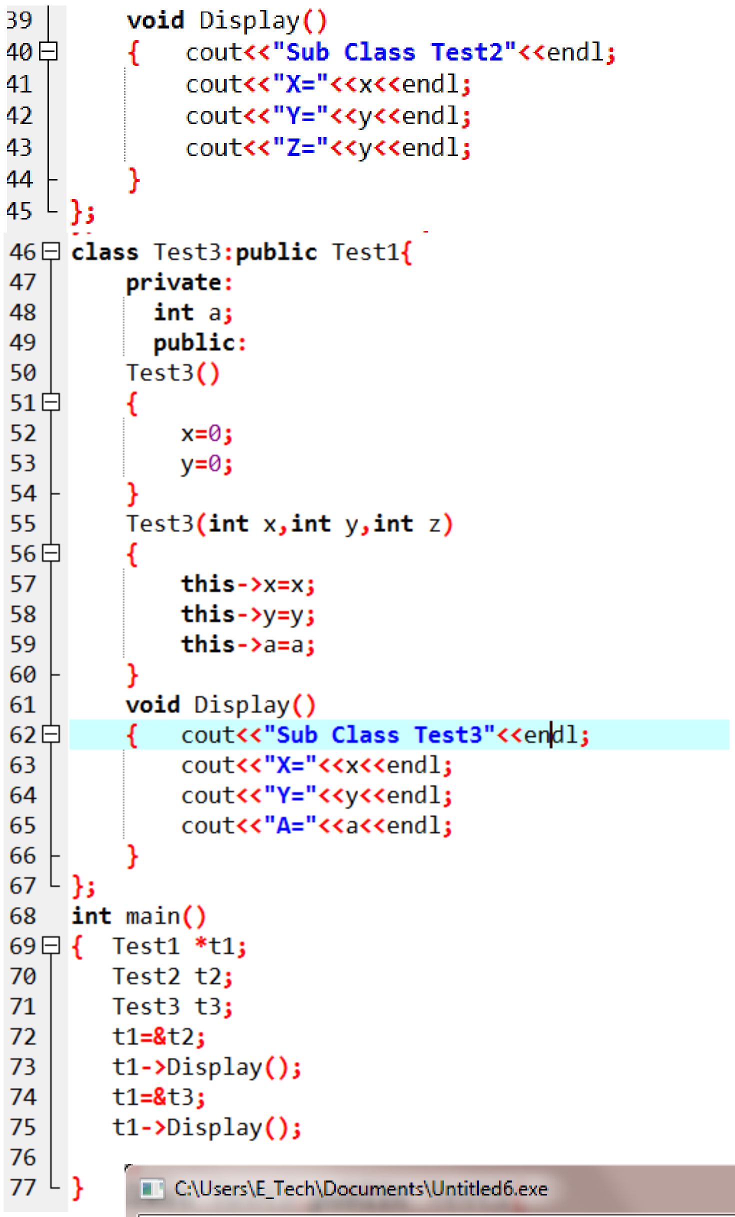
****** យើងសង្កេតឃើញថា លទ្ធផលដែលទទួលបានគឺសុទ្ធតែបានចេញពី Base Class ទាំងអស់ ពោលគឺទោះបីយើងព្យាយាមយក Object Pointer របស់ Base Class ទៅចង្អុលទៅកាន់ Object របស់ Derived Class ក៏ដោយនៅតែលទ្ធផលទទួលបានគឺខុសពីការគិតរបស់យើង។ ដូច្នេះចំនុចនេះគឺ ប្រភេទ Early Binding ឬ Compile time binding។

៧.១. Late Binding: គឺជាប្រភេទ Concept របស់ Polymorphism ដែលវាដំណើរការពេល Object កកើតឡើងក្នុងដំណាក់កាល Run Time ឬ Late Binding ហើយចង្អុលទៅកាន់ Object របស់ Sub Class ដោយប្រើប្រាស់នូវ Function virtual ។

ឧទាហរណ៍ ៖

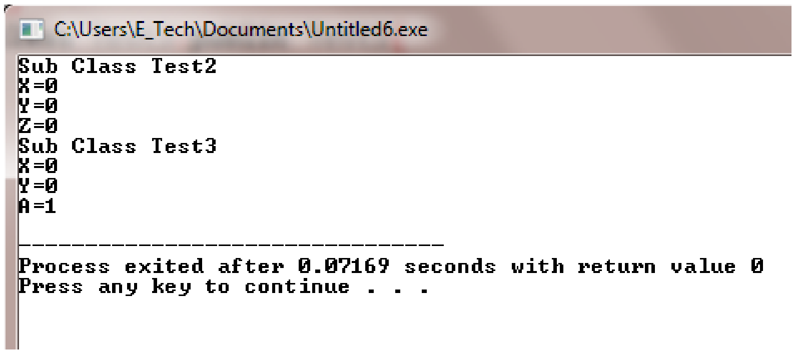


```
#include<iostream>
using namespace std;
class Test1{
	protected:
		int x;
		int y;
	public:
	Test1()
	{
		x=0;
		y=0;
	}
	Test1(int x,int y)
	{
		this->x=x;
		this->y=y;
	}
virtual void Display()
	{	cout<<"Base Class Test1"<<endl;
		cout<<"X="<<x<<endl;
		cout<<"Y="<<y<<endl;
	}
};
class Test2:public Test1{
	private:
	  int z;
	  public:
	Test2()
	{
		x=0;
		y=0;
	}
	Test2(int x,int y,int z)
	{
		this->x=x;
		this->y=y;
		this->z=z;
	}
```



```
    void Display()
    {   cout<<"Sub Class Test2"<<endl;
        cout<<"X="<<x<<endl;
        cout<<"Y="<<y<<endl;
        cout<<"Z="<<y<<endl;
    }
};
class Test3:public Test1{
    private:
      int a;
      public:
    Test3()
    {
        x=0;
        y=0;
    }
    Test3(int x,int y,int z)
    {
        this->x=x;
        this->y=y;
        this->a=a;
    }
    void Display()
    {   cout<<"Sub Class Test3"<<endl;
        cout<<"X="<<x<<endl;
        cout<<"Y="<<y<<endl;
        cout<<"A="<<a<<endl;
    }
};
int main()
{  Test1 *t1;
   Test2 t2;
   Test3 t3;
   t1=&t2;
   t1->Display();
   t1=&t3;
   t1->Display();

}
```

លទ្ធផលទទួលបាន៖



```
C:\Users\E_Tech\Documents\Untitled6.exe
Sub Class Test2
X=0
Y=0
Z=0
Sub Class Test3
X=0
Y=0
A=1

--------------------------------
Process exited after 0.07169 seconds with return value 0
Press any key to continue . . .
```

យើងសង្កេតឃើញថា ក្រោយពីដាក់នូវ Function virtual នៅពីមុខ Function Display មក ហើយអោយ Object របស់ Base Class ចង្អុលទៅ កាន់ Object របស់ Sub Class គឺវាចាប់យក ទិន្នន័យរបស់ Sub Class វិញ។