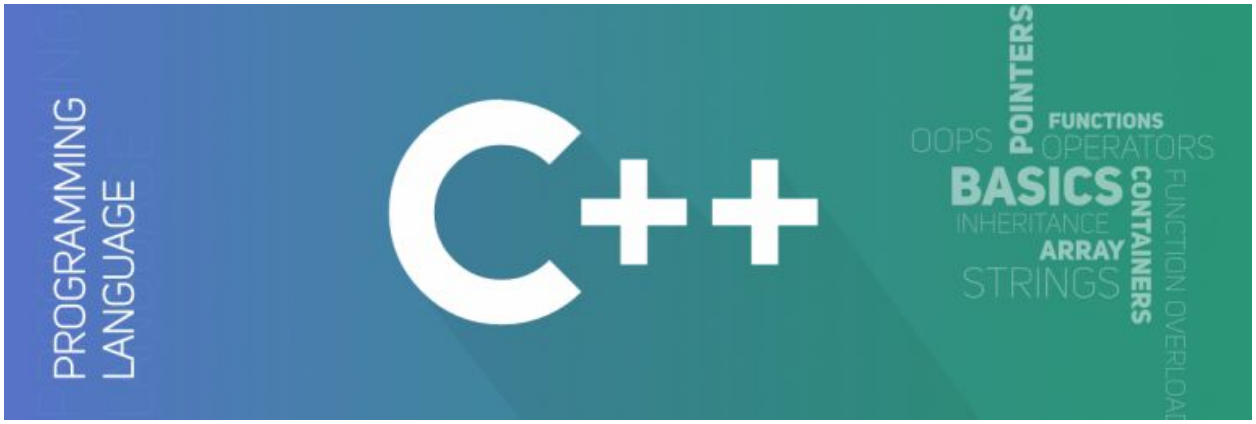


Case study : OOPs C++

“Learn with Examples”

06/Jan/2021

Submitted to,

Manishankar sir

ASAS, Mysore

Submitted by,

Arjun. S. Pramod

MY.SC.I5MCA19058

Int.MCA, ASAS, Mysore

Overview

This case study is mainly focused on my projects which performs few functionalities of different OOP concepts in C++. This project is in Informative type with examples.

Description

In this case study, we're able to see how different OOP concepts work. There are 7 parts in this program, such as:

- Operator overloading
- Usage of functions and its types:

  -> Inline function

  -> Call by reference

  -> Inline functions

  -> Default arguments

  -> Functions overloading

- Pointers
- Exception handling
- Inheritance
- Constructors
- Objects as member function

Specifications

I'm gonna use some of the main OOP concepts, such as:

- Classes and Objects
- Functions
- Inheritance
- Polymorphism

Explanation

Polymorphism : The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. A real-life example of polymorphism, a person at the same time can have different characteristics. Like 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 polymorphism. Polymorphism is considered as one of the important features of Object Oriented Programming.

Inheritance: The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important feature of Object Oriented Programming. Sub Class: The class that inherits properties from another class is called Sub-class or Derived Class. Super Class:The class whose properties are inherited by sub-class is called Base Class or Super class.

Templates: A template is a simple and yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don't need to write the same code for different data types. For example, a software company may need sort() for different data types. Rather than writing and maintaining the multiple codes, we can write one sort() and pass data type as a parameter.

Pointers: Pointers are symbolic representations of addresses. They enable programs to simulate call-by-reference as well as to create and manipulate dynamic data structures. It's general declaration in C/C++ has the format: Syntax: datatype *var_name; int *ptr; //ptr can point to an address which holds int data

Objects and Classes: A class is a blueprint for the object. A class is defined in C++ using keyword `class` followed by the name of the class. When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, we need to create objects.

Operator overloading: In C++, we can make operators to work for user defined classes. This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading.For example, we can overload an operator '+' in a class like String so that we can concatenate two strings by just using +. Other example classes where arithmetic operators may be overloaded are Complex Number, Fractional Number, Big Integer, etc.

Source code

```
#include<iostream>

using namespace std;

//Part 1: operator overloading mainly, class, functions are used here
class space
{
      int x;
      int y;
      int z;
   public:
      void getdata(int a, int b ,int c);
      void display(void);
      void operator -(); //operator function for overloaded operator unary minus(-)
```

```
};

void space::getdata(int a, int b,int c)
{   x=a;//x=10
    y=b;//y=-11
    z=c;//z=12
}

void space::display()
{
    cout<<"x= "<<x<<endl;
    cout<<"y= "<<y<<endl;
    cout<<"z= "<<z<<endl;
}
void space::operator -()
{   x=-x; //x=10 , -10
    y=-y; //y=-11, 11
    z=-z; //z=12, -12
}

//Part 2: Functions part: Inline, function overloading, call by reference, default arguments are used here

int swap(int &a1,int &b1) //call by reference
{
    int temp=a1;
    a1=b1;
    b1=temp;
    cout<<"After swapping: value of x =" <<a1 <<"\t value of y = "<<b1<<endl;
    return 0;
```

```
}
inline float cube(int a1) //inline function
{
   cout<< " cube of 3 = " << a1*a1*a1<<endl;
}

float si(int p, int t, float r=0.15) // default argument
{
   cout<<"Simple interest: "<< p*t*r<<endl;
}

int add(int a1, int b1) //function overloadng with example of 5 functions
{
   return a1 + b1;
}

int add(int a1, int b1, int c1)
{
   return a1 + b1 + c1;
}

double add(double a1, double b1)
{
   return a1 + b1;
}

double add(double a1, int b1)
{
   return a1 + b1;
```

```
}

double add(int a1, double b1)
{
   return a1 + b1;
}

//Part 3: Using function pointers, base pointers and etc...

typedef int (*funptr)(int,int);
int add2(int x2, int y2)
{
   int sum2 = x2+y2;
   cout<<sum2<<endl;
}

int sub2(int x2, int y2)
{
   int sub2 = x2-y2;
   cout<<sub2;
}
class A
{
   int a2;
   void read()
   {
      this->a2=10;
      cout<<a2<<endl;
   }
```

```
    public:
    void display()
    {
        cout<<a2<<endl; //this pointer isn't used, hence a value isn't fetched
        read(); //this pointer used through read()
    }
    void display1()
    {
        this->a2=20;
        cout<<a2<<endl;
    }
};

class Base
{
    public:
        void display(){cout<<"in base display"<<endl;};
        virtual void show(){cout<<"in base show"<<endl;}
};
class Derived : public Base
{
    public:
        void display(){cout<<"in derived display"<<endl;};
        void show(){cout<<"in derived show"<<endl;}
};

//Part 4: Exception handling
double division(int a3, int b3)
{
```

```
   if( b3 == 0 )
   {
      throw "Division by zero condition!";
   }
return (a3/b3);
}

// Part 5: Finding rectangular area using Inheritance concepts
class Shape
{
   public:
      void setWidth(int w)
      {
         width = w;
      }
      void setHeight(int h)
      {
         height = h;
      }
   protected:
      int width;
      int height;
};
   // Derived class
class Rectangle: public Shape
{
   public:
      int getArea()
      {
```

```
        return (width * height);
    }
};


//Part 6- Constructors: Multiple constructors

class integer
{
   int m,n;
   public:
      integer()
      {
         m=0; n=0; //constructor 1
         cout<<m<<" "<<n<<endl;;
      }
      integer (int a, int b)
      {
         m=a; n=b;
         cout<<m<<" "<<n<<endl;;
      } //constructor 2
      integer (integer &i)
      {
         m=i.m; n=i.n;
         cout<<m<<" "<<n<<endl;;
      } // constructor 3
};

//Part 7: Objects as member functions
```

```
class ABC;
class XYZ
{
   int x7;
   public:
      void setvalue(int i7)
      {
         x7=i7;
      }

      friend void max(XYZ,ABC);
};

class ABC{
   int a7;
   public:
      void setvalue(int i7)
      {
         a7=i7;
      }
      friend void max(XYZ,ABC);

};

void max(XYZ m7,ABC n7)
{
   if(m7.x7>=n7.a7)
   {
      cout<<"greatest value is "<<m7.x7<<endl;
```

```
    }
    else
    {
        cout<<"greatest value is "<<n7.a7<<endl;
    }

}



int main()
{

cout<<"_____
_____"<<endl;
    cout<<"\t\t\tWelcome to this case study on OOP concepts with C++"<<endl;
    cout<<"\tThe details of this program are mentioned in the decritption part of the case
study!!"<<endl<<endl;

cout<<"_____
_____"<<endl;
    //Part 1
    cout<<"\n\n\tFirst we're gonna see how Operator overloading works!"<<endl;
    cout<<"We've inserted the values for three variables as x, y, and z
respectively"<<endl;
    space s;
    s.getdata(10,-11,12);
    s.display();
    cout<<"\n\tThe overloading part is invoked using the operator we assigned, which
hyphen '-' in this case(as used)."<<endl<<"Thereby, the results are as follows:" <<endl;
    -s;
    cout<<"\n";
```

```
    s.display();


    cout<<"\t Now, we're gonna see how functions part works with different types, such as, call by reference, inline functions, and when default arguments are used..."<<endl;



    //Part 2
    int m,n,x,y=1;       //call by reference
    cout<<"Before swapping: value of x = 2\t value of y = 3"<<endl;
    int swap1 = swap( m=2, n=3); // call by ref
    cout<<"\nDone by inline function"<<endl;
    float b = cube(3); //inline function
    cout<<"\nDefault arguments used in here, i.e., rate of interest is set to 15% "<<endl;
    float s1 = si(1000, 20 ); //default arguments
    cout<<"\nFunction overloading is shown using 5 examples below: "<<endl;
    cout<< add(5,10)<<endl; //function calls for function overloading
    cout<< add(5,10,15)<<endl;
    cout<< add(7.5,2.5)<<endl;
    cout<< add(5,10.5)<<endl;
    cout<< add(5.5,10)<<endl;



    //Part 3
    cout<<"\nUsing function pointers: "<<endl;
    cout<<"We've given the value of two variables as 3 and 2, which on addition gives the value '5', '1' on subtraction"<<endl;
    funptr ptr;
    ptr = &add2;
    ptr(3,2);
    ptr = &sub2;
```

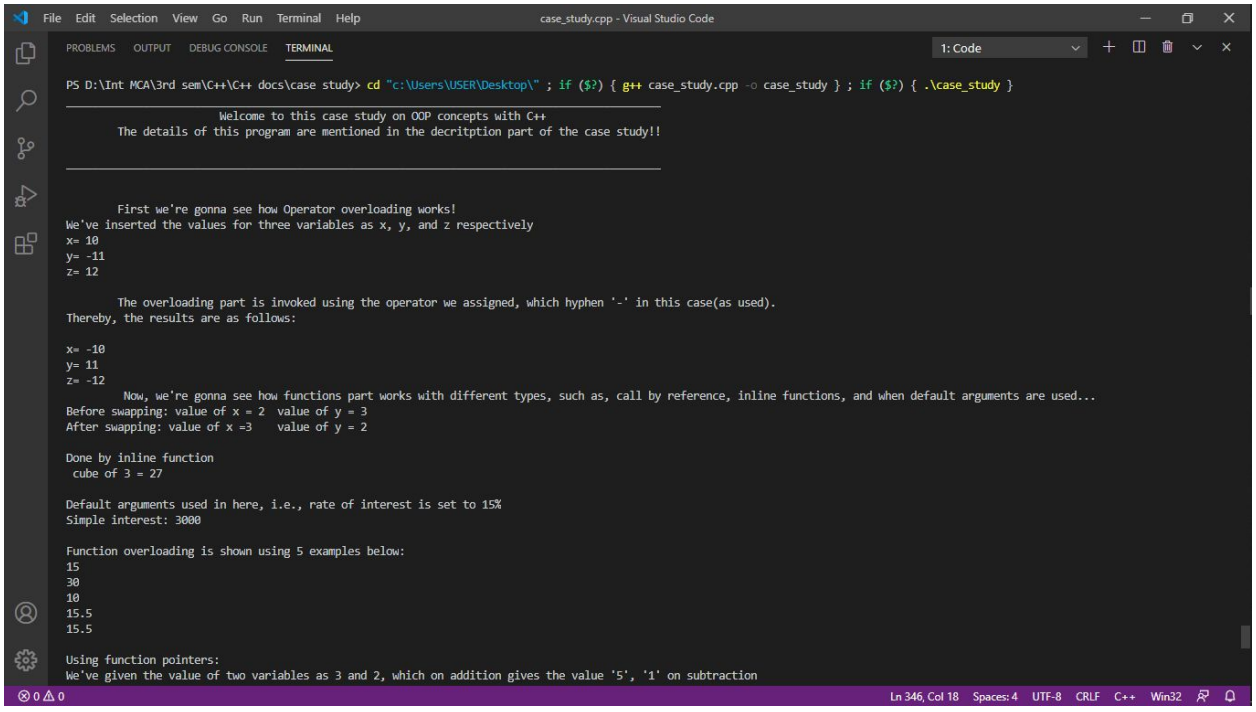
```
    ptr(3,2);
    cout<<endl;
    cout<<"\nHere, I've assigned 10 and 20 for two variables 'a' and 'b' respectively. And these variables are fetched using 'this pointer'! "<<endl;
    A aa;
    aa.display();
    aa.display1();
    cout<<"\n Here, base pointer is used to show differences like the way it works on it's usage!!\n";
    Base B;
    Derived D;
    Base *bptr;
    bptr = &B;
    bptr -> display();
    bptr -> show();
    bptr = &D;
    bptr -> display();
    bptr -> show();


    cout<<"\n\nException handling is used here, therefore message appeared when error is encountered!!"<<endl;


    //Part 4
    int x3 = 50;
    int y3 = 0;
    double z3 = 0;
    try
    {
       z3 = division(x3, y3);
       cout << z3 << endl;
```

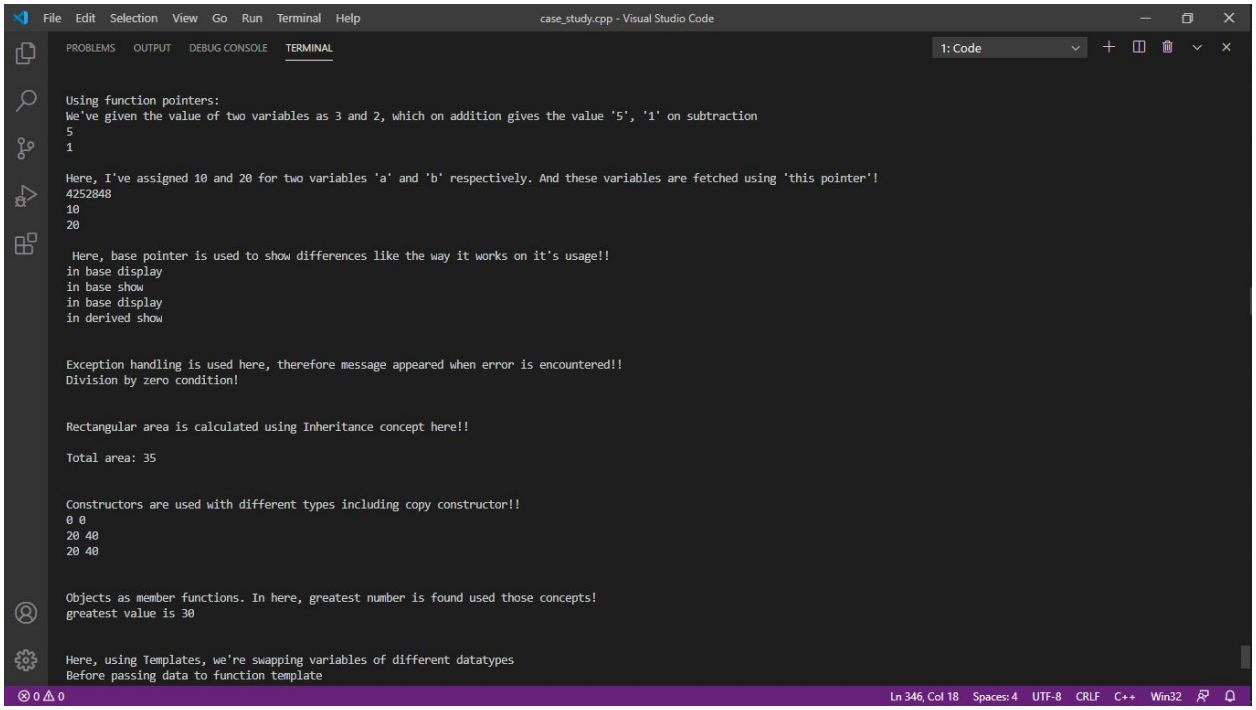
```
    } catch (const char* msg)
    {
        cerr << msg << endl;
    }

    cout<<"\n\nRectangular area is calculated using Inheritance concept here!!\n"<<endl;

    //Part 5
    Rectangle Rect;
    Rect.setWidth(5);//inherited member functions of base class shape
    Rect.setHeight(7);
    cout << "Total area: " << Rect.getArea() << endl;

    cout<<"\n\nConstructors are used with different types including copy
constructor!!"<<endl;
    //Part 6
    integer i1; //calls first constructor
    integer i2(20,40); //calls second constructor
    integer i3(i2); //calls third constructors

    cout<<"\n\nObjects as member functions. In here, greatest number is found used
those concepts!"<<endl;
    //Part 7
    XYZ d7;
    ABC b7;
    d7.setvalue(30);
    b7.setvalue(20);
    max(d7,b7);
```

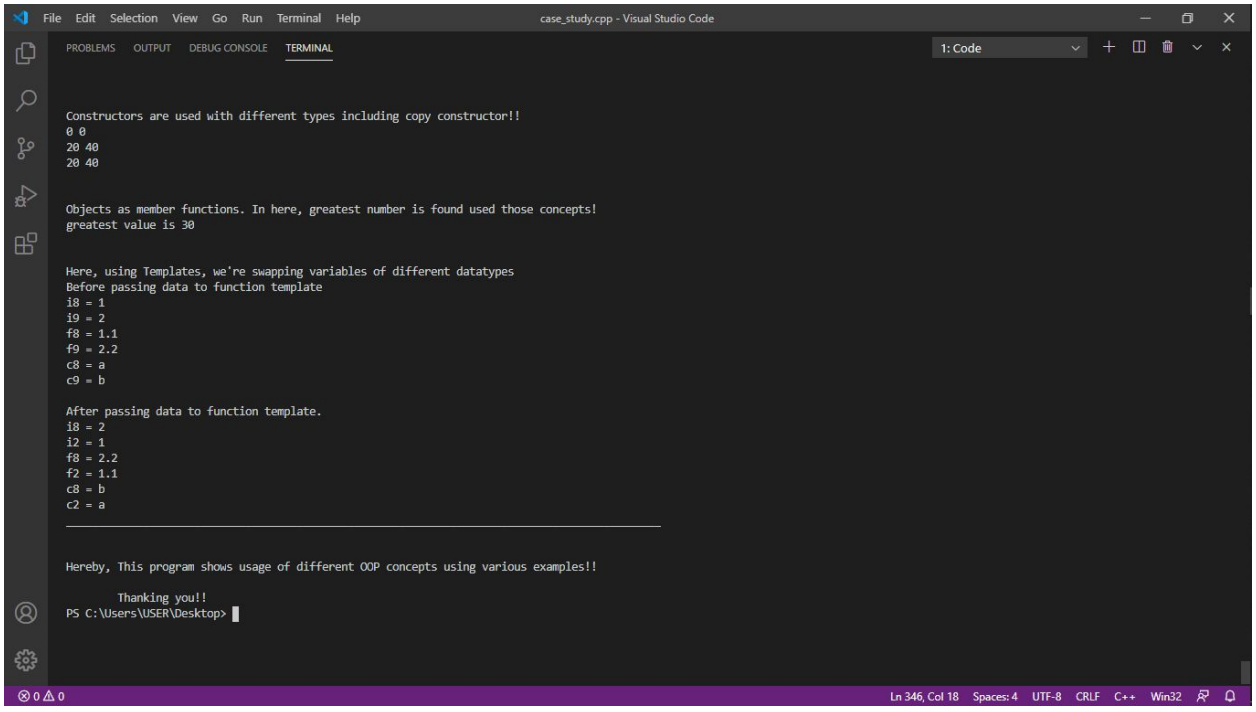
```
cout<<"\n_____
_____"<<endl;
   cout<<"\n\nHereby, This program shows usage of different OOP concepts using various examples!!"<<endl;
   cout<<"\n\tThanking you!!";

   return 0;
}
```

Output:

O







Hereby, I'm attaching my Google drive's link to view the codes too!

https://drive.google.com/file/d/1NCJWoeFwhm-lw-PiBH2lTzbECmjTSJ_G/view?usp=sharing