DHA SUFFA UNIVERSITY
Department of Computer Science
Object Oriented Programming
CS-1002L
Spring 2020

Lab 08

Polymorphism

Objective

To learn about polymorphism, virtual functions, virtual destructors,abstract classes and pure virtual functions and dynamic casting.

Polymorphism

Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance. C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Binding

Binding refers to the process of converting identifiers (such as variable and performance names) into addresses. Binding is done for each variable and functions. For functions, it means that matching the call with the right function definition by the compiler. It takes place either at compile time or at runtime.

Early Binding

Early Binding happens at compile time, function definition and function call are linked in compile time, when all the information needed to call a function is available at compile time.

Late Binding

Late Binding happens at runtime, function definition and function call are not resolved until run time, when all the information needed to call a function is not available at compile time. It can be achieved using virtual functions .

virtual functions

A virtual function is a special form of member function that is declared within a base class and redefined by a derived class.

Example 8.1: **Parent class pointing to child class without *virtual* keyword using early binding**

```
//FileName:Browser.h
#ifndef BROWSER_H
#define BROWSER_H
#include <string>

//base class of Browser
class Browser
{
      //member functions
      public:
          Browser();
          ~Browser();
          void openTab();

      //member variables
      protected:
          std::string version;
};
#endif
```

```
//FileName: Browser.cpp
#include "Browser.h"
#include <iostream>

//function definitions
Browser::Browser():version("v1.0.0"){}

Browser::~Browser(){}

void Browser::openTab() {
    std::cout << "Tab opened in Browser class" << std::endl;
}
```

```
//FileName:Firefox.h
#ifndef FIREFOX_H
#define FIREFOX_H
#include <string>
#include "Browser.h"
```

```
//base class of firefox
class Firefox : public Browser
{
      //member functions
      public:
            Firefox();
           ~Firefox();
           void openTab();
};
#endif
```

```
//FileName:FireFox.cpp
#include "Firefox.h"
#include <iostream>

Firefox::Firefox(){}

Firefox::~Firefox(){}

void Firefox::openTab() {
    std::cout << "New tab opened in Firefox" << std::endl;
}
```

```
//FileName:main.cpp
#include "Firefox.h"
#include "Browser.h"
#include <iostream>

int main() {

    // Parent class pointing to child class
    Browser* b = new Firefox();
    b->openTab();
    delete b;
}
```

Output

```
Tab opened in Browser class
```

Example 8.2: **Parent class pointing to child class with *virtual* keyword using late binding**

```
//FileName:Browser.h
#ifndef BROWSER_H
#define BROWSER_H
#include <string>

//base class of Browser
class Browser
{
      //member functions
      public:
            Browser();
            ~Browser();
            virtual void openTab();

      //member variables
      protected:
            std::string version;
};
#endif
```

```
//FileName:Browser.cpp

#include "Browser.h"
#include <iostream>

//function definitions
Browser::Browser():version("v1.0.0"){}

Browser::~Browser(){}

void Browser::openTab() {
    std::cout << "Tab opened in Browser class" << std::endl;
}
```

```
//FileName:Firefox.h

#ifndef FIREFOX_H
```

```
#define FIREFOX_H
#include <string>
#include "Browser.h"
//base class of firefox
class Firefox : public Browser
{
      //member functions
      public:
           Firefox();
           ~Firefox();
           void openTab();
};
#endif
```

```
// FileName:Firefox.cpp
#include "Firefox.h"
#include <iostream>

// function definitions
Firefox::Firefox(){}

Firefox::~Firefox(){}

// function override
void Firefox::openTab() {
    std::cout << "New tab opened in Firefox" << std::endl;
}
```

```
// FileName:main.cpp
#include "Firefox.h"
#include "Browser.h"
#include <iostream>
int main() {

    Firefox firefox;

    // parent class pointing to child class
    // using late binding
    Browser* b = &firefox;
    b->openTab();
 }
```

Output

```
New tab opened in Firefox
```

Example 8.3: **Example of late binding**

```
//FileName:Browser.h
#ifndef BROWSER_H
#define BROWSER_H
#include <string>

//base class of Browser
class Browser
{
      //member functions
      public:
            Browser();
            ~Browser();
            virtual void openTab();

      //member variables
      protected:
            std::string version;
};
#endif
```

```
//FileName:Browser.cpp

#include "Browser.h"
#include <iostream>

//function definitions
Browser::Browser():version("v1.0.0"){}

Browser::~Browser(){}

void Browser::openTab() {
    std::cout << "Tab opened in Browser class" << std::endl;
}
```

```
//FileName:Firefox.h
#ifndef FIREFOX_H
#define FIREFOX_H
#include <string>
#include "Browser.h"
//base class of firefox
class Firefox : public Browser
{
      //member functions
      public:
           Firefox();
           ~Firefox();
           void openTab();
};
#endif
```

```
// FileName: Firefox.cpp
#include "Firefox.h"
#include <iostream>

// function definitions
Firefox::Firefox(){}

Firefox::~Firefox(){}

// overriding parent class method
void Firefox::openTab() {
    std::cout << "New tab opened in Firefox" << std::endl;
}
```

```
//FileName:Chrome.h
#ifndef CHROME_H
#define CHROME_H
#include <string>
#include "Browser.h"

//base class of Chrome
class Chrome : public Browser
{
      //member functions
      public:
```

```
            Chrome();
            ~Chrome();
            void openTab();
};
#endif
```

```
#include "Chrome.h"
#include <iostream>

Chrome::Chrome(){}

Chrome::~Chrome(){}

void Chrome::openTab() {
    std::cout << "New tab opened in Chrome" << std::endl;
}
```

```
#include "Firefox.h"
#include "Chrome.h"
#include <iostream>
#include <string>

int main() {

    // parent pointing to child class
    Browser* b;
    char choice;

    std::cout << "Enter 'c' for chrome and 'f' for firefox tab: " <<
std::endl;
    std::cin >> choice;

    if(choice == 'c') {
       b = new Chrome();
    }
    else if(choice == 'f') {
       b = new Firefox();
    }
    else {
       std:: cout << "Invalid choice!!" << std::endl;
       b = NULL;
```

```
    }

    if(b != NULL) {
       b->openTab();
       delete b;
    }
}
```

Output

```
Enter 'c' for chrome and 'f' for firefox tab:
c
New tab opened in Chrome
```

virtual destructor

Deleting a derived class object using a pointer to a base class that has a non-virtual destructor, results in undefined behavior. To correct this situation, the base class should be defined with a virtual destructor.

Lab Task 8.1:

Add virtual destructor in Example 8.3 and print messages in the constructors and destructors of all classes to verify.

Pure virtual function and abstract classes

The virtual function that is only declared but not defined in the base class is called a pure virtual function. A function is made pure virtual by preceding its declaration with the keyword virtual and by post fixing it with = 0

An Abstract class is a class that has at least one pure virtual function. Abstract classes are the base class which cannot be instantiated.

Syntax

```
class className
{
   public:
   //Syntax of pure virtual function
```

```
   virtual ReturnType FunctionName()=0;
};
```

Example 8.4: **Example of pure virtual function and abstract class**

```
//FileName:Browser.h
#ifndef BROWSER_H
#define BROWSER_H
#include <string>

//base class of Browser
class Browser
{
      //member functions
      public:
      Browser();
      ~Browser();
      virtual void openTab()=0;

      //member variables
      protected:
           std::string version;
};
#endif
```

```
//FileName:Browser.cpp
#include "Browser.h"
#include <iostream>

//function definitions
Browser::Browser():version("v1.0.0"){}

Browser::~Browser(){}
```

```
//FileName:Firefox.h

#ifndef FIREFOX_H
#define FIREFOX_H
```

```
#include <string>
#include "Browser.h"
//base class of firefox
class Firefox : public Browser
{
      //member functions
      public:
            Firefox();
            ~Firefox();
            void openTab();
};
#endif
```

```
// FileName: Firefox.cpp
#include "Firefox.h"
#include <iostream>

// function definitions
Firefox::Firefox(){}

Firefox::~Firefox(){}

// overriding parent class method
void Firefox::openTab() {
    std::cout << "New tab opened in Firefox" << std::endl;
}
```

```
//FileName:Chrome.h
#ifndef CHROME_H
#define CHROME_H
#include <string>
#include "Browser.h"

//base class of Chrome
class Chrome : public Browser
{
      //member functions
      public:
            Chrome();
            ~Chrome();
            void openTab();
```

```
};
#endif
```

```
#include "Chrome.h"
#include <iostream>

Chrome::Chrome(){}

Chrome::~Chrome(){}

void Chrome::openTab() {
    std::cout << "New tab opened in Chrome" << std::endl;
}
```

```
#include "Firefox.h"
#include "Chrome.h"
#include <iostream>
#include <string>

int main() {


    // Creating object of abstract class will result in error
    // Browser b; -> error

    // Declaring Parent class pointer
    Browser* b;
    char choice;

    std::cout << "Enter 'c' for chrome and 'f' for firefox tab: " <<
std::endl;
    std::cin >> choice;

    if(choice == 'c') {
       b = new Chrome();
    }
    else if(choice == 'f') {
       b = new Firefox();
    }
    else {
       std:: cout << "Invalid choice!!" << std::endl;
```

```
        b = NULL;
    }

    if(b != NULL) {
        b->openTab();
        delete b;
    }
}
```

Output

```
Enter 'c' for chrome and 'f' for firefox tab:
c
New tab opened in Chrome
```

Lab Tasks

Task 8.2

Create a class named Polygon with member variables width and height of type int. This class should also contain member function setValues(int w,int h)to set values of width and height.

Now derive a child class triangle using public inheritance, with member function calculateArea() only.

In main program, for class triangle set the values of width and height using base class pointer.

Calculate area of triangle and display on screen.

Task 8.3

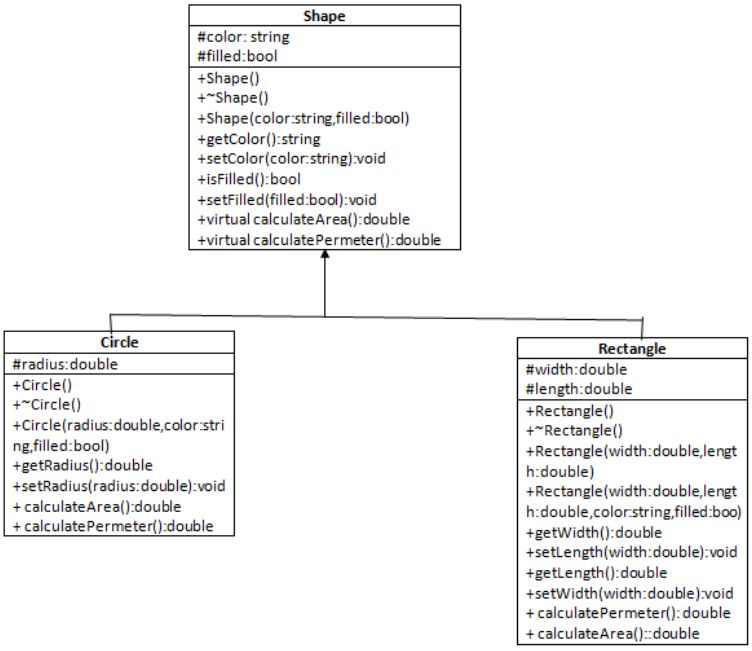
Considering the same scenario as in task 8.2,

Add a member function in class Triangle named void showAngles() which must be a derived class function only.

Call this function using a base class pointer. How will you implement this? Write your code to implement this scenario.

Home Assignment 8.4

Implement the following diagram in c++



Submission Instructions

1.Number your solution folders as question number e.g. Q1, Q2, etc. (Q is in upper case)
2.Every folder should contain three files (one header, one implementation and one driver)
3.Create a new folder named cs152abc where abc is your 3 digit roll #. e.g. cs152111.
4.Copy all the project folders into this folder.
5.Now make sure a zip file named cs152abc.zip is created e.g. cs152111.zip