6. Payment

Define a class named `Payment` that contains a member variable of type `float` that stores the amount of the payment and appropriate accessor and mutator methods. Also create a member function named `paymentDetails` that outputs an English sentence that describes the amount of the payment.

Next define a class named `CashPayment` that is derived from `Payment`. This class should redefine the `paymentDetails` function to indicate that the payment is in cash. Include appropriate constructor(s).

Define a class named `CreditCardPayment` that is derived from `Payment`. This class should contain member variables for the name on the card, expiration date, and credit card number. Include appropriate constructor(s). Finally, redefine the `paymentDetails` function to include all credit card information in the printout.

Create a main method that creates at least two `CashPayment` and two `CreditCardPayment` objects with different values and calls `paymentDetails` for each.

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

// Base class to store a payment amount and get a description
class Payment
{
public:
    Payment();
    Payment(float amount);
    void setPayment(float amount);
    float getPayment();
    void paymentDetails();
private:
    float amount;
};
```
// Constructor to initialize amount to 0
// ======================
Payment::Payment() : amount(0) {
}

// Constructor to initialize default amount
// ======================
Payment::Payment(float amt) : amount(amt) {
}

// payment accessor/mutator methods
// ======================
void Payment::setPayment(float amt) {
    amount = amt;
}
float Payment::getPayment() {
    return amount;
}

// paymentDetails
// Outputs the payment information in English
// ======================
void Payment::paymentDetails() {
    cout << "The payment amount is " << amount << endl;
}

// First define class for CashPayment, derive from Payment

class CashPayment : public Payment {
public:
    CashPayment();
    CashPayment(float amt);
    void paymentDetails();
};

// CashPayment constructors,
// just invokes base constructor.
CashPayment::CashPayment() : Payment() {

CashPayment::CashPayment(float amt) : Payment(amt)
{
}

// ======================
// paymentDetails
// Outputs the payment information in English
// ======================
void CashPayment::paymentDetails()
{
    cout << "The cash payment amount is " << getPayment() << endl;
}

// Next define class for CreditCardPayment, derive from Payment.
// This could be made more flexible with mutator and accessor
// methods for the name, expiration, and creditcard variables.

class CreditCardPayment : public Payment
{
public:
    CreditCardPayment();
    CreditCardPayment(float amt, string name, string expiration,
                       string creditcard);
    void paymentDetails();
private:
    string name;
    string expiration;
    string creditcard;
};

// ======================
// CreditCardPayment constructors,
// just invokes base constructor and initialize variables.
// ======================
CreditCardPayment::CreditCardPayment() : Payment()
{
    name = "";
    expiration = "";
    creditcard = "";
}

CreditCardPayment::CreditCardPayment(float amt, string name, string expiration,
                                       string creditcard) : Payment(amt)
{
    this->name = name;
    this->expiration = expiration;
    this->creditcard = creditcard;
}

// ======================
// paymentDetails
// Outputs the payment information in English
// ======================
void CreditCardPayment::paymentDetails()
{
    cout << "The credit card payment amount is " << getPayment() << "
    endl;
    cout << "The name on the card is: " << name << "
    endl;
    cout << "The expiration date is: " << expiration << "
    endl;
    cout << "The credit card number is: " << creditcard << "
    endl;
}

// ----------------------------------
// END USER CODE
// ----------------------------------

// =====================
// main function
// =====================
int main()
{
    // Create several test classes and invoke the paymentDetails method
    CashPayment cash1(50.5), cash2(20.45);
    CreditCardPayment credit1(10.5, "Fred", "10/5/2010", "123456789");
    CreditCardPayment credit2(100, "Barney", "11/15/2009", "987654321");

    cout << "Cash 1 details:" << "
    cash1.paymentDetails();
    cout << "
    cout << "Cash 2 details:" << "
    cash2.paymentDetails();
    cout << "

    cout << "Credit 1 details:" << "
    credit1.paymentDetails();
    cout << "
    cout << "Credit 2 details:" << "
    credit2.paymentDetails();
    cout << "

    return 0;
}

10. Pet Inheritance

Define a Pet class that stores the pet’s name, age, and weight. Add appropriate constructors, accessor, and mutator functions. Also define a function named getLifespan that returns a string with the value “unknown lifespan.”

#include <iostream>
#include <string>
using namespace std;
Soluition for HW6 CSCE240 Fall 2011

/////////////////////////
// Pet Class
/////////////////////////
class Pet
{
public:
    Pet();
    Pet(string newName, int newAge, int newWeight);
    void setName(string newName);
    string getName();
    void setAge(int newAge);
    int getAge();
    void setWeight(int newWeight);
    int getWeight();
    string getLifespan();

private:
    string name;
    int age, weight;
};

Pet::Pet() : age(0), weight(0), name("")
{
}

Pet::Pet(string newName, int newAge, int newWeight) :
    age(newAge), weight(newWeight), name(newName)
{
}

void Pet::setName(string newName)
{
    name = newName;
}

string Pet::getName()
{
    return name;
}

void Pet::setAge(int newAge)
{
    age = newAge;
}

int Pet::getAge()
{
    return age;
}

void Pet::setWeight(int newWeight)
{
    weight = newWeight;
}

int Pet::getWeight()
{
    return weight;
}
string Pet::getLifespan()
{
    return "unknown lifespan";
}

/////////////////////////////////////////////////////////////////////////////
// Dog Class
/////////////////////////////////////////////////////////////////////////////

class Dog : public Pet
{
public:
    Dog();
    Dog(string newName, int newAge, int newWeight, string newBreed);
    void setBreed(string newBreed);
    string getBreed();
    string getLifespan();
private:
    string breed;
};

Dog::Dog() : Pet(), breed("")
{
}

Dog::Dog(string newName, int newAge, int newWeight, string newBreed) :
    Pet(newName, newAge, newWeight), breed(newBreed)
{
}

void Dog::setBreed(string newBreed)
{
    breed = newBreed;
}

string Dog::getBreed()
{
    return breed;
}

string Dog::getLifespan()
{
    if (this->getWeight() <= 100)
        return "Approximately 13 years";
    else
        return "Approximately 7 years";
}

/////////////////////////////////////////////////////////////////////////////
// Rock Class
/////////////////////////////////////////////////////////////////////////////

class Rock : public Pet
{
public:
    Rock();
    Rock(string newName, int newAge, int newWeight);
}
string getLifespan();

Rock::Rock(string newName, int newAge, int newWeight) :
    Pet(newName, newAge, newWeight)
{
}

Rock::Rock() : Pet()
{
}

string Rock::getLifespan()
{
    return "Thousands of years";
}

////////////////////////////////////
// Main
////////////////////////////////////
int main()
{
    Dog fido("Fido", 5, 120, "Great Dane");
    Dog pippen("Pippen", 2, 45, "Dalmatian");
    Rock rocco("Rocco", 1000, 1);

    cout << fido.getName() << ", age " << fido.getAge() << ", width " << fido.getWeight() << ", breed " << fido.getBreed() << endl;
    cout << fido.getLifespan() << endl << endl;
    cout << pippen.getName() << ", age " << pippen.getAge() << ", weight " << pippen.getWeight() << ", breed " << pippen.getBreed() << endl;
    cout << pippen.getLifespan() << endl << endl;
    cout << rocco.getName() << ", age " << rocco.getAge() << ", weight " << rocco.getWeight() << endl;
    cout << rocco.getLifespan() << endl << endl;

    cout << "Enter a character to exit." << endl;
    char wait;
    cin >> wait;
    return 0;
}

Chapter 15 (2)

This is the Chapter 15-1, which is the basis of problem 15-2

1. Figures

Implement a graphics System that has classes for various figures: rectangles, squares, triangles, circles, etc. A rectangle has data members height, width, and center point. A square has center
point and an edge. A circle has data members center and radius. The several specific shape classes are derived from a common class, Figure.

Implement only Rectangle and Triangle classes. Derive them from Figure.

Each class has stubs for erase and draw. The class Figure has member function center that calls erase and draw. The functions erase and draw are only stubs, consequently center does little but call erase and draw. Add a message to center announcing that it is being called.

There are 3 parts:

a) Write the class definitions using no virtual functions. Compile and test.

b) Make the base class member functions virtual. Compile and test.

c) Explain the differences.

Use the following main function:

```cpp
#include <iostream>
#include "figure.h"
#include "rectangle.h"
#include "triangle.h"
using std::cout;
int main()
{
    Triangle tri;
    tri.draw();
    cout << "\nDerived class Triangle object calling"
        << " center().\n";
    tri.center();
    Rectangle rect;
    rect.draw();
    cout << "\nDerived class Rectangle object calling"
        << " center().\n";
    rect.center();
    return 0;
}
```

Solution: No member variables are used as they serve no function in this part of the exercise.

//figure.h
#include <iostream>
using namespace std;

#ifndef PROTECT Figure_H
#define PROTECT Figure_H

class Figure
{

public:
```
virtual void erase();
virtual void draw();

void center()
{
  cout << "member function center called " << endl;
  erase();
  draw();
}
};

void Figure::erase()
{
  cout << "Figure::erase()" << endl;
}

void Figure::draw()
{
  cout << "Figure::draw()" << endl;
}
#endif

//rectangle.h
#include <iostream>
#include "figure.h"

using namespace std;

class Rectangle : public Figure
{
public:

  void erase();
  void draw();
};

void Rectangle::erase()
{
  cout << "Rectangle::erase()" << endl;
}

void Rectangle::draw()
{
  cout << "Rectangle::draw()" << endl;
}

//triangle.h
```cpp
#include <iostream>
#include "figure.h"
using namespace std;

class Triangle : public Figure
{
public:
    void draw();
    void erase();
};

void Triangle::draw()
{
    cout << "Triangle::draw()" << endl;
}

void Triangle::erase()
{
    cout << "Triangle::erase()" << endl;
}

//ch15p1App.cpp
#include <iostream>
#include "figure.h"
#include "rectangle.h"
#include "triangle.h"
using std::cout;

int main()
{
    Triangle tri;
    tri.draw();
    cout << "\nIn main, Derived class Triangle object calling" << " center().\n";
    tri.center();
    Rectangle rect;
    rect.draw();
    cout << "\nIn main, Derived class Rectangle object calling" << " center().\n";
    rect.center();
    return 0;
}

A trial run follows:
-------------------------------
With Virtual erase and draw
-------------------------------
Triangle::draw()
```
In main, Derived class Triangle object calling center().
member function center called
Triangle::erase()
Triangle::draw()

In main, Derived class Rectangle object calling center().
member function center called
Rectangle::erase()
Rectangle::draw()

In main, Derived class Triangle object calling center().
member function center called
Figure::erase()
Figure::draw()
Rectangle::draw()

In main, Derived class Rectangle object calling center().
member function center called
Figure::erase()
Figure::draw()

-------------------------------
Without Virtual erase and draw
-------------------------------

Explanation of difference
The differences are due solely to the presence or absence of the keyword virtual in the base class figure.

Listed below is code to play a guessing game in which two players attempt to guess a number. Your task is to extend the program with objects that represent either a human player or a computer player.

```cpp
bool checkForWin(int guess, int answer)
{
    if (answer == guess)
    {
        cout << "You're right! You win!" << endl;
        return true;
    }
    else if (answer < guess)
        cout << "Your guess is too high." << endl;
    else
        cout << "Your guess is too low." << endl;
    return false;
}
```
void play(Player &player1, Player &player2)
{
    int answer = 0, guess = 0;
    answer = rand() % 100;
    bool win = false;

    while (!win)
    {
        cout << "Player 1's turn to guess." << endl;
        guess = player1.getGuess();
        win = checkForWin(guess, answer);
        if (win) return;

        cout << "Player 2's turn to guess." << endl;
        guess = player2.getGuess();
        win = checkForWin(guess, answer);
    }
}

The play function takes as input two Player objects. Define the Player class with a virtual function named getGuess(). The implementation of Player::getGuess() can simply return 0. Next, define a class named HumanPlayer derived from Player. The implementation of HumanPlayer::getGuess() should prompt the user to enter a number and return the value entered from the keyboard. Next, define a class named ComputerPlayer derived from Player. The implementation of ComputerPlayer::getGuess() should randomly select a number from 0 to 100. Finally, construct a main function that invokes play(Player &player1, Player &player2) with two instances of a HumanPlayer (human vs. human), an instance of a HumanPlayer and ComputerPlayer (human vs. computer), and two instances of ComputerPlayer (computer vs. computer).


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

/////////////////////////////////////////////////////////////////
// Player Class
/////////////////////////////////////////////////////////////////
class Player
{
public:
    virtual int getGuess() = 0;
};

/////////////////////////////////////////////////////////////////
// ComputerPlayer Class
/////////////////////////////////////////////////////////////////
class ComputerPlayer : public Player
{
public:
    ComputerPlayer();
    int getGuess();
};

ComputerPlayer::ComputerPlayer() : Player()
{
}

int ComputerPlayer::getGuess()
{
    int guess = rand() % 100;
    cout << "The computer guesses " << guess << endl;
    return guess;
}

/////////////////////////////////////////////////////////////////
// HumanPlayer Class
/////////////////////////////////////////////////////////////////
```
class HumanPlayer : public Player
{
public:
    HumanPlayer();
    int getGuess();
};

HumanPlayer::HumanPlayer() : Player()
{
}

int HumanPlayer::getGuess()
{
    int guess;
    cin >> guess;
    return guess;
}

////////////////////////////////////////////////////////////////////////////////
// Global functions given in project
////////////////////////////////////////////////////////////////////////////////

bool checkForWin(int guess, int answer)
{
    if (answer == guess)
    {
        cout << "You're right! You win!" << endl;
        return true;
    }
    else if (answer < guess)
    {
        cout << "Your guess is too high." << endl;
    }
    else
    {
        cout << "Your guess is too low." << endl;
        return false;
    }
}

void play(Player &player1, Player &player2)
Solution for HW6 CSCE240 Fall 2011

```cpp
{
    int answer = 0, guess = 0;
    answer = rand() % 100;
    bool win = false;

    while (!win)
    {
        cout << "Player 1's turn to guess." << endl;
        guess = player1.getGuess();
        win = checkForWin(guess, answer);
        if (win) return;

        cout << "Player 2's turn to guess." << endl;
        guess = player2.getGuess();
        win = checkForWin(guess, answer);
    }
}

////////////////////////////////////////////////////
// Main
////////////////////////////////////////////////////
int main()
{
    HumanPlayer player1; // Can change to ComputerPlayer
    ComputerPlayer player2; // Can change to HumanPlayer

    play(player1, player2);

    cout << "Enter a character to exit." << endl;
    char wait;
    cin >> wait;
    return 0;
}
```