4. Create a base class called ***Student***that has their university's name(type ***string***),their registration number(type***int***), and their proctor (type ***UniversityStaff***given in the code that follows). Then create a class called ***ScienceStudent*** that is derived from ***Student***and has additional properties science discipline(type ***string***), undergraduate or postgraduate course (type ***string***). Be sure your classes have a reasonable complement of constructorsand accessor methods, an overloaded assignment operator, and a copy constructor. Write a driver program that tests all your methods.
   The definition of the classUniversityStafffollows. The implementation of the class is part of this programming project.

```
ClassUniversityStaff
{
public:
UniversityStaff();
UniversityStaff(string theName);
UniversityStaff(constUniversityStaff& theObject);
string getName()const;
UniversityStaff& operator=(constUniversityStaff& rtSide);
friendistream& operator >> (istream& inStream,
UniversityStaff& staffObject);
friendostream&operator << (ostream& outStream,
const UniversityStaff& staffObject);
private:
string name;
};
```

6. 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 functions.
   Also create a member function named paymentDetails that outputs an English sentence describing 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 function that creates at least two CashPayment and two CreditCardPayment objects with different values and calls to paymentDetails for each.

8. Create a class for a simple blog. The owner of the blog should be able to (a) post a new message,(b) numerically list and display all messages, and(c)select a specific message and delete it.
Viewers of the blog should only be able to numerically list and display all posted messages.
Create a class Viewer and a class Owner that uses inheritance to help implement the blog functionality. Store the data messages using any format you like (a vector of type string may be easiest). A menu function should be implemented for each class that outputs the legal choices for the type of user. To test your classes, the main function of the program should allow the user to invoke the menus from the Viewer and Owner objects.

CH15

5. The following shows 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.

```
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;

        cour <<"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 name **getGuess()**. The implementation of **Player::getGuess()** can simply return 0. Next, define a class name **HumanPlayer** derived from **Player**. The implementation of **HumanPlayer::getGuess()** should prompt the user to enter a number and return the valve 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 versus human), as instance of a **HumanPlayer** and **ComputerPlayer** (human versus computer), and two instances of **ComputerPlayer** (computer versus computer).

6. The computer player in Programming Project 15.5 does not play the number guessing game very well, since it makes only random guesses. Modify the program so that the computer plays a more informed game. The specific strategy is up to you, but you must add function(s) to the **Player** and **ComputerPlayer** classes so that the **play(Player &player1, Player &player2)** function can send the results of a guess back to the computer player. In other words, the computer must be told if its last guess was too high or too low, and it also must be told if its opponent's last guess was too high or too low. The computer can then use this information to revise its next guess.