**CS2400-60 Computer Science II**

**Programming assignment #11**

**Inheritance & polymorphism**

**Michael Greco**

**P11.1** The class *derivedCL* inherits base class *baseCL*, using public inheritance. Both classes have a version of the function f(). for simplicity, all functions are implemented inline (i.e., code for all member functions have been provided). The interfaces of two classes are provided below:

class baseCL

{

public:

baseCL(int x = 1) : baseX(x), baseY(4)

{}

int f()

{ return baseX + baseY;}

private:

int baseX;

protected:

int baseY;

};

class derivedCL : public baseCL

{

public:

derivedCL() : x(10), y(11), baseCL(5)

{}

int f()

{

y = baseCL :: baseY \* 2;

x = 3;

return x \* y;

}

private:

int x, y;

};

Assume the given declarations for objects *base1Obj*, *base2Obj*, and *derivedObj* in the *main* function with *base1Obj* initialized to 2.

1. Predict the output for the following statements. If a statement is not valid, indicates that as the output.

cout << base1Obj.f() << endl; // output = 6

cout << base2Obj.f() << endl; // output = 5

cout << derivedObj.f() << endl; // output = 24

cout << derivedObj.baseCL::f() << endl; // output = 9

base2Obj = derivedObj;

cout << base2Obj.f() << endl << endl; // output = 9

cout << base2Obj.derivedCL.f() << endl; // output = Not Valid

1. Removing **protected** keyword from baseCL class will run the program into compile error. It trys to access the variable y of baseCL class which is private. Private variables or methods of the class can not be accessed outside the definition of the class. Accessing the private variables of the class will run the program into compile time error of accessing the private variables which are inaccessible.
2. If the public keyword in derivedCL class definition is replaced with either private or protected, then the program will run into compile time error as it will try to access the baseCL class methods and variables through the object of derivedCL class which has been inherited by it in either private scope or protected scope. So, inheriting the baseCL class by derivedCL class in either private scope or protected scope will run the program into compile time error.

**P11.2** When ***XYZ*** Company started two years ago, the below payroll program was written to handle the payrolls of only *salaried* and *hourly* employees. As business expands, the company finds the need to hire sales people. For each pay period, a sales person‘s pay includes a base pay of $500.00 plus commission, which is five percent of the total sales generated by the sales person during each pay period. Since the original program is written in C++, an OOP language that supports inheritance, the existing program can be easily extended to meet the new business need for the company. Write the extended version by creating a new derived classes named ***salesEmployee*** thatcontains data members *basePay*, *totalSales,* and *commissionRate*. The newly added class also contains a member function that computes and displays the pay check. Write the *main()* function to generate the payrolls of all three types of employees. For your convenience, the code for the original program that handles only *salariedEmployee* and *hourlyEmployee* is provided below:

Header File

#ifndef HEADER\_H

#define HEADER\_H

#include <iostream>

#include<iomanip>

#include<string>

using namespace std;

class Employee

{

public:

Employee();

Employee(string the\_name, string the\_ssn);

string get\_name() const;

string get\_ssn() const;

double get\_net\_pay() const;

void set\_name(string new\_name);

void set\_ssn(string new\_ssn);

void set\_net\_pay(double new\_net\_pay);

void print\_check() const;

protected:

string name;

string ssn;

double net\_pay;

};

// Base Class implementation:

Employee::Employee() : name("No name yet"), ssn("No number yet"), net\_pay(0)

{ }

Employee::Employee(string the\_name, string the\_number)

: name(the\_name), ssn(the\_number), net\_pay(0)

{ }

string Employee::get\_name() const

{

return name;

}

string Employee::get\_ssn() const

{

return ssn;

}

double Employee::get\_net\_pay() const

{

return net\_pay;

}

void Employee::set\_name(string new\_name)

{

name = new\_name;

}

void Employee::set\_ssn(string new\_ssn)

{

ssn = new\_ssn;

}

void Employee::set\_net\_pay(double new\_net\_pay)

{

net\_pay = new\_net\_pay;

}

void Employee::print\_check() const

{

cout << "\nCannot print a check for an uncatagorized employee!" << endl;

}

// Derived Class: HourlyEmployee Interface

class HourlyEmployee : public Employee

{

public:

HourlyEmployee();

HourlyEmployee(string the\_name, string the\_ssn,

double the\_wage\_rate, double the\_hours);

void set\_rate(double new\_wage\_rate);

double get\_rate() const;

void set\_hours(double hours\_worked);

double get\_hours() const;

void print\_check();

private:

double wage\_rate;

double hours;

};

//Derived Class: HourlyEmployee implementation

HourlyEmployee::HourlyEmployee() : Employee(), wage\_rate(0), hours(0)

{ }

HourlyEmployee::HourlyEmployee(string the\_name, string the\_number, double the\_wage\_rate, double the\_hours)

: Employee(the\_name, the\_number), wage\_rate(the\_wage\_rate), hours(the\_hours)

{ }

void HourlyEmployee::set\_rate(double new\_wage\_rate)

{

wage\_rate = new\_wage\_rate;

}

double HourlyEmployee::get\_rate() const

{

return wage\_rate;

}

void HourlyEmployee::set\_hours(double hours\_worked)

{

hours = hours\_worked;

}

double HourlyEmployee::get\_hours() const

{

return hours;

}

void HourlyEmployee::print\_check()

{

set\_net\_pay(hours \* wage\_rate);

cout << fixed << setprecision(2);

cout << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "Pay to the order of " << name << endl;

cout << "The sum of " << net\_pay << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "Employee Number: " << ssn << endl;

cout << "Hourly Employee. \nHours worked: " << hours << endl

<< "Hourly rate: $" << wage\_rate << "\nPay: $" << net\_pay << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

}

// Derived Class: SalariedEmployee interface

class SalariedEmployee : public Employee

{

public:

SalariedEmployee();

SalariedEmployee(string the\_name, string the\_ssn, double the\_weekly\_salary);

double get\_salary() const;

void set\_salary(double new\_salary);

void print\_check();

private:

double salary; //weekly

};

// Derived Class: SalariedEmployeemp iementation

SalariedEmployee::SalariedEmployee() : Employee(), salary(0)

{ }

SalariedEmployee::SalariedEmployee(string the\_name, string the\_number, double the\_weekly\_salary)

: Employee(the\_name, the\_number), salary(the\_weekly\_salary)

{ }

double SalariedEmployee::get\_salary() const

{

return salary;

}

void SalariedEmployee::set\_salary(double new\_salary)

{

salary = new\_salary;

}

void SalariedEmployee::print\_check()

{

cout << fixed << setprecision(2);

set\_net\_pay(salary);

cout << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "Pay to the order of " << name << endl;

cout << "The sum of " << net\_pay << " Dollars\n";

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "Employee Number: " << ssn << endl;

cout << "Salaried Employee. \nRegular Pay: $" << salary << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

}

class SalesEmployee : public Employee

{

private:

double basePay, totalSales, commissionRate;

public:

SalesEmployee(string the\_name, string the\_number, double the\_basePay, double the\_totalSales, double the\_commissionRate)

: Employee(the\_name, the\_number)

{

basePay = the\_basePay;

totalSales = the\_totalSales;

commissionRate = the\_commissionRate;

}

void setBasePay(double newBasePay)

{

basePay = newBasePay;

}

void setTotalSales(double newTotalSales)

{

totalSales = newTotalSales;

}

void setCommissionRate(double newCommissionRate)

{

commissionRate = newCommissionRate;

}

double getBasePay()

{

return basePay;

}

double getTotalSales()

{

return totalSales;

}

double getCommissionRate()

{

return commissionRate;

}

void print\_check()

{

cout << fixed << setprecision(2);

cout << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "Name: $" << get\_name() << endl;

cout << "Base Pay " << basePay << endl;

cout << "Total Sales" << totalSales << " Dollars\n";

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "Employee Number: " << get\_ssn() << endl;

cout << "Sales Employee. \nRegular Pay: $" << get\_net\_pay() << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

}

};

#endif

Main File

#include"Header.h"

int main()

{

HourlyEmployee hourlyEmployee("Neehaal", "322AD3", 343, 32);

cout << "Hourly Employee:" << endl;

hourlyEmployee.print\_check();

SalariedEmployee salariedEmployee("Akshaya", "329GA88", 3392.33);

cout << "Salaried Employee:" << endl;

salariedEmployee.print\_check();

SalesEmployee salesEmployee("Madhavi", "32ADFF", 500, 200, 0.5);

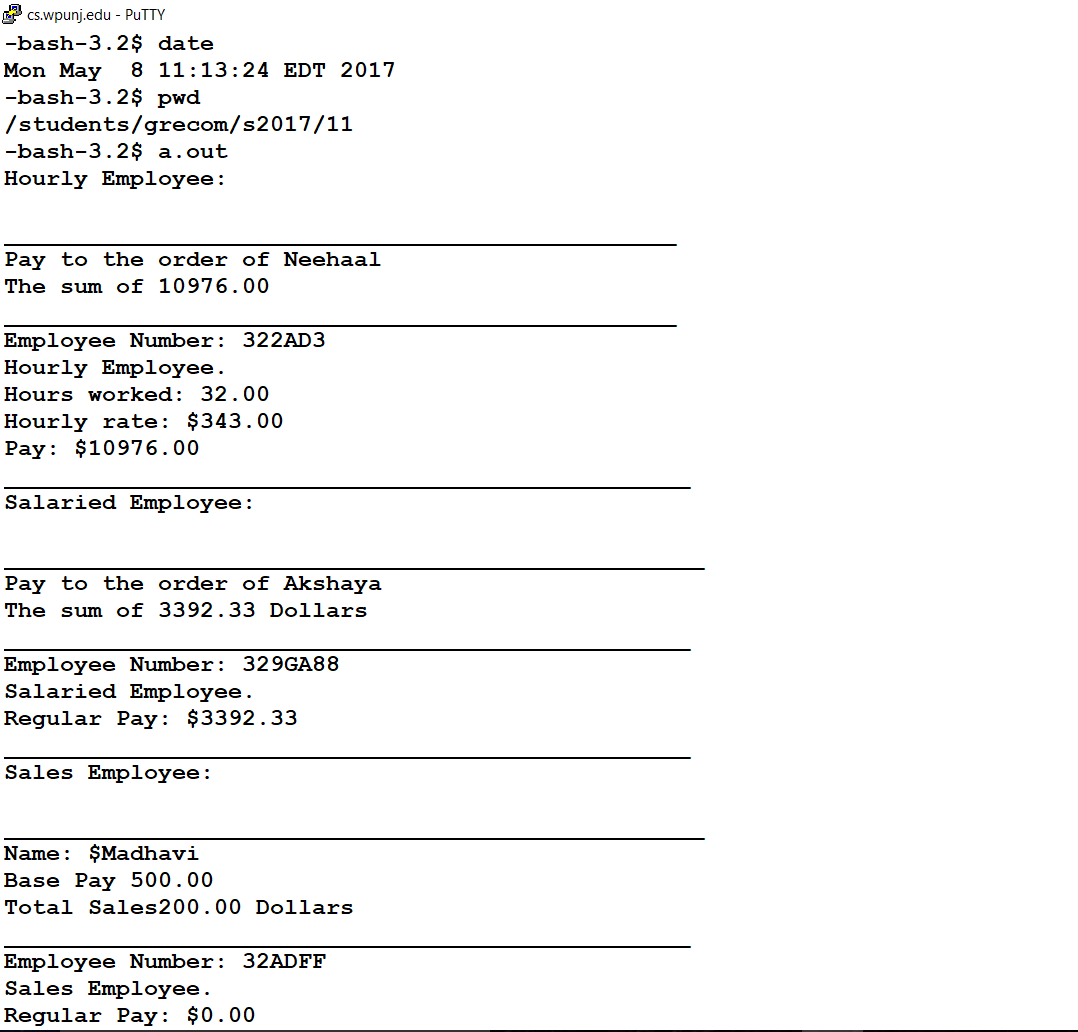
cout << "Sales Employee:" << endl;

salesEmployee.print\_check();

cout << "Hello world!" << endl;

return 0;

}



**P11.3** Modify the classes defined in P11.2 such that they support polymorphism. Write a *main()* function that defines three objects representing three types of employees and use polymorphic calls to the overriding ***print\_check()*** function to see if indeed the same call to the *print\_check()* function does produce different results! Note that you may not get the code to behave polymorphically on your first try because there are some subtlety about the coding for the *print\_check()* functions in the above code. I thought it might be of interest to you to find out.

Header File 1

#ifndef SALARY\_EMPLOYEE\_H

#define SALARY\_EMPLOYEE\_H

#include<iostream>

#include<string>

//include Employee header file

#include "Employee.h"

using namespace std;

//deriveed from Employee class

class SalariedEmployee : public Employee

{

public:

SalariedEmployee();

SalariedEmployee(string the\_name, string the\_ssn, double the\_weekly\_salary);

double get\_salary() const;

void set\_salary(double new\_salary);

void print\_check();

private:

double salary; //weekly

};

// Derived Class: SalariedEmployeemp iementation

SalariedEmployee::SalariedEmployee() : Employee(), salary(0)

{ }

SalariedEmployee::SalariedEmployee(string the\_name, string the\_number, double the\_weekly\_salary)

: Employee(the\_name, the\_number), salary(the\_weekly\_salary)

{ }

double SalariedEmployee::get\_salary() const

{

return salary;

}

void SalariedEmployee::set\_salary(double new\_salary)

{

salary = new\_salary;

}

void SalariedEmployee::print\_check()

{

cout << fixed << setprecision(2);

set\_net\_pay(salary);

cout << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "Pay to the order of " << name << endl;

cout << "The sum of " << net\_pay << " Dollars\n";

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "Employee Number: " << ssn << endl;

cout << "Salaried Employee. \nRegular Pay: $" << salary << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

}

#endif SALARY\_EMPLOYEE\_H

Header File 2

#ifndef HOURLY\_EMPLOYEE\_H

#define HOURLY\_EMPLOYEE\_H

// Derived Class: HourlyEmployee Interface

#include<iostream>

#include<string>

#include<iomanip>

//include Employee.h headaer file

#include "Employee.h"

using namespace std;

class HourlyEmployee : public Employee

{

public:

HourlyEmployee();

HourlyEmployee(string the\_name, string the\_ssn,

double the\_wage\_rate, double the\_hours);

void set\_rate(double new\_wage\_rate);

double get\_rate() const;

void set\_hours(double hours\_worked);

double get\_hours() const;

void print\_check();

private:

double wage\_rate;

double hours;

};

//Derived Class: HourlyEmployee implementation

HourlyEmployee::HourlyEmployee() : Employee(), wage\_rate(0), hours(0)

{ }

HourlyEmployee::HourlyEmployee(string the\_name, string the\_number, double the\_wage\_rate, double the\_hours)

: Employee(the\_name, the\_number), wage\_rate(the\_wage\_rate), hours(the\_hours)

{ }

void HourlyEmployee::set\_rate(double new\_wage\_rate)

{

wage\_rate = new\_wage\_rate;

}

double HourlyEmployee::get\_rate() const

{

return wage\_rate;

}

void HourlyEmployee::set\_hours(double hours\_worked)

{

hours = hours\_worked;

}

double HourlyEmployee::get\_hours() const

{

return hours;

}

void HourlyEmployee::print\_check()

{

set\_net\_pay(hours \* wage\_rate);

cout << fixed << setprecision(2);

cout << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "Pay to the order of " << name << endl;

cout << "The sum of " << net\_pay << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "Employee Number: " << ssn << endl;

cout << "Hourly Employee. \nHours worked: " << hours << endl

<< "Hourly rate: $" << wage\_rate << "\nPay: $" << net\_pay << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

}

#endif HOURLY\_EMPLOYEE\_H

Header File 3

#ifndef EMPLOYEE\_H

#define EMPLOYEE\_H

// Base class interface

#include<iostream>

#include<string>

using namespace std;

class Employee

{

public:

Employee();

Employee(string the\_name, string the\_ssn);

string get\_name() const;

string get\_ssn() const;

double get\_net\_pay() const;

void set\_name(string new\_name);

void set\_ssn(string new\_ssn);

void set\_net\_pay(double new\_net\_pay);

void print\_check() const;

protected:

string name;

string ssn;

double net\_pay;

};

// Base Class implementation:

Employee::Employee() : name("No name yet"), ssn("No number yet"), net\_pay(0)

{ }

Employee::Employee(string the\_name, string the\_number)

: name(the\_name), ssn(the\_number), net\_pay(0)

{ }

string Employee::get\_name() const

{

return name;

}

string Employee::get\_ssn() const

{

return ssn;

}

double Employee::get\_net\_pay() const

{

return net\_pay;

}

void Employee::set\_name(string new\_name)

{

name = new\_name;

}

void Employee::set\_ssn(string new\_ssn)

{

ssn = new\_ssn;

}

void Employee::set\_net\_pay(double new\_net\_pay)

{

net\_pay = new\_net\_pay;

}

void Employee::print\_check() const

{

cout << "\nCannot print a check for an uncatagorized employee!" << endl;

}

#endif EMPLOYEE\_H

Main File

#include<iostream>

#include<string>

#include "Employee.h"

#include "HourlyEmployee.h"

#include "salaryEmployee.h"

using namespace std;

//main method

int main()

{

//Create an instance of Employee class

Employee emp("johnson", "12345");

//calling print\_check

emp.print\_check();

//Create an instance of Employee class

SalariedEmployee salEmp("mark", "007", 1200);

//calling print\_check

salEmp.print\_check();

//Create an instance of Employee class

HourlyEmployee hourlyEmp("Bigd", "1235", 5, 40);

//calling print\_check

hourlyEmp.print\_check();

//pause program output console

system("pause");

return 0;

}

Would not run in unix. Ran in visual studio. Error code from unix pictured below

