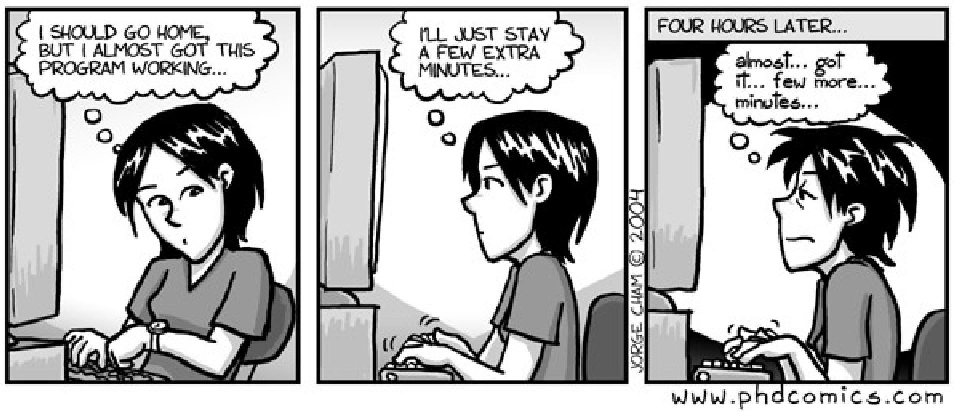
CS 202 – Assignment #3

Purpose: Learn class inheritance, multi-file class implementation, and make utility.
Points: 100

Video Link for Assignment3:
https://unlv.webex.com/unlv/ldr.php?RCID=c6be58df7b3d37897dc960250ca9d5a3

Assignment:
Design and implement three C++ classes to provide an employee and manager tracking functions. Each of the classes is described below.


I SHOULD GO HOME, BUT I ALMOST GOT THIS PROGRAM WORKING...
I'LL JUST STAY A FEW EXTRA MINUTES...
FOUR HOURS LATER...
almost... got it... few more... minutes...
JORGE CHAM © 2004
www.phdcomics.com

The classes we will implement are as follows:

- **Employee Class**
  - Implement a basic employee class, ***employee***, will track employee information and provide standard support functions for all employees. This will include name (first and last), ID, pay rate, tax rate hours worked and employee status. This will be the *base class* as this will apply to any employee (or manager). The manager class will inherit this functionality, so this class must be fully working before working on the manager class.

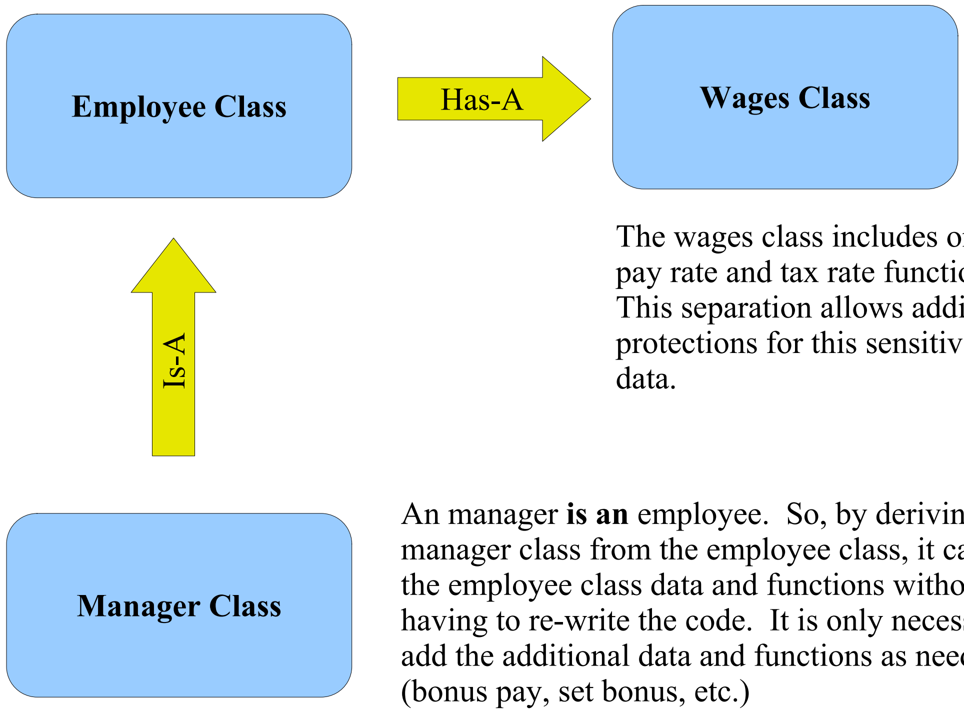
- **Wages Class**
  - Implement a small class, ***wages***, that contains only the financial data for pay rate function, tax rate functions and hours worked functions. This isolates the sensitive financial information and allows for encryption and reporting as is typically required for state and federal reporting.

- **Manager Class**
  - Implement a *manager* class will extend the functionality of the *employee* class with some additional functionality that applies only to managers. Specifically, the *manager* class will add some bonus tracking functionality. Since the *manager* will extend the *employee* class functionality, *manager* will be derived form the *employee* class.

Class Hierarchy

The following diagram should help understand the class hierarchy for this project.

The employee base class encompasses basic employee data and functions.



The wages class includes only pay rate and tax rate functions. This separation allows additional protections for this sensitive data.

An manager **is an** employee. So, by deriving the manager class from the employee class, it can use the employee class data and functions without having to re-write the code. It is only necessary to add the additional data and functions as needed (bonus pay, set bonus, etc.)

Inheritance, composition and fundamental concepts are important in object oriented programming and it is essential to fully understand. Refer to the text and class lectures for additional information regarding inheritance and composition.

Development and Testing

In order to simplify development and testing, the project is split into three (2) main parts; employee and wage classes, and manager class.

- The employee and wages classes must be developed first (before the derived class). These classes can be tested independently of the manager class. An independent main that uses and tests the employee and wages classes is provided and can be built independently using the provided main file (see next section). *Note*, this is the largest portion of the project.
- The manager class will use the employee and wages classes so they should be fully working and tested before starting the manager class.

Make File:

The provided make file assumes the source file names are as described. If you change the names, the make file will need to be edited. To build the employee class or manager class provided mains;

```
make emain
make main
```

Which will create the *emain* (for *employee* class) and *main* (for *manager* class) executables respectively.

Submission:

- All files must compile and execute on Ubuntu and compile with C++11.

- Submit source files
  - Submit a copy of the **source** files via the on-line submission.
  - *Note*, do ***not*** submit the provided mains (we have them).

- Once you submit, the system will score the project and provide feedback.
  - If you do not get full score, you can (and should) correct and resubmit.
  - You can re-submit an unlimited number of times before the due date/time.

- Late submissions will be accepted for a period of 24 hours after the due date/time for any given lab. Late submissions will be subject to a ~2% reduction in points per an hour late. If you submit 1 minute- 1 hour late -2%, 1-2 hours late -4%, … , 23-24 hours late -50%. This means after 24 hours late submissions will receive an automatic 0.

Program Header Block

All program and header files must include your name, section number, assignment, NSHE number, input and output summary. The required format is as follows:

```
/*
Name: MY_NAME, NSHE, CLASS-SECTION, ASSIGNMENT
Description: <per assignment>
Input: <per assignment>
Output: <per assignment>
*/
```

Failure to include your name in this format will result in a loss of up to 5%.

Code Quality Checks

A C++ linter[1] is used to perform some basic checks on code quality. These checks include, but are not limited to, the following:

- Unnecessary 'else' statements should not be used.
- Identifier naming style should be either camelCase or snake_case (consistently chosen).
- Named constants should be used in place of literals.
- Correct indentation should be used.
- Redundant return/continue statements should not be used.
- Selection conditions should be in the simplest form possible.
- Function prototypes should be free of top level *const*.

Not all of these items will apply to every program. Failure to to address these guidelines will result in a loss of up to 5%.

1 For more information, refer to: https://en.wikipedia.org/wiki/Lint_(software)

<u>Employee Class</u>
Implement a basic *employee* class to provide employee functions. The *employee* UML class diagram is as follows:

employee
`-lastName, firstName: string`
`-employeeID: string`
`+employee(string lst="", string frst="", double pyRt=0.0, string rmpID="", eStats es=NONE, double taxRt=0.0)`
`+getLastName() const: string`
`+getFirstName() const: string`
`+getPayRate() const: double`
`+getEmployeeID() const: string`
`+getEmploymentStatus() const: eStats`
`+getTaxRate() const: double`
`+getHoursWorked() const: double`
`+setEmployeeName(string lst, string frst): void`
`+setPayRate(double newPayRate): void`
`+setEmployeeID(string newID): void`
`+setEmploymentStatus(eStats newStat): void`
`+setTaxRate(double newRate): void`
`+setHoursWorked(double hrs): void`
`+grossPay() const: double`
`+takeHomePay() const: double`
`+showPayStub() const: void`
`-checkID(string employeeIDTmp) const: bool`
`-employementStatus: eStats`

Note: You need an object of wages(composition) in employee class(public).

In addition, the following parameters and enumeration must be defined:

```
enum eStats{FULLTIME, PARTTIME, CASUAL, NONE};
```

<u>Employee Class Function Descriptions:</u>
The following are more detailed descriptions of the required functions.

- setTaxRate(double), getTaxRate(), setHoursWorked(double), getHoursWorked(double), grossPay(), takeHomePay(),
- Function *employee()* is the constructor. Initializes the class variables to the passed arguments (if any). The parameters pyRt and txRt are inherited from wagesObj. The hoursWorked class variable should be set to 0.0. The constructor should ensure the values for pay rate and tax rate are positive and not more than the applicable defined maximum value (as defined in wages.h). The ID must start with an upper case letter followed by exactly 5 digits. An error message should be displayed for all other cases of ID. (Note: You can use "if (checkID(string))" for checking Employee ID)
- Function *getLastName()*, function *getFirstName()*, and function *getEmployeeID()* return the last name, first name, or employee ID strings respectively.
- Function *getEmploymentStatus()* returns the enumeration value.

- Functions *getPayRate()*, *getTaxRate(),* and *getHoursWorked(),*return the pay rate, tax rate, or hours worked values respectively. These functions will use the wages class object and call the appropriate functions of wages.h. (For example, to getPayRate(), you return wagesObj.getPayData().)
- Function *setEmployeeName()* should set the class variables for first and last name.
- Function *setEmployeeID()* should set the class variable for employee ID. The ID must start with an upper case letter followed by exactly 5 digits. An error should be displayed otherwise. (Note: You can use "if (checkID(string))" for checking Employee ID)
- Function *setEmploymentStatus()* should set the class variable to the passed enumeration value of type sStats. No other values should be accepted.
- The functions *setPayRate()*, and *setTaxRate()*, and *setHoursWorked()* set the pay rate, tax rate, or hours worked respectively to the passed value. These functions will use the wages class object and call the appropriate functions of wages.h. (For example, to setPayRate(double newPayRate), your definition is wagesObj.setPayData(newPayRate).)
- The function *grossPay()* returns the gross pay (hours worked * pay rate). However, if the hours worked is more than 40.0, the number of hours over 40 should be calculated. Then extra time hours pay should be calculated at half pay rate. For 40 hours above, grosspay=((total hours worked*pay rate) + (extra hours *(pay rate/2)))
- The function *takeHomePay()* computes the final take home pay by subtracting the medicare tax (grosspay()/100 * MEDICARE_TAX), the social security tax (grosspay()/100 * SOCIAL_SECURITY_TAX), and the federal tax (grosspay()/100 * tax rate). This function returns the take home pay value after the above calculations.
- The function *showPayStub()* should show the employee pay stub in the below format. The withholding values and take home pay should line up (as shown). The 'Status' should line up as shown (regardless of the length of the name). The cents must be displayed as shown, two digits (with leading 0).

```
----------------------------------------------------------------------
Name: Simpson, Homer                Status: FULLTIME
Employee Gross Pay:    1024.00
GetHoursWorked:          32.00
      Withholding
            Medicare:           14.85
            Social Security:    63.49
            Federal Tax:       240.64
       Take Home Pay:          705.02

----------------------------------------------------------------------
```

Use the provided main, *emain.cpp*, to demonstrate that the *employee* class functions correctly. The main and provided makefile reference files *employeeImp.cpp* and *employee.h*. Your implementation files should be fully commented (as per the example from previous assignments).

<u>Wages Class</u>

Implement a basic student financial class named ***wages*** to provide student financial support functions. The *wages* UML class diagram is as follows:

wages
`-payRate=0.0: double`
`-taxRate=0.0: double`
`-hoursWorked=0.0: double`
`+wages(double=0.0, double=0.0, double=0.0)`
`+getPayData(double &, double &) const: void`
`+getTaxRate() const: double`
`+getHoursWorked()const: void`
`+setPayData(double): void`
`+setHoursWorked(double): void`
`+setTaxRate(double): void`
`+MEDICARE_TAX = 1.45: static constexpr double`
`+SOCIAL_SECURITY_TAX = 6.2: static constexpr double`
`+MAX_HOURS = 80: static constexpr double`
`+MAX_PAY_RATE = 2500.0: static constexpr double`
`+MAX_TAX_RATE = 50.0: static constexpr double`

<u>Wages Class Function Descriptions:</u>

The following are more detailed descriptions of the required functions.

- Function *wages(double payRateTmp, double taxRateTmp, double hoursWorkedTmp)* is the constructor. Initializes the class variables to the passed arguments (if any). This function should ensure each value is positive and not more than the applicable defined maximum value. If the passed value is not correct, an error message should be displayed like "Error:- invalid pay rate.", "Error:- invalid tax rate." (and the variable unchanged).
- Functions *getPayData()*, *getTaxRate(),* and *getHoursWorked(),*return the pay rate, tax rate, or hours worked values respectively.
- Functions s*etPayData(double)*, s*etTaxRate(double),* and *setHoursWorked(double), set* the pay rate, tax rate, or hours worked values respectively. Each function should ensure their respective value is positive and not more than the applicable defined maximum value. If the passed value is not correct, an error message should be displayed like "Error:- invalid pay rate.", "Error:- invalid tax rate.", "Error:- invalid Hours worked." in their respective functions.

Manager Class
The *manager* class is a derived class from *employee* class. This *employee* class is inherited through public member access specifier. The *manager* class will provide the same functionality and the *employee* class with some additional functionality. Specifically, the *manager* class will add some additional capabilities including bonus functionality. Since the *manager* will extend the *employee* class functionality, *manager* will be derived form the *employee* class.

The ***manager*** UML class diagram is as follows:

manager
`-bonusPercentage =0.0: double`

`+manager(string"", string="", double=0.0,`
`   string "", eStats=NONE, double=0.0, double=0.0)`

`+grossPay() const: double`
`+bonusPay() const: double`
`+showPayStub() const: void`
`+setBonusPct(double): void`
`+getBonusPct() const: double`
`+setHoursWorked(hrs: double): void`
`+BONUS_TAX_RATE = 25.0: static constexpr double`
`+MAX_PBONUS_PCT = 100.0: static constexpr double`

For a manager, the gross pay will be based on a 40 hour week (unless the hours worked is 0, in which case the gross pay is 0.0). Additionally, the manager will also receive a bonus check (bonus percentage * gross pay) however the bonus will be taxed at a flat rate of BONUS_TAX_RATE.

Manager Class Function Descriptions:
The following are more detailed descriptions of the required functions.

- Function *manager()* is the constructor. Initializes the class variables to the passed arguments (if any). The bonus percentage must be positive and not exceed the MAX_BONUS_PCT value.
- The function *grossPay()* returns the gross pay. The gross pay for a manager is based on a 40 hours work week even if the manager worked less than or more than 40 hours, (unless the hours worked is 0, in which case the gross pay is 0.0).
- The function *bonusPay()* returns the bonus pay. This function can be used to know the payRate of the manager. This function returns employee::getPayRate()
- The function *setHoursWorked(double)* sets the hours worked based on the passed value. If the value is more than 0.0, the hours worked for a manager is set to 40.0.
- The function *setBonusPct(double)* will set the bonus percentage to the passed value. The bonus percentage must be positive and not exceed the MAX_BONUS_PCT value. If the passed value is not with the appropriate range, an error message should be displayed.
- The function *getBonusPct()* will return the current value of the bonus percentage. This function returns the percentage of (grosspay() *bonusPercentage)
- The function *showPayStub()* should show the manager pay stub in the following format. The withholding for the bonus is subject to the BONUS_TAX_RATE (as previously defined). This function should print *showPayStub()* function similar to the employee class for the regular pay and then adding the bonus information.
  The Federal tax for Bonus Check amount is calculated using the following statement: ((getBonusPct() / 100) * BONUS_TAX_RATE); and Bonus check amount is by subtracting the federal tax of bonus check from bonus amount. Please check the red font in the following sample output of a manager

Note: In the below sample output some data is represented in red color only to differentiate the difference between employee pay and manager pay. Your output no need to print in red color text. The manager gets a bonus check amount apart from take home pay.

```
----------------------------------------------------------------------
Name: Meyers, Roger                 Status: CASUAL
Manager Gross Pay:      2100.00
GetHoursWorked:           30.00
      Withholding
            Medicare:             30.45
            Social Security:     130.20
            Federal Tax:         288.75
       Take Home Pay:           1650.60
Bonus Amount:     735.00
      Withholding
            Federal Tax:               183.75
            Bonus Check Amount:        551.25

----------------------------------------------------------------------
```

Use the provided main, *main.cpp*, to demonstrate that the ***manager*** class functions correctly. The main and provided makefile reference files *managerImp.cpp* and *manager.h*. Your implementation files should be fully commented.
The employee class must be fully completed prior to moving on to the manager class. Refer to the example executions for formatting. Make sure your program includes the appropriate documentation.

Files Provided:
The following files are available to download for this assignment.

- makefile.
- main.cpp.
- emain.cpp
- employee.h
- manager.h
- wages.h

Submission:
Submit the following files for this assignment.

- employeeImp.cpp.
- managerImp.cpp.
- wagesImp.cpp.

How to Compile:
Use the following commands to compile.

- make emain
- make main

How to Run your code:
Use the following commands to run.

- ./main (This will print both manger class and employee class test cases from main.cpp)
- ./emain (This will print employee class test cases from emain.cpp)

For Example Execution:
Check the following example execution files that are uploaded to your assignment supported files.

- ast3_main_log.txt
- ast3_emain_log.txt

Example Execution:
Below is an example program execution output for the *main* program.

```
kishore-vm:kishore$ make emain
g++    -c -o employeeImp.o employeeImp.cpp
g++  -Wall -Wextra -pedantic -std=c++11 -g  -c wagesImp.cpp
g++  -Wall -Wextra -pedantic -std=c++11 -g -o emain emain.o employeeImp.o
wagesImp.o
kishore-vm:kishore$ make main
g++  -Wall -Wextra -pedantic -std=c++11 -g -c main.cpp
g++    -c -o managerImp.o managerImp.cpp
g++  -Wall -Wextra -pedantic -std=c++11 -g -o main main.o managerImp.o
employeeImp.o wagesImp.o
kishore-vm:kishore$ ./emain


----------------------------------------------------------------------
CS 202 - Assignment #3
Employee Class Test Program.

Error :- invalid employeeID
Error :- invalid employeeID
Error :- invalid employeeID
Error :- invalid employeeID
Error :- invalid employeeID
Error :- invalid employeeID
Error :- invalid employeeID
Error :- invalid employeeID
Error :- invalid employeeID
Error:- invalid tax rate.
Error:- invalid tax rate.
Error:- invalid pay rate.
Error:- invalid pay rate.
Error:- invalid pay rate.
Error:- invalid Hours worked.
Error:- invalid Hours worked.

----------------------------------------------------------------------
 Name: Hutz, Lionel                Status: PARTTIME
 Employee Gross Pay:        500
 GetHoursWorked:             40
     Withholding
          Medicare:            7.25
          Social Security:    31.00
          Federal Tax:        61.25
     Take Home Pay:          400.50

----------------------------------------------------------------------
 Name: Gumble, Barney                Status: PARTTIME
 Employee Gross Pay:     480.00
 GetHoursWorked:          40.00
     Withholding
          Medicare:            6.96
          Social Security:    29.76
          Federal Tax:        93.60
     Take Home Pay:          349.68

----------------------------------------------------------------------
 Name: Spuckler, Cletus                Status: CASUAL
 Employee Gross Pay:       5.00
 GetHoursWorked:           5.00
     Withholding
          Medicare:            0.07
          Social Security:     0.31
          Federal Tax:         1.39
     Take Home Pay:            3.23

----------------------------------------------------------------------
```

```
Name: Terwilliger, Cecil                Status: NONE
Employee Gross Pay:       0.00
GetHoursWorked:           0.00
    Withholding
         Medicare:            0.00
         Social Security:     0.00
         Federal Tax:         0.00
    Take Home Pay:            0.00

----------------------------------------------------------------------
Name: Simpson, Homer                Status: FULLTIME
Employee Gross Pay:    1024.00
GetHoursWorked:          32.00
    Withholding
         Medicare:           14.85
         Social Security:    63.49
         Federal Tax:       240.64
    Take Home Pay:          705.02

----------------------------------------------------------------------
Name: Leonard, Lenny                Status: FULLTIME
Employee Gross Pay:    1320.00
GetHoursWorked:          40.00
    Withholding
         Medicare:           19.14
         Social Security:    81.84
         Federal Tax:       227.70
    Take Home Pay:          991.32

----------------------------------------------------------------------
Name: Carlson, Carl                Status: FULLTIME
Employee Gross Pay:    1680.00
GetHoursWorked:          40.00
    Withholding
         Medicare:           24.36
         Social Security:   104.16
         Federal Tax:       361.20
    Take Home Pay:         1190.28

----------------------------------------------------------------------
Name: McClure, Selma                Status: PARTTIME
Employee Gross Pay:    1375.00
GetHoursWorked:          50.00
    Withholding
         Medicare:           19.94
         Social Security:    85.25
         Federal Tax:       230.31
    Take Home Pay:         1039.50

----------------------------------------------------------------------
Name: Krabappel, Edna                Status: FULLTIME
Employee Gross Pay:    1600.00
GetHoursWorked:          40.00
    Withholding
         Medicare:           23.20
         Social Security:    99.20
         Federal Tax:       240.00
    Take Home Pay:         1237.60

----------------------------------------------------------------------
Name: Nahasapeemapetilon, Apu                Status: PARTTIME
Employee Gross Pay:    1056.00
GetHoursWorked:          50.00
    Withholding
         Medicare:           15.31
         Social Security:    65.47
         Federal Tax:       205.92
    Take Home Pay:          769.30

----------------------------------------------------------------------
```

```
 Name: Groundskeeper, Willie                Status: FULLTIME
 Employee Gross Pay:       0.00
 GetHoursWorked:           0.00
     Withholding
          Medicare:            0.00
          Social Security:     0.00
          Federal Tax:         0.00
     Take Home Pay:            0.00

----------------------------------------------------------------------
 Name: Riviera, Nick                Status: FULLTIME
 Employee Gross Pay:   28000.00
 GetHoursWorked:          60.00
     Withholding
          Medicare:          406.00
          Social Security:  1736.00
          Federal Tax:      6230.00
     Take Home Pay:        19628.00

kishore-vm:kishore$
```