Assignment 3: Employee Management and Scheduler Application

Dates:

Assigned: 2013-06-27 Thursday
Due: 2013-07-09 **Tuesday** at 11:59 pm

Questions – email rjmfff@mail.umkc.edu

Make sure that your source code is *committed* and *synced* to your student GitHub account. You do not need to turn in the code via Blackboard or anything else.

Note that you have extra time to work on this assignment – **it is NOT due July 4th!**

Assignment Tips:

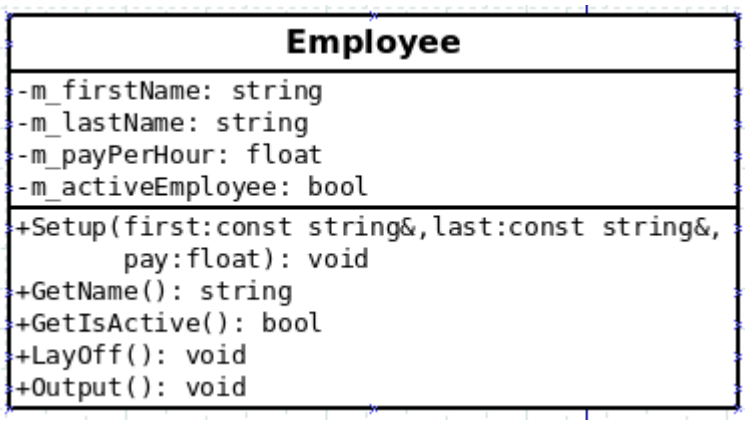
- Get one feature working at a time – and commit your changes after each working feature!
- Build often – Don't let your program sit around with build errors!
- Break down large problems into smaller ones! Focus on one thing at a time, even if you have to write some “throwaway” code!
- Work on the Employee & employee management functionality first before starting on the scheduling

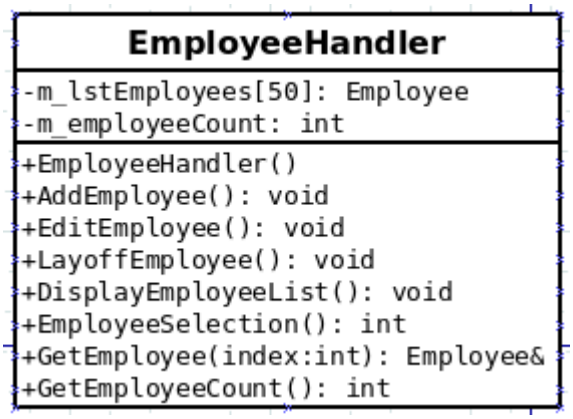
Specification:

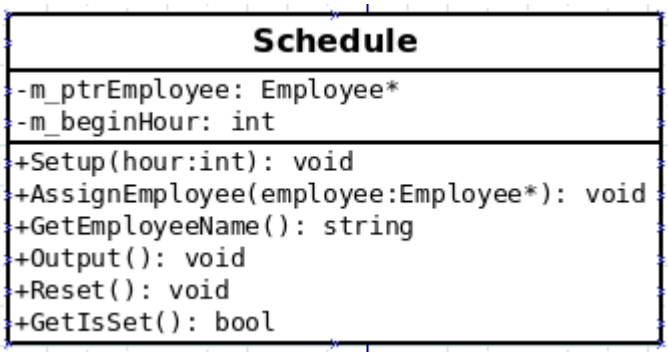
We are going to write an employee management and scheduling application. The user will be able to add, update, and lay-off employees, as well as update and clear out the schedule. Additionally, they will also be able to export the schedule to a CSV or HTML file.

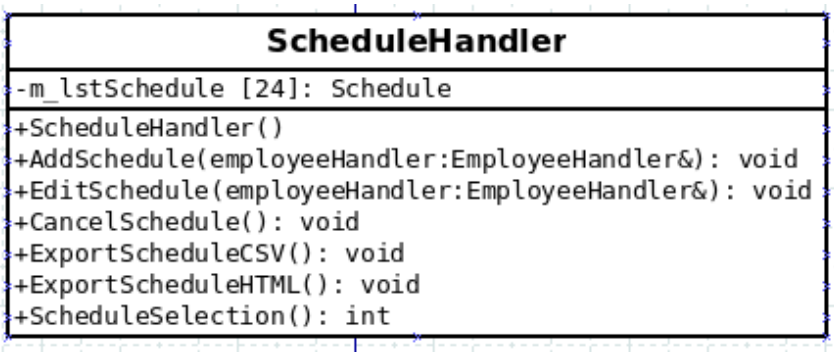
This project contains references, pointers, class composition, and file output.

Class Diagrams









Sample Menus

Main Menu

```
BUSINESS MANAGEMENT system!
---------------------------
EMPLOYEE OPTIONS
1. Add Employee
2. Edit Employee
3. Layoff Employee
4. View Employee List
---------------------------
SCHEDULING OPTIONS
5. Update Schedule
6. Cancel Schedule
7. View Schedule
8. Export Schedule to CSV
9. Export Schedule to HTML
---------------------------
0. Quit
---------------------------
```

New Employee

```
NEW EMPLOYEE
First name? Guybrush
Last name? Threepwood
Pay rate? $8.99
** Employee 0 added
```

New Schedule

```
---------------------------
Please enter an option: 5
[0]:     0:00 - UNALLOCATED
[1]:     1:00 - UNALLOCATED
[2]:     2:00 - UNALLOCATED
[3]:     3:00 - UNALLOCATED
          (etc.)
Enter the schedule hour-block: 0
EMPLOYEES
>> [0]:         Guybrush Threepwood, PAY: $8.99 (CURRENT EMPLOYEE)
>> [1]:         Elaine Marley, PAY: $8.99 (CURRENT EMPLOYEE)
>> [2]:         G.P. LeChuck, PAY: $10.99 (CURRENT EMPLOYEE)
Enter the index of the employee: 1
Assigned 0 to employee Elaine Marley
```

Step-By-Step

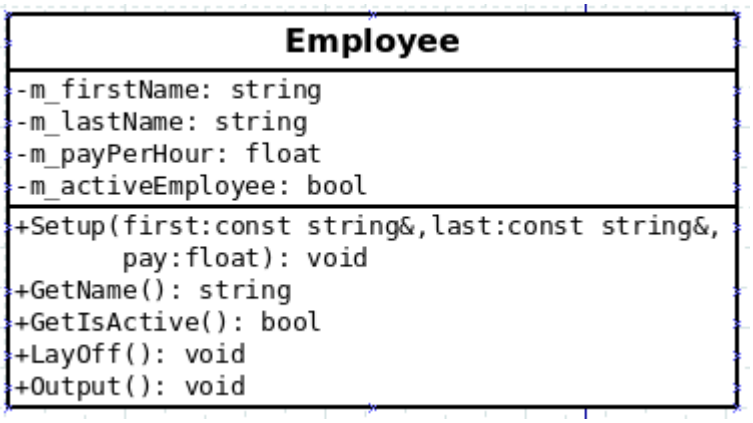
This is a sample step-by-step and you are not required to follow this.

Iteration 1: Employee Management System

First, we are just going to concentrate on getting the Employee system to work – adding, updating, and removing.

Step 1: Creating an Employee object

Create files **Employee.h** and **Employee.cpp** and set up the following structure:



We are storing the employee's first and last name, their pay, and whether or not they're currently working at the company (When we lay off an employee, they won't be removed from the system but marked as inactive).

Setup Function

In the Setup function, we have parameters for first name, last name, and pay. Assign these parameters' values to the private members, **m_firstName**, **m_lastName**, and **m_payPerHour**.

GetName Function

This function should just return the Employee's name. Concatenate first and last name together in any way you like. For example: “Lastname, Firstname” or “Firstname Lastname”. Return this concatenated string.

GetIsActive Function

Return **m_activeEmployee**

Layoff Function

Set the employee's **m_activeEmployee** variable to “false” - this will mark them as no longer working for the company.

Output Function

Use **cout** to output the employee's first/last name, pay, and active status (current/former employee).

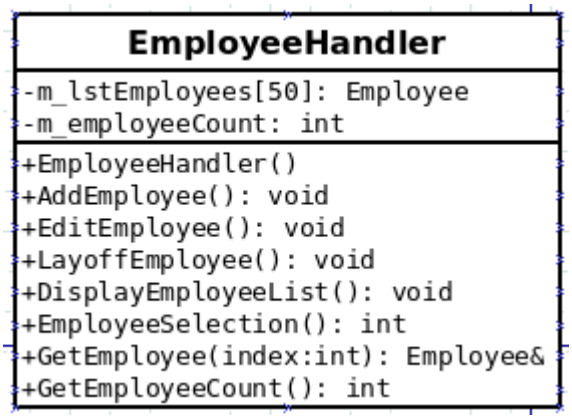
Step 2: Updating main()

In the main() function, create an **Employee** object and set its name and pay with the **Setup** function. Afterwards, call the **Output** function to make sure it's being stored correctly.

Build and run – Make sure everything is working.

Step 3: Setting up the EmployeeHandler class

Create two new files: **EmployeeHandler.h** and **EmployeeHandler.cpp**. We are going to create a class that matches this diagram:



The EmployeeHandler will be an added layer – instead of main() handling all of the functionality related to the Employee class, we've created EmployeeHandler instead.

In main(), we'll get the user's choice for a command, then pass the torch to EmployeeHandler (or ScheduleHandler, later) and have them handle that specific functionality.

For private members, EmployeeHandler will have an array of Employees – size 50. It will also have a counter variable that keeps track of how many employees we have.

EmployeeHandler Constructor

This constructor will simply initialize the **m_employeeCount** variable to 0.

DisplayEmployeeList Function

This function will output all employees from 0 to the value of **m_employeeCount**.

cout the index of the particular Employee element, then call the Employee's **Output** function to automatically handle outputting the name and pay-rate.

EmployeeSelection Function

This function will output a list of all employees and ask the user to enter the index of the employee they wish to modify. Because this behavior is shared between multiple functions, we're pulling it out into its own function to reduce *duplicate code*.

First, call the **DisplayEmployeeList** function, then ask the user to enter an index and store their answer in an **int** variable. The user's choice will be what we **return**.

AddEmployee Function

Here, we will create a series of prompts to get the first name, last name, and pay-rate information from the user. Store this information in **temporary "buffer" variables**. Once all the information is collected, we can call the **Setup** function of the current Employee.

The current Employee in the EmployeeHandler is at the index of **m_employeeCount**. Don't forget to increment **m_employeeCount** by one after adding a new employee!

> Make sure to format the console output nicely!

EditEmployee Function

Begin by calling **EmployeeSelection** – this will show the user a list of all employees and allow them to select one by index. The EmployeeSelection function returns an int, corresponding to an Employee element index in our employee array (**m_lstEmployee**). Store this in an integer variable.

After the user has selected an Employee to update, ask them again for new first name, last name, and pay values. Store these in **temporary buffer variables**. Afterwards, you can simply call the Employee's **Setup** function again to update its information.

LayoffEmployee Function

For this function, we will begin by calling **EmployeeSelection** and getting the index of the Employee to be laid off. Once we have this index variable, call the Employee's **LayOff** function.

GetEmployee Function

This function is going to return a *reference to an Employee variable*, with the reference pointing to a specific element in the array. This function requires an int parameter – the index of the Employee in the array.

Return the Employee at that index.

GetEmployeeCount Function

This function simply returns **m_employeeCount**.

Build your program to make sure you don't have any syntax errors before continuing.

Step 4: Creating a main menu

Back in the file that stores main(), let's create an enumeration. Remember that an enumeration is a way for us to represent integers as text labels.

Declare this enumeration:

```
enum MenuOptions { QUIT,
     ADD_EMPLOYEE, EDIT_EMPLOYEE, LAYOFF_EMPLOYEE, VIEW_EMPLOYEE,
     UPDATE_SCHEDULE, CANCEL_SCHEDULE, VIEW_SCHEDULE,
     EXPORT_SCHEDULE_CSV, EXPORT_SCHEDULE_HTML };
```

We are only going to work with the Employee menu options at the moment, but go ahead and add everything. Remember that this enum is declared *outside* of main().

Next, let's create a function in the same file (with main()) called **DisplayMainMenu**

DisplayMainMenu Function

This function has a return-type of **void** and no parameters. This will be our main menu for the program, containing the number codes for each piece of functionality. For example:

```
cout << endl << endl;
cout << "BUSINESS MANAGEMENT system!" << endl;
cout << "--------------------------" << endl;
cout << "EMPLOYEE OPTIONS" << endl;
cout << ADD_EMPLOYEE << ". Add Employee" << endl;
cout << EDIT_EMPLOYEE << ". Edit Employee" << endl;
cout << LAYOFF_EMPLOYEE << ". Layoff Employee" << endl;
cout << VIEW_EMPLOYEE << ". View Employee List" << endl;
```

Notice that we're using the enum values, **ADD_EMPLOYEE**, **EDIT_EMPLOYEE**, etc. in this menu. When we output these labels, the program will just show plain integers (NOT the text labels!)

Add the menu option output for the rest of the enumeration options we've created.

Back in the **main()** function, create a boolean variable to keep track of whether the program is finished or not. Create a loop that will continue looping until the program is done.

Also, create an **EmployeeHandler** variable named “employeeHandler”.

Inside of the while loop, we will display the menu (call the **DisplayMainMenu()** function) and get the user's choice of what to do.

Then, using either an **if statement** or **switch statement**, we will call different functions within **EmployeeHandler** (or, later, ScheduleHandler) to handle that functionality.

If Statement sample

```
if ( choice == ADD_EMPLOYEE )
{
     employeeHandler.AddEmployee();
}
```

Switch Statement sample

```
switch( choice )
{
     case ADD_EMPLOYEE:
          employeeHandler.AddEmployee();
     break;
}
```

Add additional **else if** or **case** statements for each menu option, even though we can't implement everything at the moment.

Build the program to make sure it compiles. If you have no errors, try running it. You should be able to:

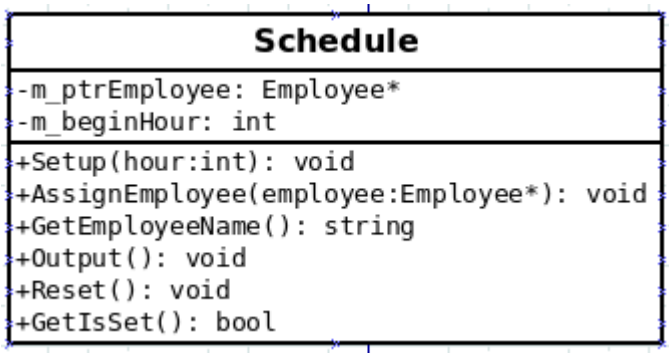
- View the main menu
- Use the “Add Employee” option
- Use the “Edit Employee” option
- Use the “Layoff Employee” option
- Use the “View Employees” option
- Use the “Quit” option

If it's not working right, spend some time debugging before continuing on.

Iteration 2: Scheduling System

Step 1: Creating a Schedule class

Create two new files: **Schedule.h** and **Schedule.cpp**.



For private members, we will have a pointer to an Employee (this is the employee scheduled for this time-slot), and an integer for the hour they begin their shift. We are going to assume everybody works one-hour at a time.

Setup Function

Our ScheduleHandler class will contain an array of 24 Schedule items (for one full day). When its setting up the Schedules, it will tell each element which index it is at. This corresponds to an hour, and the Schedule will not change position in the array.

Within the Setup function, we will initialize **m_ptrEmployee** to NULL, and set **m_beginHour** equal to the hour parameter.

AssignEmployee Function

This function will have a pointer to an employee object as a parameter. Simply set **m_ptrEmployee** equal to this parameter.

Reset Function

This function only sets **m_ptrEmployee** equal to NULL again.

GetIsSet Function

To tell whether a schedule is “set”, we just need to know if there is an employee assigned to this time-slot. Simply check whether **m_ptrEmployee** is NULL – if it *is NULL*, then this schedule block is free (ie, isSet = false). If it is *not NULL*, then this block has been taken.

Output Function

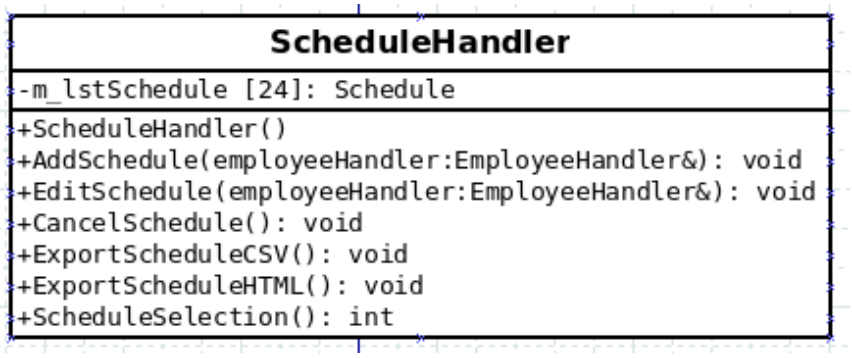
This will output the **m_beginHour**, and then output either “UNALLOCATED” or the employee's name based on whether or not **m_ptrEmployee** is NULL.

GetEmployeeName Function

This function will call the GetName() function that is part of the Employee class, IF the **m_ptrEmployee** pointer is not set to NULL. Call the m_ptrEmployee's GetName() function and return that value, otherwise return “UNALLOCATED”.

Compile the program to make sure it's working.

Step 2: ScheduleHandler class



For private members in this class, we only have an array of 24 Schedule objects. Each index represents an hour in a day (from 0 to 23).

ScheduleHandler Constructor

This constructor will be responsible for initializing every Schedule element in the **m_lstSchedule** array.

Create a for loop to iterate through all of the elements, calling the Schedule **Setup** function and passing in the index of that element.

DisplaySchedule Function

This function has a for loop to iterate through all of the elements.

For each item, output the index (the variable in the for-loop) and call the Schedule's **Output** function.

ScheduleSelection Function

This function will call **DisplaySchedule** and then ask the user to enter an index for the hour they want to update. Return this integer choice.

AddSchedule Function

This function has a parameter that is a Reference to a EmployeeHandler.

In this function, begin by calling **ScheduleSelection** and storing that in an integer that represents the index of the Schedule element we're going to update.

After the user has chosen an hour to update, they need to next choose which Employee to assign to that hour slot. Call the EmployeeHandler's **EmployeeSelection** function. Store the return value of this function in an "employeeIndex" integer.

Next, we want to get the actual Employee itself. To do this, we can use the EmployeeHandler's **GetEmployee** function. This function returns a *reference to an employee object*. Store this in a variable of type Employee& (employee reference).

```
Employee& refEmployee =
     employeeHandler.GetEmployee( employeeIndex );
```

So now we have the **hour** index and an **employee**. Using the **m_lstSchedule** array, access the element at the index **hour**. With this element, call the function **AssignEmployee**, and pass the ***address of the employee***. You need to use the address-of operator!

CancelSchedule Function

For this function, call the **ScheduleSelection** function to get the hour index. Call the **Reset** function for the Schedule item at that index.

ExportScheduleCSV Function

In this function, we need to create an output file stream (ofstream). Create an ofstream and open the file “Schedule.csv”.

> A CSV file is a “Comma-Separated Value” file. You can open these files in Excel or another Spreadsheet program to view the data.
> Rows are separated by new-lines “\n”, while colums are separated by commas “,”

Start by outputting a header for the file:

```
outfile << "HOUR,EMPLOYEE" << endl;
```

Then, we will use a for-loop to iterate through every Schedule. Stream out the **index**, then a comma, then the Employee's name via the **GetEmployeeName** function that is part of the Schedule class.

```
for ( int i = 0; i < 24; i++ )
{
     outfile << i << ":00," << m_lstSchedule[i].GetEmployeeName()
          << endl;
}
```

Remember to close your output file stream when you're done!

ExportScheduleHTML Function

Create an ofstream object and open the filename, “Schedule.html”.

You can output the following – this is HTML and CSS code (copy and paste):

```
outfile << "<head><title>Schedule</title>" << endl;
outfile << "<style type='text/css'>" << endl;
outfile << "table tr td { border: solid 1px #000000; width: 200px;";
outfile << "background: #cccccc; }" << endl;
outfile << "</style></head>" << endl;
outfile << "<body><table>" << endl;
outfile << "<tr><td>HOUR</td><td>EMPLOYEE</td></tr>" << endl;
for ( int i = 0; i < 24; i++ )
{
     outfile << "<tr>" << endl;
     outfile << "<td>" << i << ":00</td>" << endl;
     outfile << "<td>";
     outfile << m_lstSchedule[i].GetEmployeeName();
     outfile << "</td>" << endl;
     outfile << "</tr>" << endl;
}
outfile << "</table></body>" << endl;
```

When you're done, close the output file stream.

Step 3: Updating main()

Back in main() we can finish up adding our functionality.

Create a ScheduleHandler object inside of main() and outside of the while loop.

Within each if statement / switch statement case, call the appropriate function.

Build, run, and test out. Add some employees, update the schedule, export, and then try to open those files outside of the program.

Bonus Points:

Here is some things you can add to improve your program (and get some extra points):

- Add dynamic array for EmployeeHandler's m_lstEmployee array.
- Add error-checking to make sure the user doesn't enter any indices that are out-of-bounds!
- Don't let the user schedule a laid off employee in the daily schedule!
- Add functionality to export the list of Employees (names & pay) in addition to our Schedule exporter!