CMPSC 122 Project 1 User Interface and Analysis Report

Introduction

This project is based on creating payroll software. It gives a menu to a supervisor where they can view employee information, add a timecard to an employee, or approve timecards from a queue that are pending approval. The program would be used to keep track of employee time card's which are read in from .txt files.

1. TimeCard.h: the documentation of the TimeCard class and TimeCard.cpp: the implementation of the TimeCard class - The TimeCard class allows TimeCard objects to be created that contain the pay period start date in month, day, and year respectively, a variable that indicates whether or not the TimeCard is approved, and an array for the hours worked every day for the seven days of the week. The class contains functions to set the pay period start date, set the hours worked on a day of the week, and to set the approval of an object. It also has functions to return the approval, return the hours worked on a specific day, output the pay period start date from a timecard to the console, return the gross weekly pay from a single timecard, and display all of the information contained in a timecard to the console. In general it simulates a timecard for a worker and the ability to manipulate that time card. In the hierarchy of the program an instance of the TimeCard class is one time card which belongs to one employee. An employee will hold many time cards which the supervisor will need to approve.

2. Employee.h: the documentation of the Employee class and Employee.cpp: the implementation of the Employee class – The Employee class allows Employee objects to be created which contain the name of the employee, the employee's job title, the employee's wage, an array of up to 16 pointers to TimeCard objects, and the logical size of the TimeCard array. The class contains functions to set the name, title, and wage of an employee and to return all of that information individually. It also contains functions to return the logical size of the array, read in timecards from a file, add a timecard to the array, return the gross pay for all timecards, print the employee's info to the console, and print the contents of all timecards. The class in general holds onto Employee specific information and the ability to manipulate that information. In the overall software, one instance of Employee represents a single employee which a supervisor will have responsibility over.

3. Queue.h: the documentation of the Queue class and Queue.cpp: the implementation of the Queue class - The Queue class allows the creation of Queue objects that contain pointers to pending timecards. Only one queue object is necessary for this project. The class contains functions to return the logical size of the queue, to check if the queue is empty, to add a timecard to the queue, to return the first timecard in the queue and delete it from the queue, and to view the first timecard in the queue without changing it. In the hierarchy of the program, the supervisor has one queue instance which he uses to keep track of time cards for his employees.

4. supervisorUI.cpp: the driver program for the payroll software which includes the menu system for the supervisor to access. The user interface allows for the supervisor to view employee information, check the queue, select and employee and add a time card to that employee, and exit as necessary.

Code

TimeCard.h

```
// Programmer:   Dylan Fetch & Christopher Carney
// Section:      2
// Lab:          Project 1
// Date:         March 5, 2014
// Description:  Header file for timeCard class

#ifndef TimeCard_H
#define TimeCard_H

#include <iostream>
using namespace std;

const int SUNDAY = 0;                    //the named constant for Sunday
const int MONDAY = 1;                    //the named constant for Monday
const int TUESDAY = 2;                   //the named constant for Tuesday
const int WEDNESDAY = 3;                 //the named constant for Wednesday
const int THURSDAY = 4;                  //the named constant for Thursday
const int FRIDAY = 5;                    //the named constant for Friday
const int SATURDAY = 6;                  //the named constant for Saturday
const int DAYS_OF_WEEK=7;                //the constant for the amount of days in a week
```

```
class TimeCard
{
    public:
        TimeCard();
        //POST:   a default timeCard object is constructed with pMonth == 1, pDay == 1, pYear == 2014,
        //        hours[0..6] == 0 && isApproved == false

        TimeCard(int initMonth, int initDay, int initYear);
        //PRE:    initMonth has been initialized, initDay has been initialized, initYear has been
        //          initialized
        //        1<=initMonth<=12, 1<=initDay<=31, initYear>=2014 && initDay is valid in initMonth
        //POST:   a timeCard object is constructed with pMonth set to initMonth, pDay set to initDay,
        //        pYear set to initYear, hours[0..6] == 0 && isApproved == false

        void setMonth(int month);
        //PRE:    month has been initialized, 1 <= month <= 12
        //POST:   The pMonth in this timeCard is set to month

        void setDay(int day);
        //PRE:    day has been initialized, 1 <=day<= 31
        //POST:   The pDay in this timeCard is set to day

        void setYear(int year);
        //PRE:    year has been initialied, year >= 0
        //POST:   The pYear in this timeCard is set to year

        void setHours(double workHours, int day);
        //PRE:    workHours has been initialized, workHours >= 0, day has been initialized &&
        //            day must be a named constant [MONDAY...SUNDAY] || 0 <= day <= 6
        //POST:   sets hours[day] to workHours

        void setApproval(bool approval);
        //PRE:   approval has been initialized
        //       approval is false (0) if unapproved || approval is true (1) if approved
        //POST:  isApproved in this TimeCard has been set to approval

        bool getApproval() const;
        //POST:   FCTVAL == 0 if unapproved || 1 if approved

        double getHours(int day) const;
        //PRE:    day has been initialized, day must be a named constant [MONDAY...SUNDAY]
        //            || 0 <= day <= 6
        //POST:   FCTVAL == hours worked on corresponing day

        void printDate() const;
        //POST:   pay period start date from this TimeCard has been printed in MM/DD/YYYY format

        double weekPay(double wage) const;
        //PRE:    wage has been initialized, wage is in dollars/hour
        //POST:   FCTVAL == gross pay for the week on this TimeCard in dollars

        void viewCard() const;
        //POST:   all information from timecard has been printed in the format:
        //        Pay Period Start Date: MM/DD/YYYY
        //        Approved?: [Approval]
        //        Hours Worked Per Day:
        //        Sunday: [Hours worked SUNDAY]
        //        ...
        //        Saturday: [Hours worked SATURDAY]

    private:
        int pMonth;                      //the month of the start of the pay period
        int pDay;                        //the day of the start of the pay period
        int pYear;                       //the year of the start of the pay period

        bool isApproved;                 //the approval status of the time card

        double hours[DAYS_OF_WEEK];      //the hours worked per day in dollars/hour
};

#endif
```

Employee.h

```
//Programmer:   Dylan Fetch & Christopher Carney
//Section:      2
//Lab:          Project 1
//Date:         March 5, 2014
//Description:  This class models an employee who uses the TimeCard class.

#ifndef Employee_H
#define Employee_H

#include <string>
#include <fstream>
#include "Timecard.h"
#include "Queue.h"
using namespace std;

const int MAX_CARDS = 16;               //the maximum time cards an employee is allowed to have

class Employee
{
    public:
        Employee();
        //POST: a default Employee object is constructed with name = "?", jobTitle = "?",
        //      hourly wage = 7.25 in dollars, currentCards = 0, EmployeeCards[0..15] is initialized
        //      implicitly

        Employee(string initName, string initJobTitle, double initWage);
        //PRE:  initName has been initialized, initJobTitle has been initialized, initWage has been
        //          initialized and is in dollars
        //POST: An Employee object has been constructed with
        //          name = initName, jobTitle = initJobTitle
        //          hourlyWage = initWage and hourlyWage is in dollars
        //          EmployeeCards[0..15] are initialized implicitly

        void giveTimeCard(TimeCard * newCard);
        //POST: a pointer to a TimeCard has been added to this Employee's timecards array
        //POST: The next empty slot in the EmployeeCards array now holds the newCard object
        //          && current currentCards is increased by 1

        void setHourlyWage(double newWage);
        //PRE:  newWage has been initialized, newWage contains the hourly wage of the worker in
        //          dollars
        //POST: The hourlyWage in this Employee has been set to newWage

        void setName(string newName);
        //PRE:  newName is initialized && newName is the name of the employee
        //POST: The name in this Employee has been set to newName

        void setTitle(string newTitle);
        //PRE:  newTitle is initialized && newTitle is the job title of the employee
        //POST: The jobTitle in this Employee has been set to newName

        void readTimeCards(string fileName, Queue & supervisor);
        //PRE:   fileName exists and is set to the exact string name of the .txt file located
        //          in the same directory as the program.
        //       the .txt file is formatted as
        //       APPROVAL MM DD YYYY Su Mo Tu We Th Fr Sa && Su..Sa correspond to the hours
        //          worked on that day
        //       APPROVAL == 0 if unapproved && APPROVAL == 1 if approved
        //          1 <= MM <= 12 && is an integer, 1 <= DD <= 31, is an integer, && is valid in the MM
        //          0 <= YYYY && is an integer
        //       Su..Sa >= 0 && are doubles
        //       supervisor has been initialized
        //POST:  A pointer to a new TimeCard object has been created && the pointer has been added
        //          to this employee's array of TimeCards
        //       If the TimeCard is not approved it has been added to the supervisor queue

        int getNumberOfCards() const;
        //POST: FCTVAL == currentCards, the number of TimeCards this employee currently has
        //          (the logical size of the TimeCards[] array
        string getName() const;
        //POST: FCTVAL == name, the name of this employee
        double getGrossPay() const;
        //POST: FCTVAL = total pay across all timecards from EmployeeCards[0..currentCards-1] in
        //          dollars

        double getHourlyWage() const;
        //POST: FCTVAL == hourly wage in dollars/hour

        void printInfo() const;
        //POST: Basic info about this Employee has been printed in the format:
        //      [name] | [jobTitle] | Hourly Wage: $[hourlyWage] | Current Timecards: [currentCards]

        void printAllCards() const;
        //POST: All TimeCards from EmployeeCards[0..currentCards-1] have been printed in the format
        //          [name] | [jobTitle] | Hourly Wage: $[hourlyWage]
        //          #[current card]
        //          Pay Period Start Date: MM/DD/YYYY
        //          Approved?: [Approval]
        //          Hours Worked Per Day:
        //          Sunday: [Hours worked SUNDAY]
        //          ...
        //          Saturday: [Hours worked SATURDAY]

    private:
        string name;                                //the full name of this employee
        string jobTitle;                            //the job title of this employee
        double hourlyWage;                          //the current hourly wage of this employee

        TimeCard * EmployeeCards[MAX_CARDS];        //the array of TimeCard this employee has
        int currentCards;                           //the logical size of the array which holds
                                                    //   time cards
};

#endif
```

Queue.h

```
// Programmer:     Dylan Fetch & Christopher Carney
// Section:        2
// Lab:            Project 1
// Date:           March 6, 2014
// Description:    Custom Queue Class to be used by TimeCard class documentation

#ifndef Queue_H
#define Queue_H
#include "TimeCard.h"

class Queue
{
    public:
        Queue();
        //POST: a default Queue has been constructed with front and back point to NULL

        int logSize() const;
        //POST: FCTVAL == the number of NodeType nodes contained in the queue starting at front &
        //      ending at back

        bool isEmpty() const;
        //POST: FCTVAL == true (1) if the queue is empty || false (0) if the queue contains a timecard

        void enqueue(TimeCard* & k);
        //PRE:  k is a pointer to a TimeCard object to be enqueued
        //POST: k is inserted into the back of the queue

        TimeCard* dequeue();
        //POST: FCTVAL == the pointer to the newly deallocated front node of this queue
        //      the NodeType node which is pointed to by front has been deallocated, front now
        //          points to the NodeType node which was the previous successor of the deallocated node

        TimeCard* viewFront() const;
        //PRE:  front is a pointer to a NodeType node, back is a pointer to a NodeType node
        //POST: FCTVAL == the timecard at the front of the queue
    private:
        struct NodeType;
        typedef NodeType* NodePtr;
        struct NodeType
        {
            TimeCard* timecard;           //a pointer to hold the timecard of the list node
            NodePtr next;                 //a pointer to the next node in the queue
        };

        NodePtr front;                     //a pointer to the first node in the queue
        NodePtr back;                      //a pointer to the last node in the queue
};

#endif
```

supervisorUI.cpp

```
//Programmer:   Dylan Fetch & Christopher Carney
//Section:      2
//Lab:          Project 1
//Date:         March 7, 2014
//Description:  This driver uses the Employee, TimeCard, and Queue classes to simulate
//                  a payroll program for a supervisor.

#include <iostream>
#include <iomanip>
#include "Employee.h"

using namespace std;

const int MAX_EMPLOYEES = 5;                //the maximum number of employees a supervisor can have

void supervisorCheckEmployee(Employee & selectedEmployee, Queue & supervisor)
//PRE:  selectedEmployee has been initialized
//      supervisor has been initialized
//POST: menu for a single Employee object has been displayed && the user has been prompted to add a
//          TimeCard to an employee or exit the menu
{
    int select;                                 //the menu option selected by the user
    string fileName;                            //the name of the timecard file

    do                                          //continue to display the menu for the specific
    {                                           //  employee while user d/n want to exit
        cout << "\nSelect an option: \n";

        selectedEmployee.printInfo();           //print name & other basic info for employee

        cout << "1. Add a time card\n";
        cout << "2. Back\n";

        cin >> select;

        switch(select)                          //different cases based on user selection
        {
            case 1:                             //user specifies timecard file to give to employee
                cout << "Please enter the name of the file with the file extension: ";
                cin >> fileName;
                selectedEmployee.readTimeCards(fileName, supervisor);
                break;

            case 2:                             //exits out of the current menu
                return;

            default:                            //otherwise input cannot be understood & re-prompt
                "\nCould not understand input...please try again: \n";
                cin >> select;
        }

    } while (select != 2);
}
```

```
void supervisorEmployeeOptions(Employee currentEmployees[], Queue & supervisor)
//PRE:  currentEmployees[0..MAX_EMPLOYEES] has been initialized
//      supervisor has been initialized
//POST: Each employee's name has been displayed and the user has been prompted for
//          input on which employee to select or to exit the menu
{
    int select;                                     //the option selected by the user

    do                                              //continue to prompt the user until they want
    {                                               //  to go back in the menu system
         cout <<"\tSelect an Employee: \n";

         for (int i = 0; i < MAX_EMPLOYEES; i++)         //display each employee in the supervisor's
         {                                               //   Employee array by name
            cout << "\t" << i + 1 << ". " << currentEmployees[i].getName() << endl;
         }

        cout << "\t" << MAX_EMPLOYEES + 1 << ". Back\n";

        cout << "\t";
        cin >> select;

        if (select > MAX_EMPLOYEES + 1 || select <= 0)                         //only usable input
            cout << "Could not understand input...please try again: ";          //1..MAX_EMPLOYEES+1

        else if (select < (MAX_EMPLOYEES + 1))                                 //if selection is an
            supervisorCheckEmployee(currentEmployees[select - 1], supervisor);  //  employee go to
                                                                               //  specific menu
        else                                                                   //otherwise exit menu
            return;

    } while (select != MAX_EMPLOYEES + 1);
}

void supervisorCheckQueue(Queue & supervisor)
//PRE:  supervisor has been initialized
//POST: pending queue has been displayed to the user and each timecard has been prompted for
//          approval and been removed from the queue.
{
    char approved;                                //the supervisor's option to approve the time card

    if (supervisor.isEmpty() == true)            //if the queue is empty nothing needs to execute
    {
        cout << "\nThe pending queue is empty.\n";
        return;
    }
    else                                        //otherwise display queue
    {
        while (supervisor.isEmpty() == false)   //while the queue isn't empty, display the next card
        {                                       //   in the queue
            (*supervisor.viewFront()).viewCard();
            cout << "Approve Card? (Y/N)\n";

            cin >> approved;
            approved = toupper(approved);       //make sure all input is uppercase

            switch (approved)                   //diferent cases based on if the timecard is approved
            {                                   //    or not
                case 'Y':                       //if approved set approval to true and dequeue
                    (*supervisor.dequeue()).setApproval(true);
                    break;

                case 'N':                       //if not approved leave approval false and dequeue
                    (*supervisor.dequeue()).setApproval(false);
                    break;

                default:                       //if input not in y/n format then re-prompt
                    cout << "Could not understand input...please try again: ";
                    cin >> approved;
            }
        }
    }
}
```

```
void supervisorMainMenu(Employee currentEmployees[], Queue & supervisor)
//PRE:  currentEmployees[0..MAX_EMPLOYEES] has been initialized
//      supervisor has been initialized
//POST: main menu displaying options to check queue, check employees, and quit program have been
//          displayed && user has been prompted for input
{
    int select;                                 //the option selected by the user

    do                                          //a loop to continue the menu while the user has
    {                                           //    not exited.
        cout << "Enter Menu Numbers: \n";
        cout << "1. Check Queue\n";
        cout << "2. Check Employees\n";
        cout << "3. Quit\n";

        cin >> select;

        switch(select)                          //test cases corresponding to the menu numbers of
        {                                       //      the prompt
            case 1:                             //send superviosor to check the queue
                supervisorCheckQueue(supervisor);
                break;

            case 2:                             //send supervisor to the employee options menu
                supervisorEmployeeOptions(currentEmployees, supervisor);
                break;

            case 3:                             //exit the program
                return;

            default:                            //if select is not 1..3 then re-try input
                cout << "Could not understand input...please try again: ";
                cin >> select;
        }
    } while (select != 3);
}

int main()
{
    Queue supervisor;                                           //the supervisor's queue to maintain

    Employee currentEmployees[MAX_EMPLOYEES];                   //the supervisor's Employee array

    //hardcode 5 employees for supervisor
    currentEmployees[0].setName("Chris Carney");
    currentEmployees[0].setTitle("Software Engineer");

    currentEmployees[1].setName("Dylan Fetch");
    currentEmployees[1].setTitle("Software Engineer");

    currentEmployees[2].setName("John Smith");
    currentEmployees[2].setTitle("Custodian");

    currentEmployees[3].setName("Andrew Ryan");
    currentEmployees[3].setTitle("Marketing Manager");

    currentEmployees[4].setName("Michael James");
    currentEmployees[4].setTitle("Finance Intern");

    supervisorMainMenu(currentEmployees, supervisor);           //run the payroll software

    return 0;
}
```

Sample Run

```
Enter Menu Numbers:
1. Check Queue
2. Check Employees
3. Quit
1
```

```
The pending queue is empty.
Enter Menu Numbers:
1. Check Queue
2. Check Employees
3. Quit
2
        Select an Employee:
        1. Chris Carney
        2. Dylan Fetch
        3. John Smith
        4. Andrew Ryan
        5. Michael James
        6. Back
        1

Select an option:
Chris Carney | Software Engineer | Hourly Wage: $7.25 | Current Timecards: 0
1. Add a time card
2. Back
1
Please enter the name of the file with the file extension: pendingcard.txt

Select an option:
Chris Carney | Software Engineer | Hourly Wage: $7.25 | Current Timecards: 1
1. Add a time card
2. Back
1
Please enter the name of the file with the file extension: pendingcard1.txt

Select an option:
Chris Carney | Software Engineer | Hourly Wage: $7.25 | Current Timecards: 2
1. Add a time card
2. Back
2
        Select an Employee:
        1. Chris Carney
        2. Dylan Fetch
        3. John Smith
        4. Andrew Ryan
        5. Michael James
        6. Back
        2

Select an option:
Dylan Fetch | Software Engineer | Hourly Wage: $7.25 | Current Timecards: 0
1. Add a time card
2. Back
1
Please enter the name of the file with the file extension: pendingcard2.txt

Select an option:
Dylan Fetch | Software Engineer | Hourly Wage: $7.25 | Current Timecards: 1
1. Add a time card
2. Back
2
        Select an Employee:
        1. Chris Carney
        2. Dylan Fetch
        3. John Smith
        4. Andrew Ryan
        5. Michael James
        6. Back
        3

Select an option:
John Smith | Custodian | Hourly Wage: $7.25 | Current Timecards: 0
1. Add a time card
2. Back
1
Please enter the name of the file with the file extension: dssdjfkks.dar
```

```
Select an option:
John Smith | Custodian | Hourly Wage: $7.25 | Current Timecards: 0
1. Add a time card
2. Back
2
        Select an Employee:
        1. Chris Carney
        2. Dylan Fetch
        3. John Smith
        4. Andrew Ryan
        5. Michael James
        6. Back
        6
Enter Menu Numbers:
1. Check Queue
2. Check Employees
3. Quit
1

Pay Period Start Date:12/05/2014
Approved?: false
Hours Worked Per Day:
   Sunday:     0.00
   Monday:     8.00
  Tuesday:     7.00
Wednesday:     5.00
 Thursday:     6.00
   Friday:     8.00
 Saturday:     0.00

Approve Card? (Y/N)
Y

Pay Period Start Date:03/08/2014
Approved?: false
Hours Worked Per Day:
   Sunday:     0.00
   Monday:     2.00
  Tuesday:     3.00
Wednesday:     4.00
 Thursday:     1.00
   Friday:     7.00
 Saturday:     0.00

Approve Card? (Y/N)
Y

Pay Period Start Date:02/12/2014
Approved?: false
Hours Worked Per Day:
   Sunday:    10.00
   Monday:     1.00
  Tuesday:     1.00
Wednesday:     4.00
 Thursday:     1.00
   Friday:     8.00
 Saturday:     8.00

Approve Card? (Y/N)
n
Enter Menu Numbers:
1. Check Queue
2. Check Employees
3. Quit
1

The pending queue is empty.
Enter Menu Numbers:
1. Check Queue
2. Check Employees
3. Quit
3
```

Loop Analysis

TimeCard.cpp
This loop is within the `weekPay` function of the TimeCard class. It is the meat of the function as it is what actually does the work. The loop goes through the `hours[]` array one day at a time, multiplies the hours on a day by the wage of the employee, and adds it to the `weeklyPay` variable. The loop itself is as follows:

```
for (int i = 0; i < 7; i++)
{
      weeklyPay += hours[i] * wage;
}
```

This loop is right for this situation because it is a determinate loop. We know the number of indexes in the array, so a `for` loop is preferable to a while loop in this situation. An invariant for this loop is: At the start of the i^th iteration, `weeklyPay` is equal to the sum of the hours from `hours[0..i-1]*wage`.

Employee.cpp
This loop is within the `getGrossPay` function of the Employee class. It is used to advance through an array of timecards, calling the `weekPay` function for each timecard, and adding it to the variable `grossPay`. The loop is as follows:

```
for (int i = 0; i < currentCards; i++)
{
      grossPay += (*EmployeeCards[i]).weekPay(hourlyWage);
}
```

This loop is right for the situation because it is a determinate loop. We know the size of the array we want to advance through, so a `for` loop is the best option. An invariant for this loop is: At the start of the i^th iteration, `grossPay` is the sum of the weekly pay from `EmployeeCards[0..i-1]`.

Discussion

This lab helped to give us experience writing a large program with multiple files, multiple classes, and a user interface. This program was much larger and more involved than previous programs. This was also our first computer science group project, which gives us experience working with others and doing things like version control and sharing files. Most importantly, we learned how to organize a large project by breaking it down into various classes and using those classes inside other classes to create a "hierarchy" in the program. We also gained a huge insight into the real value of pointers and what it truly means to have an object exist only once in memory.

Some difficulties we ran into dealt with pointers and passing by reference. Making an array of pointers to timecards and a queue of pointers to the same timecards was a challenge. We had trouble determining the correct way of going about using pointers in certain situations, such as when to dereference and when to set a pointer equal to another pointer. Returning a pointer to an object from a function was also something that accomplished that we had never done before. Another problem we faced was passing arguments by reference. *Not* passing an object by reference into a function caused us many bugs that were hard to detect, but after using the debugger in Visual Studio to view the state of objects throughout the program we were able to figure out where to add an ampersand and the bugs were fixed with a single character. Another challenge we faced was implementing a queue via a class instead of just procedurally like we had learned in class. Determining how to define the node structure was a process of trial and error but it worked perfectly after drawing a few diagrams. The zero-th law of pointers definitely helped us greatly in this project.

Some limitations of the software are that it can only read one file at a time of a specific format, and the fact that the menu is through the console and not through a GUI. An improvement to this software would be the ability to input multiple files at a time, possibly choosing them rather than inputting the file name. Also being able to read multiple time cards from one file instead of having a "one timecard for one file rule". A GUI would vastly improve this program. A console based menu can be confusing; the ability to click on things is very helpful to most people. Having a friendlier user interface would greatly improve the user experience for the supervisor.

Teamwork was very important and we executed it very well; we made a plan and listed the things we needed to do the first day and were able to follow that plan to the end. Our ideas worked well together, and we didn't disagree about anything major and were able to resolve differences in coding style. Communication and determining when to we had the time to work together and when to work apart was crucial. It was the first time we had worked with someone else's code in another project but the way our classes were implemented it ultimately was very easy to combine the best aspects of our two labs.