CMPSC 122 Project 1 Back End 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 corresponding 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
```

TimeCard.cpp

```
// Programmer:   Dylan Fetch & Christopher Carney
// Section:      2
// Lab:          Project 1
// Date:         March 6, 2014
// Description:  Implementation file for timeCard class

#include "TimeCard.h"
#include <iomanip>

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

    for(int i = 0; i <= 6; i++)               //determinate loop to set hours[0..6] to zero
    {
          hours[i] = 0;
     }

    isApproved = false;
}

TimeCard::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
{
    pMonth = initMonth;
    pDay = initDay;
    pYear = initYear;

    for(int i = 0; i <= 6; i++)               //determinate loop to set hours[0..6] to zero
    {
        hours[i] = 0;
    }

    isApproved = false;
}

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

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

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

void TimeCard::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
{
    hours[day] = workHours;
}
```

```
void TimeCard::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
{
    isApproved = approval;
}

bool TimeCard::getApproval() const
//POST:   FCTVAL == 0 if unapproved || 1 if approved
{
    return isApproved;
}

double TimeCard::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 specified day
{
    return hours[day];
}

void TimeCard::printDate() const
//POST:   pay period start date from this TimeCard has been printed in MM/DD/YYYY format
{
    cout << setw(2) << setfill('0') << pMonth << "/" << setw(2) << pDay << "/" << setw(4) << pYear;
}

double TimeCard::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
{
    double weeklyPay;                         //the weekly pay of the worker in dollars

    weeklyPay = 0;

    for (int i = 0; i < 7; i++)             //goes through the hours[0..6] array and multiplies the
    {                                       //  wage by the hours worked that day to get the weekly pay
        weeklyPay += hours[i] * wage;
    }

    return weeklyPay;
}

void TimeCard::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]
{
    cout << endl << "Pay Period Start Date:";
    printDate();
    cout << endl;

    cout << "Approved?: " << boolalpha << isApproved << endl;
    cout << setfill(' ');
    cout << "Hours Worked Per Day:"<<endl;
    cout << setw(10) << "Sunday:" << setw(9) << fixed << setprecision(2) << hours[0] << endl;
    cout << setw(10) << "Monday:" << setw(9) << fixed << setprecision(2) << hours[1] << endl;
    cout << setw(10) << "Tuesday:" << setw(9) << fixed << setprecision(2) << hours[2] << endl;
    cout << setw(10) << "Wednesday:" << setw(9) << fixed << setprecision(2) << hours[3] << endl;
    cout << setw(10) << "Thursday:" << setw(9) << fixed << setprecision(2) << hours[4] << endl;
    cout << setw(10) << "Friday:" << setw(9) << fixed << setprecision(2) << hours[5] << endl;
    cout << setw(10) << "Saturday:" << setw(9) << fixed << setprecision(2) << hours[6] << endl;
    cout << endl;
}
```

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 dollar/hour, 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/hour
        //          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
```

Employee.cpp

```
//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.

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

using namespace std;

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

    currentCards = 0;
    //EmployeeCards[16] -- implicit default constructor call
}

Employee::Employee(string initName, string initJobTitle, double initWage)
//PRE:  initName has been initialized, initJobTitle has been initialized, initWage has been
//          initialized and is in dollars/hour
//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
{
    name = initName;
    jobTitle = initJobTitle;
    hourlyWage = initWage;

    currentCards = 0;
    //EmployeeCards[16] -- implicit default constructor call
}

void Employee::giveTimeCard(TimeCard * newCard)
//POST: a pointer to a TimeCard has been added to this Employee's timecards array
{
    if (currentCards < MAX_CARDS)
    {
        EmployeeCards[currentCards] = newCard;        //give this employee the new TimeCard object
        currentCards++;
    }
}

void Employee::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
{
    hourlyWage = newWage;
}

void Employee::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
{
    name = newName;
}

void Employee::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
{
    jobTitle = newTitle;
}
```

```
void Employee::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
{
    bool fileApproval;      //the approval status of the timecard from the file
    int fileMonth;          //the starting month of the pay period from the file
    int fileDay;            //the starting day of the pay period from the file
    int fileYear;           //the starting year of the pay period from the file

    double fileHours;       //the temporary variable to read in hours from the file
    int counter;            //a counter to keep track of the current day to read in fileHours

    if (currentCards < MAX_CARDS)               //only read in files if there is room in the array
    {
        TimeCard * newCard = new TimeCard;      //create a new pointer to a TimeCard to add to
                                                //    EmployeeCards[]

        ifstream inFile(fileName);              //open the specified file

        if (inFile == false)                    //if the file cannot be read then return
            return;

        inFile >> fileApproval >> fileMonth >> fileDay >> fileYear;   //read in values to the temp file
                                                                      //    variables
        (*newCard).setApproval(fileApproval);                         //set the file Approval status
        (*newCard).setMonth(fileMonth);                               //set the starting month
        (*newCard).setDay(fileDay);                                   //set the starting day
        (*newCard).setYear(fileYear);                                 //set the starting year

        int counter = 0;                                        //start the counter on SUNDAY
        while(inFile >> fileHours)                              //read in while there are values in
        {                                                       //    the file.
            (*newCard).setHours(fileHours, counter);            //set the hours of the current day to
            counter++;                                          //    what they are in the file
        }

        if((*newCard).getApproval() == false)                  //if the file is not approved pointer must
        {                                                       //     be added to the supervisor queue
            supervisor.enqueue(newCard);
        }

        EmployeeCards[currentCards] = newCard;                //add TimeCard pointer to this Employee's
                                                              //      TimeCard pointer array
        currentCards++;

        inFile.close();                                      //close the file
    }
}

int Employee::getNumberOfCards() const
//POST: FCTVAL = currentCards, the number of TimeCards the employee currently has
//          (the logical size of the TimeCards[] array)
{
    return currentCards;
}

string Employee::getName() const
//POST: FCTVAL == name, the name of this employee
{
    return name;
}
```

```
double Employee::getGrossPay() const
//POST: FCTVAL = total pay across all timecards from EmployeeCards[0..currentCards-1] in
//          dollars
{
    double grossPay;                            //the money made accross all time cards in dollars

    grossPay = 0;                               //the initial gross pay is $0

    for (int i = 0; i < currentCards; i++)                  //go through all the cards this Employee
    {                                                       //  has, calculating weekly pay
                                                            //  && adding to gross pay
        grossPay += (*EmployeeCards[i]).weekPay(hourlyWage);
    }

    return grossPay;
}

double Employee::getHourlyWage() const
//POST: FCTVAL == hourly wage in dollars/hour
{
    return hourlyWage;
}

void Employee::printInfo() const
//POST: Basic info about this Employee has been printed in the format:
//      [name] | [jobTitle] | Hourly Wage: $[hourlyWage] | Current Timecards: [currentCards]
{
    cout << name << " | " << jobTitle << " | Hourly Wage: $";           //details of the employee
    cout << setprecision(2) << fixed << hourlyWage;                     //formatted currency value
    cout << " | " << "Current Timecards: " << currentCards << endl;
}

void Employee::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]
{
    printInfo();                                //print basic heading info a/b employee

    for (int i = 0; i < currentCards; i++)      //start at the first TimeCard and go from
    {                                           //EmployeeCards[0..currentCards-1] & output all info
        cout << "Card #" << i+1 << endl;        //output which card it is about to display
        (*EmployeeCards[i]).viewCard();         //prints all the info of the card
        cout << endl << endl;
    }
}
```

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
```

Queue.cpp

```
// Programmer:     Dylan Fetch & Christopher Carney
// Section:        2
// Lab:            Project 1
// Date:           March 6, 2014
// Description:    Custom Queue Class implementation for use with TimeCard pointers

#include "Queue.h"

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

```
int Queue::logSize() const
//POST: FCTVAL == the number of NodeType nodes contained in the queue starting at front &
//      ending at back
{
    NodePtr cur;                            //a pointer to a NodeType node to traverse the queue
    int logSize;                            //the logical size of the queue

    logSize = 0;                            //initialize the logical size to start at 0
    cur = front;                            //set pointer to start at the head

    while (cur != NULL)                     //starting at the front of the queue, increment the logical
    {                                       //    size then go to the next node until it reaches NULL
        cur = cur->next;                    //point cur to the next node in the list
        logSize++;
    }

    return logSize;
}

bool Queue::isEmpty() const
//POST: FCTVAL == true (1) if the queue is empty || false (0) if the queue contains a timecard
{
    int logicalSize;                       //the logical size of current queue

    logicalSize = logSize();                //check the logical size of the current queue and set it to
                                            //    the var logicalSize

    if (logicalSize == 0)                   //the queue is empty if the logical size is 0
        return true;
    else                                    //otherwise the logical size is > 0 so queue contains values
        return false;
}

void Queue::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
{
    NodePtr temp;                   //a temporary node to store the value to be inserted

    if (front == NULL)              //if the front of the list points to nothing then it is an empty
    {                               //     queue
        back = new NodeType;        //a new node for the back of the queue to hold new TimeCard

        back->timecard = k;         //TimeCard in queue is inserted at the back
        back->next = NULL;          //the back points to NULL pointer

        front = back;               //since the list is empty the back is the front
    }
    else                            //otherwise the queue already has values in it
    {

        temp = new NodeType;        //a new temp node to hold the new TimeCard

        temp->timecard = k;         //store new TimeCard to be queued in temp
        temp->next = NULL;          //temp's successor is a NULL pointer (the new back of queue)

        back->next = temp;          //point back's next (old back of queue) to temp (new back)
        back = temp;                //the new back is the temp node (so the back is always known)

        temp = NULL;                //temp pointer is no longer needed b/c back is what temp was
    }

    return;
}
```

```
TimeCard* Queue::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
{
    NodePtr temp;                   //a pointer to the front of the queue to be deleted

    TimeCard* tempCard;             //a temporary pointer to a timecard object

    temp = front;                   //temp is whatever the front is
    front = temp->next;             //front now points to temp's successor (the new front)

    temp->next = NULL;              //set temp's next to NULL to avoid dangling pointer

    tempCard = temp->timecard;      //tempCard is a pointer to temp's TimeCard (what is to be returned)

    delete temp;

    return tempCard;
}

TimeCard* Queue::viewFront() const
//POST: FCTVAL == the pointer to the timecard at the front of the queue
{
    return front->timecard;
}
```