Sarah Wood

C++ Part 2

Final Project

“Crew Chief’s / Steward’s Payroll Tools”

Initial Requirements

**Date Approved:** Tom approved this project on March 30, 2016.

**Application Title:** “Steward’s Payroll Tools”

**Purpose:** To simplify the payroll part of a Crew Chief’s duties. Reduce payroll errors, provide accurate information in a timely manner (time is CRITICAL in the field, and always in short supply).

**Procedures:**

* Import RATES & CONDITIONS (extracted from CONTRACT)
* make a CREW LIST, export it to WORKSHEET (spreadsheet or text) to collect field values
* import filled-out WORKSHEET, process hours, filter for Straight Time, OT, DT, &c as per CONDITIONS
* generate TIME SHEET - mostly a double-checking of hours breakdown, detailed
* generate PAYROLL INVOICE – in a format compatible with existing invoice spreadsheet, output to screen for error-checking
* estimate CONTINGENCY planning – If we keep the crew longer, how much will it cost per hour?

**Algorithms, Processing and Conditions:**

Get CONDITIONS & RATES – File input into structured records

Make EMPLOYEE objects (must be objects to accommodate special pay conditions)

(early) Manual entry of daily work – console I/O

(later) file output of daily WORKSHEET, import & processing of filled-out daily WORKSHEET

Compute each EMPLOYEE’S daily pay

Compute bottom-line INVOICE TOTAL (just the labor cost, before benefits)

Estimate “What-if” cost for management planning

Output TIME breakdown

Output PAYROLL INVOICE to screen

Output PAYROLL INVOICE to be compatible with existing INVOICE spreadsheet

**Notes/Restrictions:**

MUST be: easy to use, thoroughly labelled, fast

**Comments:**

The tedious data entry was mostly eliminated. Getting field values for time was streamlined with the WORKSHEET. It’s in a familiar format, and also provides data that’s useful on-site. (Can also be printed and stuck to a clipboard, or used with a tablet. It’s not program dependent once it’s generated.) Whichever spreadsheet program our Steward uses gives them the ability to copy & paste repetitive values. Importing that WORKSHEET back into the PROGRAM gives us our figures to use in calculating pay.

Initial Notes:

Why do we need this software? **(USER STORIES)**

“As a crew chief, I need to spend less time preparing payroll, so that I can attend to all my other duties.”

“As an employee, I need to see my payroll figures before I leave for the day, so that I know what my paycheck should look like.”

“As an employer’s production manager, I need to know costs, so that I can make good decisions.”

“As a crew chief, I need to be able to deliver payroll before I leave for the day—no matter how hectic the day’s been.”

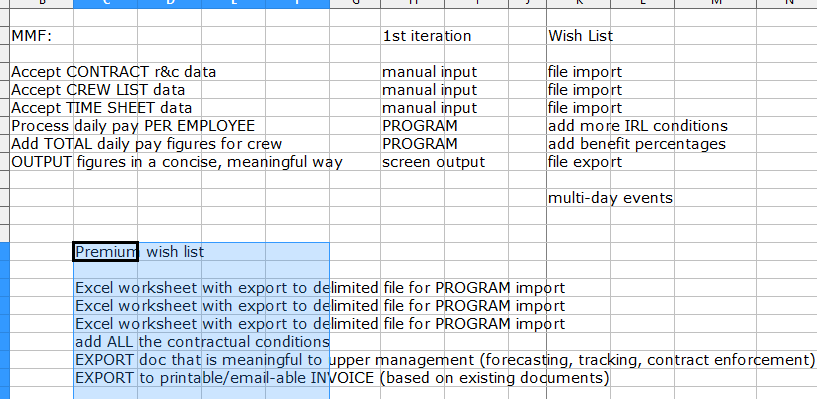
“As the payroll department, we need to avoid late or mistaken paychecks for our crew, so that we don’t have people angry with us (or have employees in a hardship because they didn’t get paid correctly on time).”

“As a business agent, I need to be able to keep my stewards up-to-date on current rates and conditions, so that field-work is contract compliant.”

Minimal Marketable Features:

* Accept rates & conditions from file import
* Accept crew list (with job classification assignments) (could be manually entered, file I/O is wish-listed)
* Accept hours/shows for each employee
* Process daily pay for each employee accurately
* Add total pay figures for crew
* Display figures in a meaningful way

Some of my early notes:



Burn Down Quick Notes

1.0: 5 pts

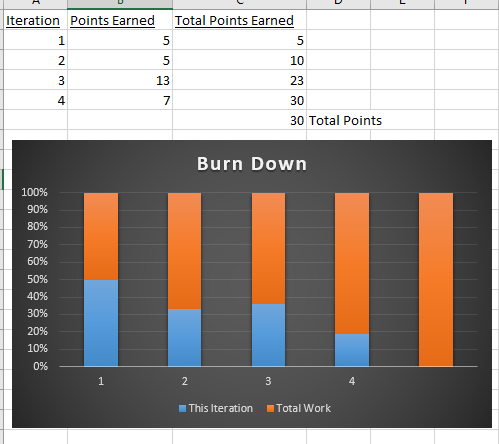
2.0: 5 pts (+5pts = 10pts)

3.0: 13 pts (+10pts = 23 pts)

4.0: 7 pts (+23 pts = 30 pts)

It looks like I need to re-evaluate the point values for my early work. Iteration 3.0 moved quickly because of foundational work in 1.0 and 2.0.

Burn Down Chart



**Programming Journal**

Pre-Iteration

read Agile Samurai Planning Section

buy index cards

Review IRL contracts, documents, & field assets

Extract relevant data for program

Make notes, draw diagrams, plan program

Iteration 1.0

Make up index cards, plan program, assign point values

Initial planning (3 points)

Build Labor Pool (1 point)

Skeleton Program code (1 point)

5 points earned

Iteration 2.0

Build Employee class & member functions (3 points)

Deep research on contract & relevant data (1 point)

Rates & conditions (build structures, build file imports) (1 point)

5 points earned

Iteration 3.0

Preliminary UI (3 points)

I minimized repetitive data entry by building a worksheet to be filled out onsite

Diagram & program rates & conditions (5 points)

Build working time sheet ! (5 points)

It started to get exciting at this point. <Frankenstein> IT'S ALIVE!!!</Frankenstein>

13 points earned

Iteration 3.5

Compute Daily pay per employee (3 points)

Iteration 4.0

Estimate contingency costs (1 point)

(Management often wants a cost estimate to decide whether to keep the crew longer. It needs to be based on the current rate condition: Straight time, Overtime, &c. )

Display /File export Daily Payroll Invoice (3 points)

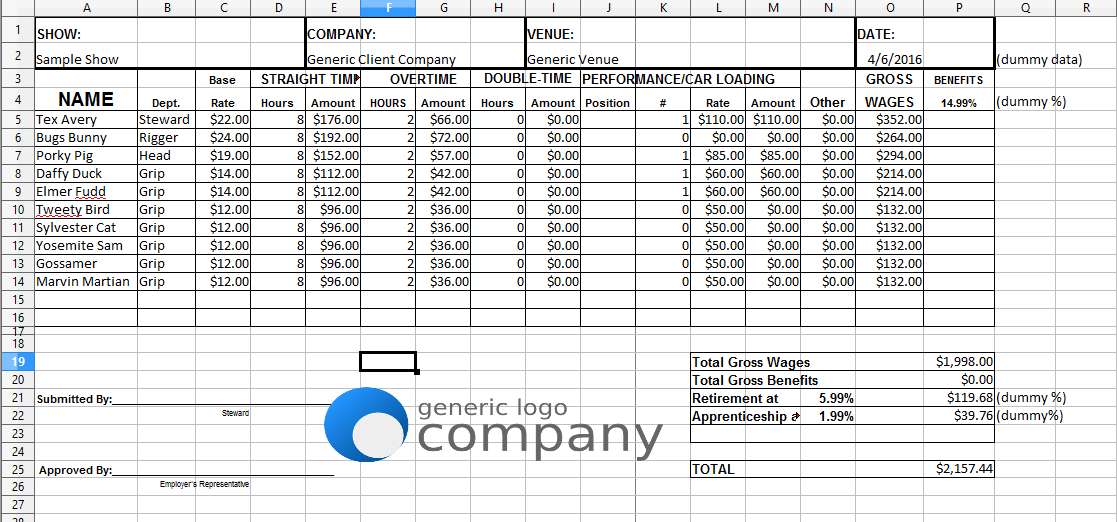
7 points earned

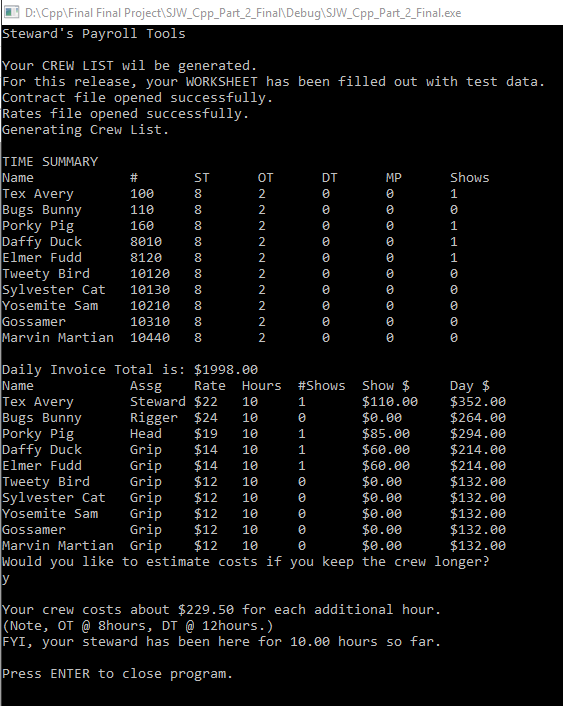
It works! It processes crew data, and gives a time sheet and payroll in seconds, whereas it takes an hour or two to do it by hand. For 20-200 employees, it's a full-time gig. I could reduce that to just a few minutes.

**Screen Caps**

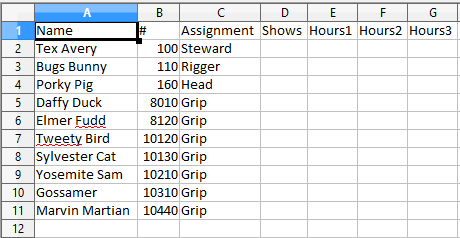
This is the final product. My data imported cleanly into the familiar invoice spreadsheet!

(All the calculations were done in my program, up to each employee’s GROSS WAGES. The TOTAL GROSS in the spreadsheet matched my in-program sum. Calculating benefits was not critical at this time.)

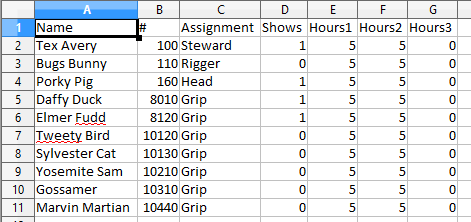




Here is what a CREW LIST or unfilled WORKSHEET looks like



This is how it looks when it imports back into the program with field values entered



The TIME SHEET is simply a file output of what is displayed on the console screen as TIME SUMMARY

Source Code

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Employee.h

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#pragma once

#ifndef EMPLOYEE\_H

#define EMPLOYEE\_H

#include <iostream>

#include <fstream>

#include <string>

#include <ctime> //raw time entry might come in a later release

#include <time.h>

#include "Structures.h"

using namespace std;

//make an Employee class

//employee specialties may inherit from this class as needed

const int DAY = 32;

class Employee

{

protected:

public:

int employeeNumber;

string employeeName;

string employeeBaseQualification;

string employeeBaseQualificationSHORT;

string employeeDailyAssignment;

//we can add other qualifications as needed

//employee time-related members

double employeeDailySTHours;

double employeeDailyOTHours;

double employeeDailyDTHours;

double employeeDailyMPHours;

double employeeDailyShows;

double employeeDailyTotalHours;

//employee pay-related members

double employeeApplicableRateHourly;

double employeeApplicableRateShow;

double employeeDailySTPay;

double employeeDailyOTPay;

double employeeDailyDTPay;

double employeeDailyMPPay;

double employeeDailyShowPay;

double employeeDailyOtherPay;

double employeeDailyTotalPay;

/\* will track ongoing employee hours later

(adapt from part 1 final project)

double employeeDailySTHours[DAY];

double employeeDailyOTHours[DAY];

double employeeDailyDTHours[DAY];

double employeeDailyMPHours[DAY];

double employeeDailyShows[DAY];

double employeeDailySTPay[DAY];

double employeeDailyOTPay[DAY];

double employeeDailyDTPay[DAY];

double employeeDailyMPPay[DAY];

double employeeDailyShowPay[DAY];

double employeeDailyOtherPay[DAY];

\*/

//default constructor

Employee()

{

employeeNumber = 99999;

employeeName = " ";

employeeBaseQualification = "Hand";

employeeBaseQualificationSHORT = "H";

employeeDailyAssignment = "Grip";

//employee time-related members

employeeDailySTHours = 0;

employeeDailyOTHours = 0;

employeeDailyDTHours = 0;

employeeDailyMPHours = 0;

employeeDailyShows = 0;

employeeDailyTotalHours = 0;

//employee pay-related members

employeeApplicableRateHourly = 0;

employeeApplicableRateShow = 0;

employeeDailySTPay = 0;

employeeDailyOTPay = 0;

employeeDailyDTPay = 0;

employeeDailyMPPay = 0;

employeeDailyShowPay = 0;

employeeDailyOtherPay = 0;

employeeDailyTotalPay = 0;

/\* will track ongoing employee pay later

(adapt from Part 1 final project)

employeeDailySTHours[DAY] = {};

employeeDailyOTHours[DAY] = {};

employeeDailyDTHours[DAY] = {};

employeeDailyMPHours[DAY] = {};

employeeDailyShows[DAY] = {};

employeeDailySTPay[DAY] = {};

employeeDailySTPay[DAY] = {};

employeeDailySTPay[DAY] = {};

employeeDailySTPay[DAY] = {};

employeeDailyShowPay[DAY] = {};

employeeDailyOtherPay[DAY] = {};

\*/

}

//contructor

Employee(int number, string name, string qualification, string shortQual, string assign)

{

employeeNumber = number;

employeeName = name;

employeeBaseQualification = qualification;

employeeBaseQualificationSHORT = shortQual;

employeeDailyAssignment = assign;

}

//mutators & accessors for basic info

void setEmployeeNumber(int);

void setEmployeeName(string);

void setEmployeeQualification(string);

void setEmployeeQualificationSHORT(string);

void setEmployeeDailyAssignment(string);

int getEmployeeNumber();

string getEmployeeName();

string getEmployeeQualification();

string getEmployeeQualificationSHORT();

string getEmployeeDailyAssignment();

//mutators & accessors for hours

void setEmployeeDailyWork(EmployerContract, EmployerJobPayRates, Employee);

void setEmployeeDailySTHours(double);

void setEmployeeDailyOTHours(double);

void setEmployeeDailyDTHours(double);

void setEmployeeDailyMPHours(double);

void setEmployeeDailyShows(double);

void setEmployeeDailyTotalHours(double);

double getEmployeeDailySTHours();

double getEmployeeDailyOTHours();

double getEmployeeDailyDTHours();

double getEmployeeDailyMPHours();

double getEmployeeDailyShows();

double getEmployeeDailyTotalHours();

//mutators & accessors for pay

void setEmployeeApplicableRateHourly(double);

void setEmployeeApplicableRateShow(double);

void setEmployeeDailySTPay(double);

void setEmployeeDailyOTPay(double);

void setEmployeeDailyDTPay(double);

void setEmployeeDailyMPPay(double);

void setEmployeeDailyShowPay(double);

void setEmployeeDailyOtherPay(double);

void setEmployeeDailyPayTotal(double);

double getEmployeeApplicableHourlyRate();

double getEmployeeApplicableShowRate();

double getEmployeeDailySTPay();

double getEmployeeDailyOTPay();

double getEmployeeDailyDTPay();

double getEmployeeDailyMPPay();

double getEmployeeDailyShowPay();

double getEmployeeDailyOtherPay();

double getEmployeeDailyPayTotal();

//other member functions as needed

//destructor declaration

~Employee();

};

#endif // !EMPLOYEE\_H

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Employee.cpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <iostream>

#include <fstream>

#include <string>

#include <cstdlib>

#include "Employee.h"

#include "Structures.h"

using namespace std;

//mutators & accessors for basic info

void Employee::setEmployeeNumber(int number)

{

employeeNumber = number;

}

void Employee::setEmployeeName(string name)

{

employeeName = name;

}

void Employee::setEmployeeQualification(string qual)

{

employeeBaseQualification = qual;

}

void Employee::setEmployeeQualificationSHORT(string qualShort)

{

employeeBaseQualificationSHORT = qualShort;

}

int Employee::getEmployeeNumber()

{

return employeeNumber;

}

string Employee::getEmployeeName()

{

return employeeName;

}

string Employee::getEmployeeQualification()

{

return employeeBaseQualification;

}

string Employee::getEmployeeQualificationSHORT()

{

return employeeBaseQualificationSHORT;

}

//Member functions related to hours

void Employee::setEmployeeDailyWork( EmployerContract contract, EmployerJobPayRates rates, Employee emp)

{

//declare local variables

int choice, number;

double tempHours =0, tempMPHours=0;

char yesNo2;

bool valid = false;

//select a classification for [SELECTED EMPLOYEE]

cout << '\n' << "Please select a job classification for "<<emp.employeeName<<" :" << '\n';

cout << "1 Steward" << '\t' << "2 Rigger" << '\t'<<"3 Specialist"<<'\n';

cout << "4 Head"<<'\t'<<'\t' << "5 Programmer" << '\t'<<"6 Grip"<<'\n';

cout << "Please make a selection: " << '\t';

cin >> choice;

//validate user input

while (!cin)

{

cout << "Invalid Input. Try again. Enter #1-6: " << '\t';

cin.clear();

cin.ignore();

cin >> choice;

}

while (choice<1 || choice>6)

{

valid = !true; //declare this attampt !valid

cin.clear(); //clear the input stream

cin.ignore(); //ignore invalid stuff.

cout << "Invalid input. Try again. Enter #1-6: " << '\t'; //user prompt

cin >> choice; //try again...

}

//Assign Rate to each JobClassification choice

if (choice == 1)

{

emp.setEmployeeApplicableRateHourly(rates.StewardRateHourly);

emp.setEmployeeApplicableRateShow(rates.StewardRateShow);

}

else if (choice == 2)

{

emp.setEmployeeApplicableRateHourly( rates.RiggerRateHourly);

emp.setEmployeeApplicableRateShow ( rates.RiggerRateShow);

}

else if (choice == 3)

{

emp.setEmployeeApplicableRateHourly ( rates.SpecialistRateHourly);

emp.setEmployeeApplicableRateShow ( rates.SpecialistRateShow);

}

else if (choice == 4)

{

emp.setEmployeeApplicableRateHourly ( rates.HeadRateHourly);

emp.setEmployeeApplicableRateShow ( rates.HeadRateShow);

}

else if (choice == 5)

{

//this is where we use the base qualifications.

if (emp.employeeBaseQualificationSHORT == "J")

{

emp.setEmployeeApplicableRateHourly ( rates.JourneymanRateHourly);

emp.setEmployeeApplicableRateShow ( rates.JourneymanRateShow);

}

else if (emp.employeeBaseQualificationSHORT == "A3" || emp.employeeBaseQualificationSHORT == "A2" || emp.employeeBaseQualificationSHORT == "A1")

{

emp.setEmployeeApplicableRateHourly ( rates.ApprenticeRateHourly);

emp.setEmployeeApplicableRateShow ( rates.ApprenticeRateShow);

}

else if (emp.employeeBaseQualificationSHORT == "H")

{

emp.setEmployeeApplicableRateHourly(rates.HandRateHourly);

emp.setEmployeeApplicableRateShow ( rates.HandRateShow);

}

else

{

cout << "ERROR. Employee does not have a base qualification. Will be paid hand rate." << '\n';

emp.setEmployeeApplicableRateHourly(rates.HandRateHourly);

emp.setEmployeeApplicableRateShow(rates.HandRateShow);

}

}//close "choice == INT"

cout << "Please enter the number of shows "<<emp.employeeName<<" worked today (0 if none): " << '\t';

cin >> number;

//validate input. Max 3 shows/day before idiot-checking

while (!cin)

{

valid = !true;

cout << "Invalid Input. Try again." << '\t';

cin.clear();

cin.ignore();

cin >> number;

}

if (number>3)

{

cout << "Did they really work " << number << " shows today? Y/N?";

cin >> yesNo2;

while (yesNo2 != 'Y' && yesNo2 != 'y' && yesNo2 != 'N' && yesNo2 != 'n')

{

cout << "Invalid Input. Y or N? " << '\t';

cin.clear();

cin.ignore();

cin >> yesNo2;

}

if (yesNo2 == 'N' || yesNo2 == 'n')

{

cin.clear();

cin.ignore();

cout << "How many shows did they REALLY work?" << '\t';

cin >> number;

while (!cin)

{

cout << "Invalid Input. Try again." << '\t';

cin.clear();

cin.ignore();

cin >> number;

}

}

} //close idiot-check for number of shows worked

emp.setEmployeeDailyShows(number);

//Hourly Work? YN, if Y, get hours

cout << "Hourly work today? Y/N " << '\t';

cin >> yesNo2;

while (yesNo2 != 'Y' && yesNo2 != 'y' && yesNo2 != 'N' && yesNo2 != 'n')

{

cout << "Invalid Input. Y or N? " << '\t';

cin.clear();

cin.ignore();

cin >> yesNo2;

}

if (yesNo2 == 'Y' || yesNo2 == 'y') //yes, OK, more questions

{

cout << '\n' << "Please enter the number of hours between CALL TIME and FIRST meal break: " << '\n';

cin >> number;

while (!cin)

{

cout << "Invalid Input. Try again. " << '\t';

cin.clear();

cin.ignore();

cin >> number;

}

if (number >6)

{

cout << '\n' << "Did they really work " << number << " hours without a meal break? Y/N?";

cin >> yesNo2;

while (yesNo2 != 'Y' && yesNo2 != 'y' && yesNo2 != 'N' && yesNo2 != 'n')

{

cout << "Invalid Input. Y or N? " << '\t';

cin.clear();

cin.ignore();

cin >> yesNo2;

}

if (yesNo2 == 'N' || yesNo2 == 'n')

{

cin.clear();

cin.ignore();

cout << "How many hours did they REALLY work?" << '\t';

cin >> number;

}

}

//if <4 make equal to 4 (industry standard is 4 hour minimum)

if (number <= 4)

{

number = 4;

tempHours = number;

}

else if (number = 5)

{

tempHours = number;

}

//if >5, process Meal Penalty,

else if (number >5)

{

tempHours = number;

tempMPHours = (number - 5);

}

cout << "Please enter the number of hours between CALLBACK and SECOND meal break: " << '\n';

cout << "Enter 0 if none." << '\t';

cin >> number;

while (!cin)

{

cout << "Invalid Input. Try again. " << '\t';

cin.clear();

cin.ignore();

cin >> number;

}

if (number >6)

{

cout << "Did they really work " << number << " hours without a meal break? Y/N?";

cin >> yesNo2;

while (yesNo2 != 'Y' && yesNo2 != 'y' && yesNo2 != 'N' && yesNo2 != 'n')

{

cout << "Invalid Input. Y or N? " << '\t';

cin.clear();

cin.ignore();

cin >> yesNo2;

}

if (yesNo2 == 'N' || yesNo2 == 'n')

{

cin.clear();

cin.ignore();

cout << "How many hours did they REALLY work?" << '\t';

cin >> number;

}

}

//if >5, process Meal Penalty, if <2 make equal to 2 (industry standard is 2 hours pay after "lunch")

if (number < 2 )

{

number = 2;

tempHours = number;

}

else if (number < 5)

{

tempHours = number;

}

//if >5, process Meal Penalty,

else if (number >5)

{

tempHours = number;

tempMPHours = (number - 5);

}

cout << "Please enter the number of hours between CALLBACK and CUT: " << '\n';

cout << "Enter 0 if none." << '\t';

cin >> number;

while (!cin)

{

cout << "Invalid Input. Try again. " << '\t';

cin.clear();

cin.ignore();

cin >> number;

}

if (number >6)

{

cout << "Did they really work " << number << " hours without a meal break? Y/N?";

cin >> yesNo2;

while (yesNo2 != 'Y' && yesNo2 != 'y' && yesNo2 != 'N' && yesNo2 != 'n')

{

cout << "Invalid Input. Y or N? " << '\t';

cin.clear();

cin.ignore();

cin >> yesNo2;

}

if (yesNo2 == 'N' || yesNo2 == 'n')

{

cin.clear();

cin.ignore();

cout << "How many hours did they REALLY work?" << '\t';

cin >> number;

}

}

//if >5, process Meal Penalty, if <2 make equal to 2 (industry standard is 2 hours pay after "lunch")

if (number < 2)

{

number = 2;

tempHours = number;

}

else if (number <5)

{

tempHours = number;

}

else if (number >5)

{

tempHours = number;

tempMPHours = (number - 5);

}

emp.setEmployeeDailyTotalHours(tempHours);

//if total >8, process OT, if >12, process DT

if (emp.employeeDailyTotalHours > 12)

{

emp.setEmployeeDailySTHours(8);

emp.setEmployeeDailyOTHours(4);

emp.setEmployeeDailyDTHours (tempHours - 12);

emp.setEmployeeDailyMPHours(tempMPHours);

}

else if (emp.employeeDailyTotalHours>8)

{

emp.setEmployeeDailySTHours(8);

emp.setEmployeeDailyOTHours (tempHours - 8);

emp.setEmployeeDailyMPHours(tempMPHours);

}

else

{

emp.setEmployeeDailySTHours (tempHours);

emp.setEmployeeDailyMPHours(tempMPHours);

}

} //close daily hourly input ( IF YES)

else if (yesNo2 == 'n' || yesNo2 == 'N')

cout << "No hourly pay today." << '\n';

}

void Employee::setEmployeeDailySTHours(double st)

{

employeeDailySTHours = st;

}

void Employee::setEmployeeDailyOTHours(double ot)

{

employeeDailyOTHours = ot;

}

void Employee::setEmployeeDailyDTHours(double dt)

{

employeeDailyDTHours = dt;

}

void Employee::setEmployeeDailyMPHours(double mp)

{

employeeDailyMPHours = mp;

}

void Employee::setEmployeeDailyShows(double shows)

{

employeeDailyShows = shows;

}

void Employee::setEmployeeDailyTotalHours(double total)

{

employeeDailyTotalHours = total;

}

double Employee::getEmployeeDailySTHours()

{

return employeeDailySTHours;

}

double Employee::getEmployeeDailyOTHours()

{

return employeeDailyOTHours;

}

double Employee::getEmployeeDailyDTHours()

{

return employeeDailyDTHours;

}

double Employee::getEmployeeDailyMPHours()

{

return employeeDailyMPHours;

}

double Employee::getEmployeeDailyShows()

{

return employeeDailyShows;

}

double Employee::getEmployeeDailyTotalHours()

{

return employeeDailyTotalHours;

}

//member functions related to pay

void Employee::setEmployeeApplicableRateHourly(double hrRate)

{

employeeApplicableRateHourly = hrRate;

}

void Employee::setEmployeeApplicableRateShow(double showRate)

{

employeeApplicableRateShow = showRate;

}

void Employee::setEmployeeDailyAssignment(string assign)

{

employeeDailyAssignment = assign;

}

void Employee::setEmployeeDailySTPay(double st)

{

employeeDailySTPay = st;

}

void Employee::setEmployeeDailyOTPay(double ot)

{

employeeDailyOTPay = ot;

}

void Employee::setEmployeeDailyDTPay(double dt)

{

employeeDailyDTPay = dt;

}

void Employee::setEmployeeDailyMPPay(double mp)

{

employeeDailyMPPay = mp;

}

void Employee::setEmployeeDailyShowPay(double showpay)

{

employeeDailyShowPay = showpay;

}

void Employee::setEmployeeDailyOtherPay(double other)

{

employeeDailyOtherPay = other;

}

void Employee::setEmployeeDailyPayTotal(double total)

{

employeeDailyTotalPay = total;

}

double Employee::getEmployeeApplicableHourlyRate()

{

return employeeApplicableRateHourly;

}

double Employee::getEmployeeApplicableShowRate()

{

return employeeApplicableRateShow;

}

string Employee::getEmployeeDailyAssignment()

{

return employeeDailyAssignment;

}

double Employee::getEmployeeDailySTPay()

{

return employeeDailySTPay;

}

double Employee::getEmployeeDailyOTPay()

{

return employeeDailyOTPay;

}

double Employee::getEmployeeDailyDTPay()

{

return employeeDailyDTPay;

}

double Employee::getEmployeeDailyMPPay()

{

return employeeDailyMPPay;

}

double Employee::getEmployeeDailyShowPay()

{

return employeeDailyShowPay;

}

double Employee::getEmployeeDailyOtherPay()

{

return employeeDailyOtherPay;

}

double Employee::getEmployeeDailyPayTotal()

{

return employeeDailyTotalPay;

}

//destructor

Employee::~Employee()

{

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Structures.h

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <iostream>

#include <fstream>

#include <string>

#include <ctime> //raw time entry might come in a later release

#include <time.h>

using namespace std;

/\* declare and define structures:

-Time & Date

-EmployerContract

-EmployerRates

-

\*/

//structure declarations & default constructors

struct EmployerContract

{

string EmployerName;

string EmployerShortName;

bool FourHrMini; //Industry Standard is a four-hour minimum for all field assignments

int HrsBeforeMP; //Number of hours before Meal Penalty is incurred

int HrsBeforeOTdaily; //Number of daily hours before Overtime (time-and-a-half) is incurred

int HrsBeforeOTweekly; //Number of weekly hours before OT is incurred

string otherOTconditions; //Might there be any other condition that would incur Overtime(i.e., is Sunday OT on this contract?)

int DaysOfWeekOTApplicable1; //Days of the week that OT applies

int DaysOfWeekOTApplicable2; //Days of the week that OT applies

int HrsBeforeDTdaily; //Number of daily hours before Double Time is incurred

string otherDTconditions; //Might there be any other condition that would incur Double Time (i.e., insufficient turnaround)

int MinHoursOffToAvoidTurnaroundPay;

int NumberOfContractualHolidays;

int HolidayDateCodes2016[11]; //Jan 1 is day 1

double HolidayRatePremium; //1.5 base rate if OT on holiday, 2 if DT on holiday

//Construct default values

EmployerContract()

{

EmployerName = "default";

EmployerShortName = "Def";

FourHrMini = true;

HrsBeforeMP = 5;

HrsBeforeOTdaily = 8;

HrsBeforeOTweekly = 40;

otherOTconditions = "none";

DaysOfWeekOTApplicable1 = 0;

DaysOfWeekOTApplicable2 = 0;

HrsBeforeDTdaily = 16;

otherDTconditions = "none";

MinHoursOffToAvoidTurnaroundPay = 8;

NumberOfContractualHolidays = 0;

HolidayRatePremium = 1.5;

HolidayDateCodes2016[11] = 0;

}

};

struct EmployerJobPayRates

{

double MealPenaltyPercentageAsDec; //Some Meal Penalties are a percentage added to hourly pay

double StewardRateHourly; //Job classifications for hourly and "Show Call" pay

double StewardRateShow;

double RiggerRateHourly;

double RiggerRateShow;

double SpecialistRateHourly;

double SpecialistRateShow;

double HeadRateHourly;

double HeadRateShow;

double JourneymanRateHourly;

double JourneymanRateShow;

double ApprenticeRateHourly;

double ApprenticeRateShow;

double HandRateHourly;

double HandRateShow;

double ProgrammerRateHourly;

double ProgrammerRateShow;

//construct defaults

EmployerJobPayRates()

{

MealPenaltyPercentageAsDec = 2;

StewardRateHourly = 22;

StewardRateShow = 110;

RiggerRateHourly = 24;

RiggerRateShow = 0;

SpecialistRateHourly = 20;

SpecialistRateShow = 95;

HeadRateHourly = 19;

HeadRateShow = 85;

JourneymanRateHourly = 18;

JourneymanRateShow = 80;

ApprenticeRateHourly = 14;

ApprenticeRateShow = 60;

HandRateHourly = 12;

HandRateShow = 50;

ProgrammerRateHourly = 60;

ProgrammerRateShow = 0;

}

};

#pragma once

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Source.cpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

Sarah Wood

C++ Part 2

FINAL PROJECT

"Steward's Payroll Tools"

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This program presumes 2016!!

(in order to use for other years

\*update holiday calendar

\*implement leap-year checking)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Other presumptions/places to improve:

\*One single time to start work for the whole crew (implement multiple start times - figure out elegant UI for that)

\*One contract

\*One Field Job ("Gig" "Call" "Show" In real life, this could be multiple days, but under the umbrella of one event.)

\*One day (Multiple day tracking provisioned, could be implemented on a per-employee or per-client basis)

\*Our workday is the first day of the Job.

\*/

#include <iostream>

#include <iomanip>

#include <fstream>

#include <string>

#include <ctime> //raw time entry might come in a later release

#include <time.h>

#include "Employee.h"

#include "Structures.h"

using namespace std;

//structure declarations & definitions included in "structures.cpp"

//function prototypes

bool getContractData(EmployerContract);

bool getRatesData(EmployerJobPayRates);

void makeCrewList(Employee\*);

void makeTimeSheet(Employee\*);

void MANUALinputDailyWork(EmployerContract, EmployerJobPayRates\*, Employee\*); //IRL contract 9.0x, pg 6-7

void FILEinputDailyWork(EmployerContract, EmployerJobPayRates\*, Employee\*); //IRL contract 9.0x, pg 6-7

void calculateDailyTotalPayPerEmployee(Employee\*, EmployerJobPayRates\*);

void calculateTotalGrossForCrew(Employee\*);

void outputFiguresPerEmployee(Employee\*);

void contingentTotalGrossForCrew(Employee\*); //"What if" function for management planning purposes

void testInput(Employee\*, EmployerJobPayRates\*);

//global variables

int day, month, hour, minute; //time-based hours are on the wishlist

const int SIZE = 10; //size of our labor Pool

double invoiceTotal;

EmployerContract GenericEmployerContract; //conditions impacting pay

EmployerJobPayRates EmployerRatesArray[2] ; //rates

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//I ran into the "most vexing parse" problem in C++. I found it exceedingly difficult to overwrite my default constructor.

//I instead built an array, so that default could be [0], and my working data could be[1].

//I think I didn't run into this in the part 1 final project, because I was using my defaults as test data.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//instantiate a pool of crew to draw from

//note: (emp#, name, baseQual, baseQualShort = J, a1, a2, a3, H, Assignment)

//make an array of test Employees

Employee laborPool[SIZE] = {

Employee(100, "Tex Avery", "Journeyman", "J", "Steward"),

Employee(110, "Bugs Bunny", "Journeyman", "J", "Rigger"),

Employee(160, "Porky Pig", "Journeyman", "J", "Head"),

Employee(8010, "Daffy Duck", "Apprentice 3", "A3", "Grip"),

Employee(8120, "Elmer Fudd", "Apprentice 2", "A2", "Grip"),

Employee(10120, "Tweety Bird", "Hand", "H", "Grip"),

Employee(10130, "Sylvester Cat", "Hand", "H", "Grip"),

Employee(10210, "Yosemite Sam", "Hand", "H", "Grip"),

Employee(10310, "Gossamer", "Hand", "H", "Grip"),

Employee(10440, "Marvin Martian", "Hand", "H", "Grip"),

};

void main()

{

//initialize console

cout << "Steward's Payroll Tools" << '\n' << '\n';

cout << "Your CREW LIST wil be generated." << '\n';

cout << "For this release, your WORKSHEET has been filled out with test data." << '\n';

//get CONTRACT data (rates & conditions),

getContractData(GenericEmployerContract);

getRatesData(EmployerRatesArray[1]);

//get CREW LIST

//(I thought I may not need a crew list. However, it is a great worksheet onsite. It will be used to automate work input.)

makeCrewList(laborPool);

//manual input of daily work

//MANUALinputDailyWork(GenericEmployerContract, EmployerRatesArray[1], laborPool);

//file import of daily work -- CREW LIST filled out in spreadsheet editor with hours/shows worked

FILEinputDailyWork(GenericEmployerContract, EmployerRatesArray, laborPool);

//make TIME SHEET,

makeTimeSheet(laborPool);

//calculate total per day per employee

calculateDailyTotalPayPerEmployee(laborPool, EmployerRatesArray);

//calculate total gross wages invoiced

calculateTotalGrossForCrew(laborPool);

//output table of figures per employee

outputFiguresPerEmployee(laborPool);

//output total gross, based on contingencies

contingentTotalGrossForCrew(laborPool);

//testInput(laborPool, EmployerRatesArray);

//Keep window open

cout << "Press ENTER to close program." << endl;

cin.ignore(256, '\n');

cin.get();

}

bool getContractData(EmployerContract contract)

{

//If we were working with multiple contracts, we could import the relevant one and send it into the appropriate structure.

//As currently scoped, this is a single-contract program, but I am building it to be scalable in the future.

fstream contractData;

contractData.open("testContract.txt");

if (contractData)

{

cout << "Contract file opened successfully." << '\n';

contractData >> contract.EmployerName;

contractData >> contract.EmployerShortName;

contractData >> contract.FourHrMini;

contractData >> contract.HrsBeforeMP;

contractData >> contract.HrsBeforeOTdaily;

contractData >> contract.HrsBeforeOTweekly;

contractData >> contract.otherOTconditions;

contractData >> contract.HrsBeforeDTdaily;

contractData >> contract.otherDTconditions;

contractData >> contract.MinHoursOffToAvoidTurnaroundPay;

contractData >> contract.NumberOfContractualHolidays;

contractData >> contract.HolidayRatePremium;

for (int i = 1; i < 11; i++)

{

contractData >> contract.HolidayDateCodes2016[i];

}

return true;

}

else

{

cout << "Contract File Error." << '\n';

return false;

}

}

bool getRatesData(EmployerJobPayRates rates)

{

ifstream ratesData;

ratesData.open("testRates.txt");

while (ratesData)

{

ratesData >> rates.MealPenaltyPercentageAsDec;

ratesData >> rates.StewardRateHourly;

ratesData >> rates.StewardRateShow;

ratesData >> rates.RiggerRateHourly;

ratesData >> rates.RiggerRateShow;

ratesData >> rates.SpecialistRateHourly;

ratesData >> rates.SpecialistRateShow;

ratesData >> rates.HeadRateHourly;

ratesData >> rates.HeadRateShow;

ratesData >> rates.JourneymanRateHourly;

ratesData >> rates.JourneymanRateShow;

ratesData >> rates.ApprenticeRateHourly;

ratesData >> rates.ApprenticeRateShow;

ratesData >> rates.HandRateHourly;

ratesData >> rates.HandRateShow;

ratesData >> rates.ProgrammerRateHourly;

ratesData >> rates.ProgrammerRateShow;

cout << "Rates file opened successfully." << '\n';

return true;

}

return false;

}

void MANUALinputDailyWork(EmployerContract contract, EmployerJobPayRates rates, Employee[])

{

//logic gates for daily work

//(is straight hourly, OT, DT, SHOW pay, penalties? other?)

//(dictated by CONTRACT data)

for (int i = 0; i < SIZE; i++)

{

//declare local variables

double number, number2, number3;

double tempHours = 0, tempMPHours = 0;

char yesNo2;

bool valid = false;

/\*

//select a classification for [SELECTED EMPLOYEE]

cout << '\n' << "Please select a job classification for " << laborPool[i].employeeName << " :" << '\n';

cout << "1 Steward" << '\t' << "2 Rigger" << '\t' << "3 Specialist" << '\n';

cout << "4 Head" << '\t' << '\t' << "5 Programmer" << '\t' << "6 Grip" << '\n';

cout << "Please make a selection: " << '\t';

cin >> choice;

//validate user input

while (!cin)

{

cout << "Invalid Input. Try again. Enter #1-6: " << '\t';

cin.clear();

cin.ignore();

cin >> choice;

}

while (choice<1 || choice>6)

{

valid = !true; //declare this attampt !valid

cin.clear(); //clear the input stream

cin.ignore(); //ignore invalid stuff.

cout << "Invalid input. Try again. Enter #1-6: " << '\t'; //user prompt

cin >> choice; //try again...

}

//Assign Rate to each JobClassification choice

if (choice == 1)

{

laborPool[i].setEmployeeApplicableRateHourly(rates.StewardRateHourly);

laborPool[i].setEmployeeApplicableRateShow(rates.StewardRateShow);

}

else if (choice == 2)

{

laborPool[i].setEmployeeApplicableRateHourly(rates.RiggerRateHourly);

laborPool[i].setEmployeeApplicableRateShow(rates.RiggerRateShow);

}

else if (choice == 3)

{

laborPool[i].setEmployeeApplicableRateHourly(rates.SpecialistRateHourly);

laborPool[i].setEmployeeApplicableRateShow(rates.SpecialistRateShow);

}

else if (choice == 4)

{

laborPool[i].setEmployeeApplicableRateHourly(rates.HeadRateHourly);

laborPool[i].setEmployeeApplicableRateShow(rates.HeadRateShow);

}

else if (choice == 5)

{

//this is where we use the base qualifications.

if (laborPool[i].employeeBaseQualificationSHORT == "J")

{

laborPool[i].setEmployeeApplicableRateHourly(rates.JourneymanRateHourly);

laborPool[i].setEmployeeApplicableRateShow(rates.JourneymanRateShow);

}

else if (laborPool[i].employeeBaseQualificationSHORT == "A3" || laborPool[i].employeeBaseQualificationSHORT == "A2" || laborPool[i].employeeBaseQualificationSHORT == "A1")

{

laborPool[i].setEmployeeApplicableRateHourly(rates.ApprenticeRateHourly);

laborPool[i].setEmployeeApplicableRateShow(rates.ApprenticeRateShow);

}

else if (laborPool[i].employeeBaseQualificationSHORT == "H")

{

laborPool[i].setEmployeeApplicableRateHourly(rates.HandRateHourly);

laborPool[i].setEmployeeApplicableRateShow(rates.HandRateShow);

}

else

{

cout << "ERROR. Employee does not have a base qualification. Will be paid hand rate." << '\n';

laborPool[i].setEmployeeApplicableRateHourly(rates.HandRateHourly);

laborPool[i].setEmployeeApplicableRateShow(rates.HandRateShow);

}

}//close "choice == INT"

\*/

cout << "Please enter the number of shows " << laborPool[i].employeeName << " worked today (0 if none): " << '\t';

cin >> number;

//validate input. Max 3 shows/day before idiot-checking

while (!cin)

{

valid = !true;

cout << "Invalid Input. Try again." << '\t';

cin.clear();

cin.ignore();

cin >> number;

}

if (number>3)

{

cout << "Did they really work " << number << " shows today? Y/N?";

cin >> yesNo2;

while (yesNo2 != 'Y' && yesNo2 != 'y' && yesNo2 != 'N' && yesNo2 != 'n')

{

cout << "Invalid Input. Y or N? " << '\t';

cin.clear();

cin.ignore();

cin >> yesNo2;

}

if (yesNo2 == 'N' || yesNo2 == 'n')

{

cin.clear();

cin.ignore();

cout << "How many shows did they REALLY work?" << '\t';

cin >> number;

while (!cin)

{

cout << "Invalid Input. Try again." << '\t';

cin.clear();

cin.ignore();

cin >> number;

}

}

} //close idiot-check for number of shows worked

laborPool[i].setEmployeeDailyShows(number);

//Hourly Work? get hours

cout << '\n' << "Please enter the number of hours between CALL TIME and FIRST meal break: " << '\n';

cin >> number;

while (!cin)

{

cout << "Invalid Input. Try again. " << '\t';

cin.clear();

cin.ignore();

cin >> number;

}

if (number >6)

{

cout << '\n' << "Did they really work " << number << " hours without a meal break? Y/N?";

cin >> yesNo2;

while (yesNo2 != 'Y' && yesNo2 != 'y' && yesNo2 != 'N' && yesNo2 != 'n')

{

cout << "Invalid Input. Y or N? " << '\t';

cin.clear();

cin.ignore();

cin >> yesNo2;

}

if (yesNo2 == 'N' || yesNo2 == 'n')

{

cin.clear();

cin.ignore();

cout << "How many hours did they REALLY work?" << '\t';

cin >> number;

}

}

//if <4 make equal to 4 (industry standard is 4 hour minimum)

if (number <= 4)

{

number = 4;

tempHours = number;

}

else if (number <= 5)

{

tempHours = number;

}

//if >5, process Meal Penalty,

else if (number >5)

{

tempHours = number;

tempMPHours = (number - 5);

}

cout << "Please enter the number of hours between CALLBACK and SECOND meal break: " << '\n';

cout << "Enter 0 if none." << '\t';

cin >> number2;

while (!cin)

{

cout << "Invalid Input. Try again. " << '\t';

cin.clear();

cin.ignore();

cin >> number2;

}

if (number2 >6)

{

cout << "Did they really work " << number2 << " hours without a meal break? Y/N?";

cin >> yesNo2;

while (yesNo2 != 'Y' && yesNo2 != 'y' && yesNo2 != 'N' && yesNo2 != 'n')

{

cout << "Invalid Input. Y or N? " << '\t';

cin.clear();

cin.ignore();

cin >> yesNo2;

}

if (yesNo2 == 'N' || yesNo2 == 'n')

{

cin.clear();

cin.ignore();

cout << "How many hours did they REALLY work?" << '\t';

cin >> number2;

}

}

//if >5, process Meal Penalty, if <2 make equal to 2 (industry standard is 2 hours pay after "lunch")

if (number2 == 0)

{

tempHours += 0;

}

else if (number2>0 && number2 < 2)

{

number2 = 2;

tempHours += number2;

}

else if (number2 <= 5)

{

tempHours += number2;

}

//if >5, process Meal Penalty,

else if (number2 >5)

{

tempHours += number2;

tempMPHours += (number2 - 5);

}

cout << "Please enter the number of hours between CALLBACK and CUT: " << '\n';

cout << "Enter 0 if none." << '\t';

cin >> number3;

while (!cin)

{

cout << "Invalid Input. Try again. " << '\t';

cin.clear();

cin.ignore();

cin >> number3;

}

if (number3 >6)

{

cout << "Did they really work " << number3 << " hours without a meal break? Y/N?";

cin >> yesNo2;

while (yesNo2 != 'Y' && yesNo2 != 'y' && yesNo2 != 'N' && yesNo2 != 'n')

{

cout << "Invalid Input. Y or N? " << '\t';

cin.clear();

cin.ignore();

cin >> yesNo2;

}

if (yesNo2 == 'N' || yesNo2 == 'n')

{

cin.clear();

cin.ignore();

cout << "How many hours did they REALLY work?" << '\t';

cin >> number3;

}

}

//if >5, process Meal Penalty, if <2 make equal to 2 (industry standard is 2 hours pay after "lunch")

if (number3 == 0)

{

tempHours += 0;

}

else if (number3>0 && number3 < 2)

{

number3 = 2;

tempHours += number3;

}

else if (number3 <=5)

{

tempHours += number3;

}

else if (number3 >5)

{

tempHours += number3;

tempMPHours += (number3 - 5);

}

laborPool[i].setEmployeeDailyTotalHours(tempHours);

//if total >8, process OT, if >12, process DT

if (tempHours > 12)

{

laborPool[i].setEmployeeDailySTHours(8);

laborPool[i].setEmployeeDailyOTHours(4);

laborPool[i].setEmployeeDailyDTHours(tempHours - 12);

laborPool[i].setEmployeeDailyMPHours(tempMPHours);

}

else if (tempHours>8)

{

laborPool[i].setEmployeeDailySTHours(8);

laborPool[i].setEmployeeDailyOTHours(tempHours - 8);

laborPool[i].setEmployeeDailyMPHours(tempMPHours);

}

else

{

laborPool[i].setEmployeeDailySTHours(tempHours);

laborPool[i].setEmployeeDailyMPHours(tempMPHours);

}

}

}

void FILEinputDailyWork(EmployerContract contract, EmployerJobPayRates \*rates, Employee[])

{

double workTemp1, showsTemp, hours1Temp, hours2Temp, hours3Temp, tempMPHours=0;

string tempAssignment;

ifstream crewWorksheet;

crewWorksheet.open("crewWorksheet.txt", ios::in);

crewWorksheet.ignore(256, '\n'); //ignore the first line. It's only headers.

for (int i = 0; i < SIZE ; i++)

{

//read from the worksheet

crewWorksheet.ignore(20, '\t'); //ignore the first field (emp name)

crewWorksheet.ignore(20, '\t'); //ignore the second field (emp#)

crewWorksheet >> tempAssignment>>showsTemp >> hours1Temp >> hours2Temp >> hours3Temp;

//assign temp values read from worksheet to Employee object values

laborPool[i].setEmployeeDailyShows(showsTemp);

//workTemp1 = (hours1Temp + hours2Temp + hours3Temp);

//laborPool[i].setEmployeeDailyTotalHours(workTemp1);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Compute ST, OT, DT, MP

if (hours1Temp <= 4)

{

//set to 4 hour minimum

hours1Temp = 4;

workTemp1 = hours1Temp;

}

else if (hours1Temp <= 5)

{

workTemp1 = hours1Temp;

}

//if >5, process Meal Penalty,

else if (hours1Temp >5)

{

workTemp1 = 5;

tempMPHours = (hours1Temp - 5);

}

if (hours2Temp == 0)

{

workTemp1 += 0;

}

else if (hours2Temp>0 && hours2Temp < 2)

{

hours2Temp = 2;

workTemp1 += hours2Temp;

}

else if (hours2Temp <= 5)

{

workTemp1 += 5;

}

//if >5, process Meal Penalty,

else if (hours2Temp >5)

{

workTemp1 += hours2Temp;

tempMPHours += (hours2Temp - 5);

}

//if >5, process Meal Penalty, if <2 make equal to 2 (industry standard is 2 hours pay after "lunch")

if (hours3Temp == 0)

{

workTemp1 += 0;

}

else if (hours3Temp>0 && hours3Temp < 2)

{

hours3Temp = 2;

workTemp1 += hours3Temp;

}

else if (hours3Temp <= 5)

{

workTemp1 += hours3Temp;

}

else if (hours3Temp >5)

{

workTemp1 += hours3Temp;

tempMPHours += (hours3Temp - 5);

}

laborPool[i].setEmployeeDailyTotalHours(workTemp1);

//if total >8, process OT, if >12, process DT

if (workTemp1 > 12)

{

laborPool[i].setEmployeeDailySTHours(8);

laborPool[i].setEmployeeDailyOTHours(4);

laborPool[i].setEmployeeDailyDTHours(workTemp1 - 12);

laborPool[i].setEmployeeDailyMPHours(tempMPHours);

}

else if (workTemp1>8)

{

laborPool[i].setEmployeeDailySTHours(8);

laborPool[i].setEmployeeDailyOTHours(workTemp1 - 8);

laborPool[i].setEmployeeDailyMPHours(tempMPHours);

}

else

{

laborPool[i].setEmployeeDailySTHours(workTemp1);

laborPool[i].setEmployeeDailyMPHours(tempMPHours);

}

}

}

void testInput(Employee \*emp, EmployerJobPayRates \*rates)

{

cout<< '\n' << rates[1].MealPenaltyPercentageAsDec

<< '\n' << rates[1].StewardRateHourly

<< '\n' << rates[1].StewardRateShow

<< '\n' << rates[1].RiggerRateHourly

<< '\n' << rates[1].RiggerRateShow

<< '\n' << rates[1].SpecialistRateHourly

<< '\n' << rates[1].SpecialistRateShow

<< '\n' << rates[1].HeadRateHourly

<< '\n' << rates[1].HeadRateShow

<< '\n' << rates[1].JourneymanRateHourly

<< '\n' << rates[1].JourneymanRateShow

<< '\n' << rates[1].ApprenticeRateHourly

<< '\n' << rates[1].ApprenticeRateShow

<< '\n' << rates[1].HandRateHourly

<< '\n' << rates[1].HandRateShow

<< '\n' << rates[1].ProgrammerRateHourly

<< '\n' << rates[1].ProgrammerRateShow << '\n';

for (int i = 0; i < SIZE; i++)

cout << laborPool[i].getEmployeeName() << '\t' <<

laborPool[i].getEmployeeNumber() << '\t' <<

laborPool[i].getEmployeeDailyTotalHours() << '\t' <<

laborPool[i].getEmployeeDailyShows() << '\t'<<

laborPool[i].getEmployeeDailyAssignment()<< '\t'<<

laborPool[i].getEmployeeApplicableHourlyRate()<<'\t'<<

laborPool[i].getEmployeeApplicableShowRate()<<'\t'<<

laborPool[i].getEmployeeDailyPayTotal()<<'\n';

}

void makeCrewList(Employee \*emp)

{

fstream crewList;

crewList.open("crewList.txt", ios::out);

cout << "Generating Crew List." << '\n';

crewList << "Name" << '\t' << "#" << '\t'<<"Assignment"<<'\t' <<"Shows"<<'\t'<< "Hours1" << '\t' << "Hours2" << '\t' << "Hours3"<< '\n';

for (int i = 0; i < SIZE; i++)

{

crewList << laborPool[i].getEmployeeName() << '\t' << laborPool[i].getEmployeeNumber() << '\t'<<laborPool[i].getEmployeeDailyAssignment()<<'\t'<<'\t'<<'\t'<<'\n';

}

}

void makeTimeSheet(Employee \*emp)

{

ofstream timeSheet;

timeSheet.open("timeSheet.txt", ios::out);

cout << '\n' << "TIME SUMMARY" << '\n';

cout << "Name" << '\t' << '\t' << "#" << '\t' << "ST" << '\t' << "OT" << '\t' << "DT" << '\t' << "MP" << '\t' << "Shows" <<'\n';

timeSheet<< "Name" << '\t' << "#" << '\t' << "ST" << '\t' << "OT" << '\t' << "DT" << '\t' << "MP" << '\t' << "Shows" << '\n';

for (int i = 0; i < SIZE; i++)

{

cout << laborPool[i].getEmployeeName() << '\t' <<

laborPool[i].getEmployeeNumber() << '\t' <<

laborPool[i].getEmployeeDailySTHours() << '\t' <<

laborPool[i].getEmployeeDailyOTHours() << '\t' <<

laborPool[i].getEmployeeDailyDTHours() << '\t' <<

laborPool[i].getEmployeeDailyMPHours() << '\t' <<

laborPool[i].getEmployeeDailyShows() << '\n';

timeSheet << laborPool[i].getEmployeeName() << '\t' <<

laborPool[i].getEmployeeNumber() << '\t' <<

laborPool[i].getEmployeeDailySTHours() << '\t' <<

laborPool[i].getEmployeeDailyOTHours() << '\t' <<

laborPool[i].getEmployeeDailyDTHours() << '\t' <<

laborPool[i].getEmployeeDailyMPHours() << '\t' <<

laborPool[i].getEmployeeDailyShows() << '\n';

}

}

void calculateDailyTotalPayPerEmployee(Employee \*emp, EmployerJobPayRates \*rates)

{

double tempSTPay, tempOTPay, tempDTPay, tempMPPay, tempShowPay, tempTotalPay;

double hourlyRate, showRate;

string tempAssignment;

for (int i = 0; i < SIZE; i++)

{

//Assign Rate to each JobClassification choice

if (laborPool[i].employeeDailyAssignment == "Steward")

{

hourlyRate = rates[1].StewardRateHourly;

showRate = rates[1].StewardRateShow;

}

else if (laborPool[i].employeeDailyAssignment == "Rigger")

{

hourlyRate = rates[1].RiggerRateHourly;

showRate = rates[1].RiggerRateShow;

}

else if (laborPool[i].employeeDailyAssignment == "Specialist")

{

hourlyRate = rates[1].SpecialistRateHourly;

showRate = rates[1].SpecialistRateShow;

}

else if (laborPool[i].employeeDailyAssignment == "Head")

{

hourlyRate = rates[1].HeadRateHourly;

showRate = rates[1].HeadRateShow;

}

else if (laborPool[i].employeeDailyAssignment == "Grip")

{

//this is where we use the base qualifications.

if (laborPool[i].employeeBaseQualificationSHORT == "J")

{

hourlyRate = rates[1].JourneymanRateHourly;

showRate = rates[1].JourneymanRateShow;

}

else if (laborPool[i].employeeBaseQualificationSHORT == "A3" || laborPool[i].employeeBaseQualificationSHORT == "A2" || laborPool[i].employeeBaseQualificationSHORT == "A1")

{

hourlyRate = rates[1].ApprenticeRateHourly;

showRate = rates[1].ApprenticeRateShow;

}

else if (laborPool[i].employeeBaseQualificationSHORT == "H")

{

hourlyRate = rates[1].HandRateHourly;

showRate = rates[1].HandRateShow;

}

else

{

cout << "ERROR. Employee does not have a base qualification. Will be paid hand rate." << '\n';

hourlyRate = rates[1].HandRateHourly;

showRate = rates[1].HandRateShow;

}

}

//do math for pay

tempSTPay = (hourlyRate \* laborPool[i].employeeDailySTHours);

tempOTPay = (1.5 \* (hourlyRate \* laborPool[i].employeeDailyOTHours));

tempDTPay = (2 \* (hourlyRate \* laborPool[i].employeeDailyDTHours));

tempMPPay = (2 \* (hourlyRate \* laborPool[i].employeeDailyMPHours));

tempShowPay = (showRate \* laborPool[i].employeeDailyShows);

tempTotalPay = (tempSTPay + tempOTPay + tempDTPay + tempMPPay + tempShowPay);

//store values in member variables

laborPool[i].setEmployeeApplicableRateHourly(hourlyRate);

laborPool[i].setEmployeeApplicableRateShow(showRate);

laborPool[i].setEmployeeDailySTPay(tempSTPay);

laborPool[i].setEmployeeDailyOTPay(tempOTPay);

laborPool[i].setEmployeeDailyDTPay(tempDTPay);

laborPool[i].setEmployeeDailyMPPay(tempMPPay);

laborPool[i].setEmployeeDailyShowPay(tempShowPay);

laborPool[i].setEmployeeDailyPayTotal(tempTotalPay);

}

}

void calculateTotalGrossForCrew(Employee \*emp)

{

double tempAccumulator=0, tempEmployeeDaily;

for (int i = 0; i < SIZE; i++)

{

tempEmployeeDaily = laborPool[i].employeeDailyTotalPay;

tempAccumulator += tempEmployeeDaily;

}

cout << fixed << showpoint<<setprecision(2) << '\n';

cout << "Daily Invoice Total is: $" << tempAccumulator << '\n';

invoiceTotal = tempAccumulator;

}

void contingentTotalGrossForCrew(Employee \*emp)

{

double currentHourlyCost, tempInvoiceTotal, tempAccumulator=0;

char yesNo;

tempInvoiceTotal = invoiceTotal;

cout << "Would you like to estimate costs if you keep the crew longer?" << '\n';

cin >> yesNo;

while (yesNo != 'Y' && yesNo != 'y' && yesNo != 'N' && yesNo != 'n')

{

cout << "Invalid Input. Try again." << '\t';

cin.clear();

cin.ignore();

cin >> yesNo;

}

//note: the crew chief is the bellwether in this case, because they are the first one on the clock, and the last one off the clock

//They have been "on" the longest, and will put in the most hours.

if (yesNo == 'Y' || yesNo == 'y')

{

if (laborPool[0].employeeDailyOTHours > 0) //if the crew chief is in OT, base estimate on OT

{

//add up crew rate, \*1.5, make that current hourly cost

for (int i = 0; i < SIZE; i++)

{

tempAccumulator += laborPool[i].employeeApplicableRateHourly;

}

currentHourlyCost = (1.5 \* tempAccumulator);

}

else if (laborPool[0].employeeDailyDTHours > 0)//if crew chief is in DT, base estimate on DT

{

//add up crew rate, \*2.0, make that current hourly cost

for (int i = 0; i < SIZE; i++)

{

tempAccumulator += laborPool[i].employeeApplicableRateHourly;

}

currentHourlyCost = (2 \* tempAccumulator);

}

else //if crew chief in Straight Time, base estimate on Straight Time

{

for (int i = 0; i < SIZE; i++)

{

tempAccumulator += laborPool[i].employeeApplicableRateHourly;

}

currentHourlyCost = tempAccumulator;

}

cout << fixed << showpoint << setprecision(2) << '\n';

cout << "Your crew costs about $" << currentHourlyCost << " for each additional hour." << '\n' <<

"(Note, OT @ 8hours, DT @ 12hours.)" << '\n';

cout << "FYI, your steward has been here for " << laborPool[0].employeeDailyTotalHours << " hours so far." <<'\n'<< '\n';

}

}

void outputFiguresPerEmployee(Employee \*emp)

{

fstream payrollInvoice;

payrollInvoice.open("testInvoice.txt", ios::out);

//This is formatted to accomodate real payroll invoices used in the field.

//It will be tested with a real invoice that has been "scrubbed" of any confidential data.

for (int i = 0; i < SIZE; i++)

{

payrollInvoice << laborPool[i].getEmployeeName() << '\t' <<

laborPool[i].getEmployeeDailyAssignment() << '\t' <<

laborPool[i].getEmployeeApplicableHourlyRate() << '\t' <<

laborPool[i].getEmployeeDailySTHours() << '\t' <<

laborPool[i].getEmployeeDailySTPay() << '\t' <<

laborPool[i].getEmployeeDailyOTHours() << '\t' <<

laborPool[i].getEmployeeDailyOTPay() << '\t' <<

laborPool[i].getEmployeeDailyDTHours() << '\t' <<

laborPool[i].getEmployeeDailyDTPay() << '\t' << '\t' <<

laborPool[i].getEmployeeDailyShows() << '\t' <<

laborPool[i].getEmployeeApplicableShowRate() << '\t' <<

laborPool[i].getEmployeeDailyShowPay() << '\t' <<

laborPool[i].getEmployeeDailyMPPay() << '\t' <<

laborPool[i].getEmployeeDailyPayTotal() << '\n';

}

//output same values to screen

//It was just too much to fit on console screen. Abridged accordingly.

cout << "Name" << '\t'<<'\t' << "Assg" << '\t' << "Rate" << " " << "Hours" << " " << "#Shows" << " " << "Show $" << '\t' << "Day $" << '\n';

cout << resetiosflags(ios::fixed | ios::showpoint);

for (int i = 0; i < SIZE; i++)

{

cout<< laborPool[i].getEmployeeName() << '\t' <<

laborPool[i].getEmployeeDailyAssignment() << '\t'<<"$" <<

laborPool[i].getEmployeeApplicableHourlyRate() << " " <<

laborPool[i].getEmployeeDailyTotalHours()<<" "<<

laborPool[i].getEmployeeDailyShows() << " $" << fixed << showpoint << setprecision(2)<<

laborPool[i].getEmployeeDailyShowPay() << "\t$" <<

laborPool[i].getEmployeeDailyPayTotal() << resetiosflags(ios::fixed | ios::showpoint) << '\n';

}

}