Mariah Diaz

CIS 200

Professor Mann

March 12, 2019

Project 2

Initial Test Plan:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test # | Valid / Invalid Data | Description of test | Input Value | Expected Output | Actual Output | Test Pass / Fail |
| 1 | Valid Data | Ensure that the program is accepting user input | Input of first name “Ignis,” a last name of “Scientia,” a ssn of “987-65-4321” | The program should store this data and display it to the user as requried |  |  |

Source Code:

#include<iostream>

#include<fstream>

#include<sstream>

#include<string>

#include<cmath>

#include<cstdlib>

#include<vector>

#include<iomanip>

#include<cassert>

#include<ctime>

#include<cctype>

#include<cstring>

using namespace std;

//base class Employee

class Employee

{

private:

//first & last name, ssn, employeeNum

string firstName;

string lastName;

string SSN;

string employeeNum;

public:

Employee()

{

firstName = "Noctis";

lastName = "LucisCaelum";

SSN = "123-456-7891";

employeeNum = "1234";

}

//setter functions

void setName(string, string);

bool setSocial(string);

bool setEmployeeNum(string);

string getName();

string getSocial();

string getEmployeeNum();

void DisplayEmployee();

};

class SalaryEmployeePay :public Employee //needs to have employee name, ssn, and employee no.

{

private:

float annualPay;

float weeklyPay; //calculated from annualPay

int taxRateCode;

public:

SalaryEmployeePay()

{

annualPay = 0;

}

SalaryEmployeePay(float w)

{

annualPay = w;

}

//member functions to manipulate the class data members - get, set, print

bool setAnnualPay(float);

float getAnnualPay();

float getWeeklyPay();

void calculateWeeklyPay();

float gettaxRateCode();

float settaxRateCode();

void DisplayEmployee();

};

bool SalaryEmployeePay::setAnnualPay(float w)

{

if (w < 0)

{

cout << "Error : Invalid entry" << endl;

return 0;

}

else

{

annualPay = w;

calculateWeeklyPay();

return 1;

}

}

//annual pay -- will help determine the weekly pay

float SalaryEmployeePay::getAnnualPay()

{

return annualPay;

}

//get info about annualPay --> weekly pay

float SalaryEmployeePay::getWeeklyPay()

{

weeklyPay = annualPay / (365 / 7); //annual pay divided by number of weeks in a year (365/7)

return weeklyPay;

}

/\*void SalaryEmployeePay::calculateWeeklyPay()

{

weeklyPay = annualPay / (365 / 7);

return weeklyPay;

}

\*/

//print the employee, their annual & weekly pay

void SalaryEmployeePay::DisplayEmployee()

{

//Employee::DisplayEmployee();

cout << "Annual pay: " << annualPay << endl;

cout << "Weekly pay: " << weeklyPay << endl;

}

class HourlyPay : public SalaryEmployeePay

{

private:

float rate; //hourly rate

const float overTime = 1.5; //overTime = 1.5(hoursWorked - 40)

float hours;

public:

HourlyPay()

{

rate = 0;

hours = 40; //40 is the max amount of hours before overTime is kicked in

}

HourlyPay(float r, float h)

{

rate = r;

hours = h;

}

//get & set > print

void setHourlyRate();

float getHourlyRate();

void setHoursWorked(float);

float getHoursWorked();

float getOverTIme();

void DisplayEmployee();

void setSocial();

float getSocial();

void SetName(string, string);

float setEmployeeNum();

};

void HourlyPay::setHourlyRate()

{

float weeklyPay;

weeklyPay = getWeeklyPay();

rate = weeklyPay / 40; //40 = hours worked, will ask user to input how

//many hours a particular employee worked

}

float HourlyPay::getHourlyRate()

{

return rate;

}

void HourlyPay::setHoursWorked(float h)

{

hours = h;

}

float HourlyPay::getHoursWorked()

{

return hours;

}

float HourlyPay::getOverTIme()

{

float OT; //overtime

float OTPay; //overtime pay -- remember that it equals 1.5(hours-40)

OT = hours - 40.0;

OTPay = (rate \* 40.0) + (OT\*(rate\*1.5));

return OTPay;

}

void HourlyPay::DisplayEmployee()

{

SalaryEmployeePay::DisplayEmployee();

cout << "Pay rate: " << rate << endl;

cout << "Hours worked: " << hours << endl;

if (hours > 40)

{

cout << "Overtime pay this week: " << getOverTIme() << endl;

}

}

//function definitions & setter functions

void Employee::setName(string first, string last)

{

firstName = first;

lastName = last;

}

//SSN - make sure it's in the XXX-XX-XXXX format; where X is a digit from 1-9

bool Employee::setSocial(string social)

{

bool SSNlength = false;

bool SSNdigit = false;

if (social.length() == 11)

{

if (social[3] == '-' && social[6] == '-') // 012 3 45 6 7890, 3 & 6 are the dashes

{

if (isdigit(social[0]) && isdigit(social[1]) && isdigit(social[2]) && isdigit(social[4]) && isdigit(social[5])

&& isdigit(social[7]) && isdigit(social[8]) && isdigit(social[9]) && isdigit(social[10]))

{

SSNdigit = true;

SSNlength = true;

}

}

}

if (SSNlength == true && SSNdigit == true)

{

SSN = social;

return true;

}

else

{

cout << "Social security number is invalid." << endl << endl;

}

}

//employeeNum & validation(s)

//in XXX-L, X is any number 0-9 while L is a letter A through M

bool Employee::setEmployeeNum(string empNum)

{

bool empLength = false;

bool empLetter = false;

if (empNum.length() == 5)

{

if (empNum[3] == '-') //012 3 L

{

if (isdigit(empNum[0]) && isdigit(empNum[1]) && isdigit(empNum[2]))

{

if (empNum[4] >= 'A' && empNum[4] <= 'L')

{

empLength = true;

empLetter = true;

}

}

}

}

if (empLength = true && empLetter == true)

{

employeeNum = empNum;

return true;

}

else

{

cout << "Invalid, incorrect format was entered." << endl;

}

}

//getter functions - first and last name of the employee

string Employee::getName()

{

return firstName;

cout << " ";

return lastName;

}

//employee's SSN

string Employee::getSocial()

{

return SSN;

}

//employee's employee number

string Employee::getEmployeeNum()

{

return employeeNum;

}

void Employee::DisplayEmployee()

{

cout << "EMPLOYEE INFORMATION" << endl << endl;

cout << "Name: " << firstName << " " << lastName << endl;

cout << "Employee Number: " << employeeNum << endl;

cout << "Social Security Number: " << SSN << endl;

}

void PrintCheck(HourlyPay);

int main()

{

HourlyPay employee1;

string first\_name;

string last\_name;

string employee\_ssn;

string employee\_num;

string userInput;

float annual\_pay;

float hours;

float rate;

HourlyPay \*emp1 = nullptr;

emp1 = &employee1;

//have user input info

cout << "Enter the information below..." << endl << endl;

cout << "First name: ";

cin >> first\_name;

cout << endl << "Last name: ";

cin >> last\_name;

employee1.SetName(first\_name, last\_name);

//SSN

//do

//{

cout << "Social security number must be in the following format:" << endl;

cout << "XXX-XX-XXXX; where each \"X\" is any digit between 0 and 9." << endl;

cout << "Please enter " << first\_name << " " << last\_name << "'s social security number: ";

cin >> employee\_ssn;

//} while (employee1.setSocial(employee\_ssn) != true);

//employee number

//do

//{

cout << "Employee number must be entered in the following format:" << endl;

cout << "XXX-L where X is any digit from 0 to 9 while L is any letter from \"A\" to \"M\"." << endl;

cout << "Please enter " << first\_name << " " << last\_name << "'s employee number:";

cin >> employee\_num;

//} while (employee1.setEmployeeNum(employee\_num) != true);

//are they a hourly, salary, or agency employee

cout << "Is " << first\_name << " " << last\_name << " hourly, salary, or agency?";

getline(cin, userInput);

//if-else statements for hourly > salary > agency

//also, do not forget tax rate code!!!

/\*if (userInput == "hourly")

{

cout << "What is their hourly pay rate?";

cin >> rate;

employee1.setHourlyRate();

cout << endl << "How many hours do they work per week?";

cin >> hours;

employee1.setHoursWorked(hours);

PrintCheck(\*emp1);

}

else if (userInput == "salary")

{

cout << "What is their annual pay?";

cin >> annual\_pay;

employee1.setAnnualPay(annual\_pay);

employee1.setHourlyRate();

cout << "Set amount of hours worked: ";

cin >> hours;

employee1.setHoursWorked(hours);

PrintCheck(\*emp1);

}

else

{

cout << "What is the company paying " << first\_name << " " << last\_name << "?";

cin >> annual\_pay;

employee1.setAnnualPay(annual\_pay);

cout << endl << "How many hours were worked?";

cin >> hours;

employee1.setHoursWorked(hours);

PrintCheck(\*emp1);

}

\*/

/\*void PrintCheck(HourlyPay emp1)

{

emp1.DisplayEmployee();

}

\*/

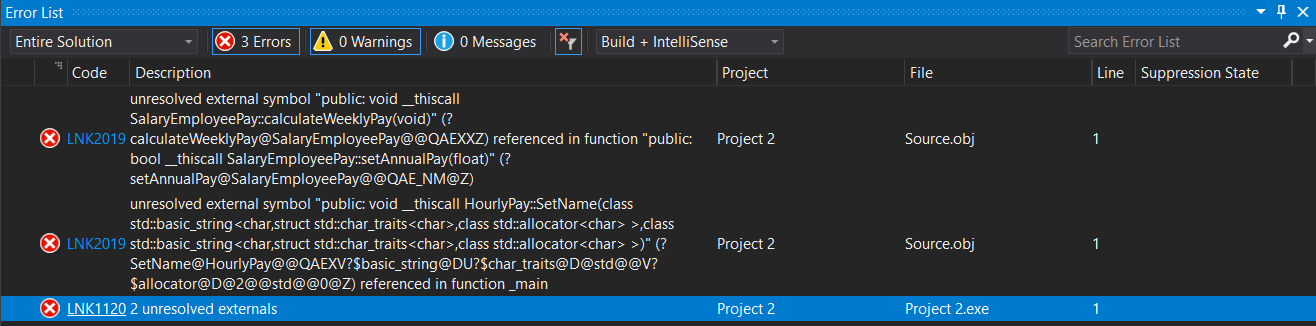
return 0;

}

Final Test Plan:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test # | Valid / Invalid Data | Description of test | Input Value | Expected Output | Actual Output | Test Pass / Fail |
| 1 | Valid Data | Ensure that the program is accepting user input | Input of first name “Ignis,” a last name of “Scientia,” a ssn of “987-65-4321” | The program should store this data and display it to the user as requried | The program will not run, it claims there are fatal external errors | FAIL |

Output Results:



It is because of these fatal errors that I was afraid to turn in the assignment at all. I thought my logic was solid and that I would be able to run my program. My stomach dropped, however, when I was unable to fix these errors, and I found the lab work piling on. I just could not try to go back and fix my mistakes, and for that I know I paid a large price.