**Source Code**

**Salary-Calc-driver.cpp**

#include “Salary-CalculatorClass.cpp”

// start main

*int* main () {

    // create instance of Salary Calculator class

    SalaryCalculator sal\_calc;

    sal\_calc.get\_hrs();

    sal\_calc.get\_pay();

    sal\_calc.get\_ot();

    sal\_calc.calculate();

    sal\_calc.display();

    return 0;

}

**Salary-CalculatorClass.cpp**

/\*

Salary Calculator Program

CSC 450 Programming III

Johnny Olmedo

\*/

#include <iostream>

#include <iomanip>

#include <ostream>

using *namespace* std;

*class* SalaryCalculator

{

*public:*

*void* get\_hrs();

        // this function will get input from user for hours worked

*void* get\_pay();

        // this function will retrieve pay of user

*void* get\_ot();

        // this function will get overtime hours worked by user (if any)

*void* calculate();

        // this function will determine the users salary

*void* display();

        // this function displays salary

*private:*

*double* hours\_worked;

*double* pay\_rate;

*double* overtime;

*double* salary;

*double* \* hours\_PTR;

*double* \* pay\_PTR;

*double* \* ot\_PTR;

*double* \* otPay\_PTR;

*double* \* salary\_PTR;

};

*void* SalaryCalculator::get\_hrs(){

    cout << "Enter hour worked: ";

    cin >> hours\_worked;

    hours\_PTR = &hours\_worked;

}

*void* SalaryCalculator::get\_pay() {

    cout << "Enter Pay Rate: ";

    cin >> pay\_rate;

    pay\_PTR = &pay\_rate;

}

*void* SalaryCalculator::get\_ot() {

    cout << "Enter overtime hours worked: ";

    cin >> overtime;

    ot\_PTR = &overtime;

}

*void* SalaryCalculator::calculate() {

*double* otPay;

    if (\*ot\_PTR > 0) {

        // calculate overtime pay

        otPay = \*ot\_PTR \* (\*pay\_PTR \* 1.5);

        // set salary with overtime pay

        // salary = hours \* pay rate + overtime pay

        salary = \*hours\_PTR \* \*pay\_PTR + otPay;

    }

    else {

        salary = \*hours\_PTR \* \*pay\_PTR;

    }

    // set salary ptr with salary calculated

    salary\_PTR = &salary;

}

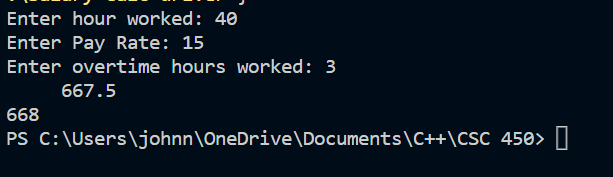
*void* SalaryCalculator::display() {

    cout << setw(10) << \*salary\_PTR << endl;

    cout << setprecision(3) << \*salary\_PTR << endl;

}

**Screenshots:**

****

**Analysis**

The C++ programming language has a couple of different variable types designed to manage text data. Some of these variables are C strings, they can be thought of as an array of characters, and the C++ string data type. These types of variables can be used for printing messages to the console, reading and writing to files, and copying from one memory buffer to another (Howard, 2021). One of the main things I want to focus on while writing c++ programs is to create programs that safely handle input and output from the user and minimize vulnerability. Let’s look at the program to see what we can find.

Main:

#include “Salary-CalculatorClass.cpp”

// start main

*int* main () {

    // create instance of Salary Calculator class

    SalaryCalculator sal\_calc;

    sal\_calc.get\_hrs();

    sal\_calc.get\_pay();

    sal\_calc.get\_ot();

    sal\_calc.calculate();

    sal\_calc.display();

    return 0;

}

As you can see here. The main of this program is a driver file that includes the Salary Calculator class in order to create an instance of the object and access its data. This is a secure way of using object oriented programming in order to call on different data as needed from different files. I don’t see anything wrong at this point. Next let’s look at the class to see what attributes it has.

Class:

*class* SalaryCalculator

{

*public:*

*void* get\_hrs();

        // this function will get input from user for hours worked

*void* get\_pay();

        // this function will retrieve pay of user

*void* get\_ot();

        // this function will get overtime hours worked by user (if any)

*void* calculate();

        // this function will determine the users salary

*void* display();

        // this function displays salary

*private:*

*double* hours\_worked;

*double* pay\_rate;

*double* overtime;

*double* salary;

*double* \* hours\_PTR;

*double* \* pay\_PTR;

*double* \* ot\_PTR;

*double* \* otPay\_PTR;

*double* \* salary\_PTR;

};

We can see here that the class has a handful of public functions that are used to manipulte data in the private variables. This is a secure way of writing code because this means the users data cannot be taken and used by just anyone. Next we will look at the induvial functions.

*void* SalaryCalculator::get\_hrs(){

    cout << "Enter hour worked: ";

    cin >> hours\_worked;

    hours\_PTR = &hours\_worked;

}

*void* SalaryCalculator::get\_pay() {

    cout << "Enter Pay Rate: ";

    cin >> pay\_rate;

    pay\_PTR = &pay\_rate;

}

*void* SalaryCalculator::get\_ot() {

    cout << "Enter overtime hours worked: ";

    cin >> overtime;

    ot\_PTR = &overtime;

}

We see each functions gives instructions for what the programis expecting

And obtains user input with cin. After, the input is stored in the memory space of the variable’s pointer.

*void* SalaryCalculator::calculate() {

*double* otPay;

    if (\*ot\_PTR > 0) {

        // calculate overtime pay

        otPay = \*ot\_PTR \* (\*pay\_PTR \* 1.5);

        // set salary with overtime pay

        // salary = hours \* pay rate + overtime pay

        salary = \*hours\_PTR \* \*pay\_PTR + otPay;

    }

    else {

        salary = \*hours\_PTR \* \*pay\_PTR;

    }

    // set salary ptr with salary calculated

    salary\_PTR = &salary;

}    return 0;

Here is the classes calculation function that adjusts the users salary based on inputs given.

*void* SalaryCalculator::display() {

    cout << setw(10) << \*salary\_PTR << endl;

    cout << setprecision(3) << \*salary\_PTR << endl;

}

Here is the display function that prints the data using precision formulas.

Finally, I believe this code looks secure for the most part. The only thing I would add to improve functionality and security of the code are exception handling, meaning using try and catch blocks to validates the user is entering doubles. Also I would implement the use of printf functions for display. Printf can either be a self-contained string to be printed, copied, and it can draw from other variables to build the final string. A failure to use this format string properly creates vulnerabilities in a C++ application (Howard, 2021).

**References**

Howard Poston. (2021, November 3). Format string vulnerabilities. Infosec Resources. <https://resources.infosecinstitute.com/topic/format-string-vulnerabilities/>