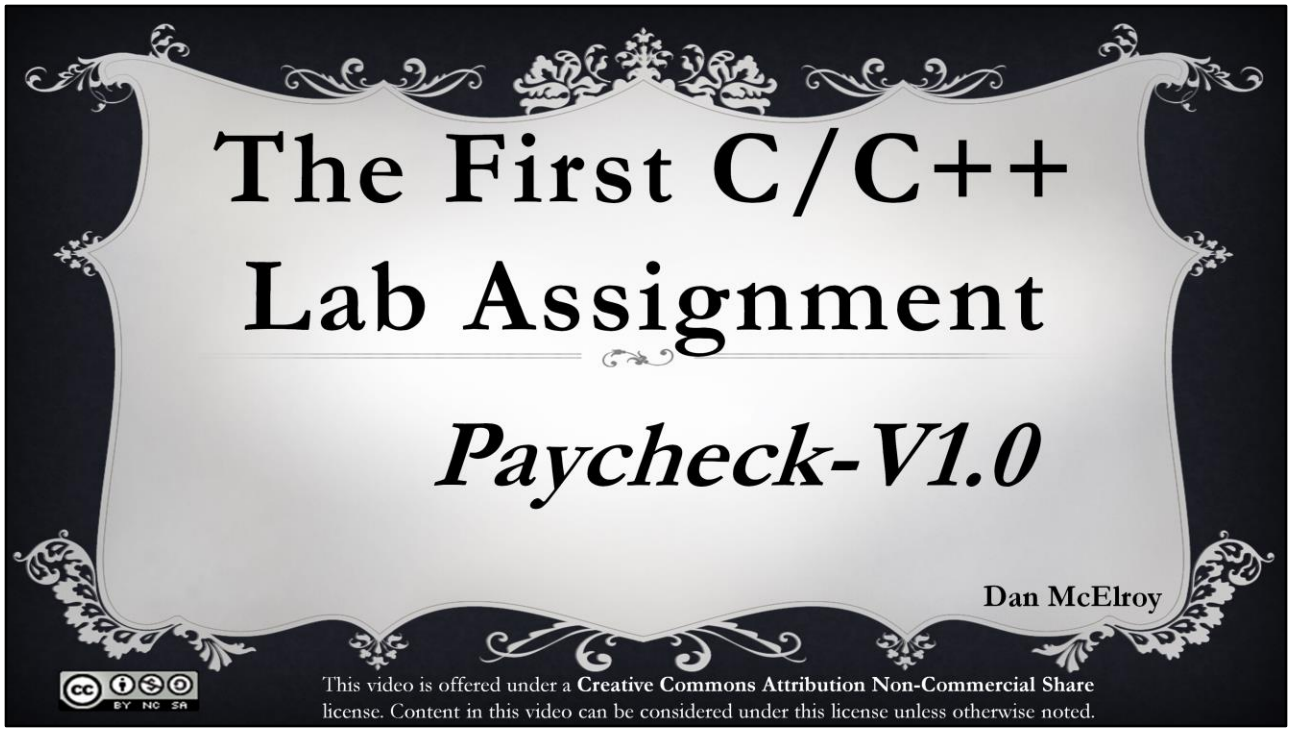


Hello programmers and welcome to your first programming assignment in the class. You will be given a program to type in and then make it work. The discussion in this video gives a detailed description of the inner workings of the Paycheck program. It is not necessary for you to understand everything here. We will cover many of the details of C and C++ later and at much slower pace. I am giving the details of this program now for two reasons. 1) so that you will see why things are done the way they are, and 2) when later we have a dedicated presentation of the individual pieces of this program, you can say to yourself, "Now I understand what he was talking about in the Paycheck program."

List of Topics

PART-1 DISCUSSION OF THE PAYCHECK PROJECT

- The Paycheck v1.0 assignment
- Project definition
- Develop an algorithm
- Create the code in C or C++

PART-2: SPECIFIC INSTRUCTIONS FOR C or C++

- Use a software development system to enter the code
- Run the program and fix any errors

PART-3 SUBMIT THE LAB REPORT

- Complete and submit the lab report

The Paycheck project is presented in three parts.

Part 1 - is this video. Because C and C++ are related and very similar for many parts of the languages, the discussion covers both languages. I will identify anything that is different. You can can choose to do the project in either in C or C++ or both, but you will only get credit for one project submission.

Part 2 - has the instructions on completing the lab project.

Part 3 - complete and submit the lab report.
This first project is to compute a paycheck. Watch this video carefully. It not only covers the first project, but many of the small details. You may want to watch this video more than

one time, especially if this is your first class in programming. A PowerPoint presentation of the slides and script for the video is also available.

As the class progresses, you will be required to start a project from scratch and create the entire project yourself.

Here are a list of topics for the Paycheck project.

- The Paycheck v1.0 assignment defined.
- Project definition.
- Develop an algorithm.
- Create the code in C or C++.

Part-2 has information on how to

- Use a software development system to enter the code.
- Run the program and fix any errors.

Part-3 has information on how to

- Complete and submit the lab report.

After you have created the program, you need to make it work, test it using at least three different values, then complete and submit the lab report.

Before You Even Start

I suggest that you create several folders on your computer or storage device

* Main folder for the CIS054 C/C++ Programming Class
* One subfolder to keep copies of all your lab reports
* Separate subfolder for each lab project
* You may want another folder to keep copies of the class presentations. The college removes the Canvas web page about a month after the end of the class.

Before you even start, I suggest that you create several folders on your computer or storage device.

- Main folder for the CIS054 C/C++ Programming Class.
- One subfolder to keep copies of all your lab reports.
- Separate subfolder to hold all of the lab reports.
- You may want another folder to keep copies of the class presentations. The college removes the Canvas web page about a month after the end of the class.

1 - Project Definition

Create a program using C or C++ that does the following:

Read the number of hours worked and pay rate from the keyboard.
Separate the number of hours entered into
regular hours, which are less than or equal to 40, and
overtime hours of anything over 40.
Compute the pay for the regular hours.
Compute the pay for the overtime hours at time-and-a-half.
Compute total pay which is pay for regular hours + overtime pay.
Display regular pay, overtime pay and total pay.

I am giving you all of the code both in C or C++ for this project. All you need to do is type it in and make it work. Here is the definition for the Paycheck project.

Create a program using C or C++ that does the following.

Read the number of hours worked and pay rate from the keyboard.
Separate the number of hours entered into regular hours,
which are less than or equal to 40, and
overtime hours which are anything over 40.
Compute the pay for the regular hours.
Compute the pay for the overtime hours at time-and-a-half.
Compute the total pay which is pay for the regular hours

+ overtime pay.
Display regular pay, overtime pay and total pay.

Identify the Inputs and Outputs

Read the number of **hours** worked and **pay rate** from the keyboard.

Separate the number of hours entered into
regular hours, which are less than or equal to 40, and
overtime hours of anything over 40.

Compute the pay for the regular hours.

Compute the pay for the overtime hours at time-and-a-half.

Compute total pay which is pay for regular hours + overtime pay.

Display **regular pay**, **overtime pay** and **total pay**.

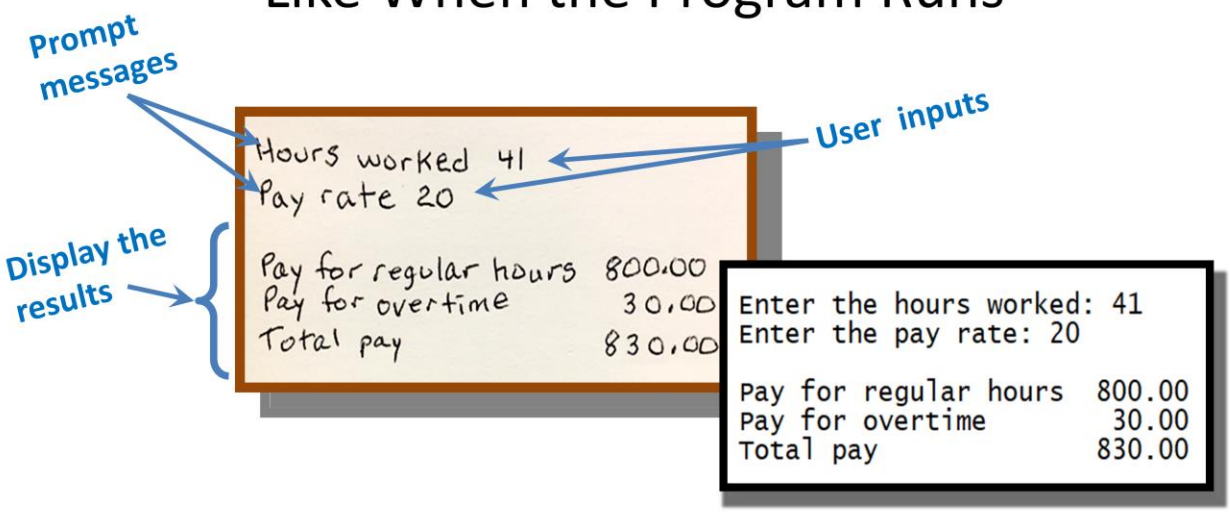
The first thing you should do when you get a programming project is to identify the inputs, the outputs and then the processing needed to go from the inputs to the outputs.

The inputs to the program are from the keyboard. They are **hours** and **pay rate**.

The outputs are to be displayed on the monitor. They are **regular pay**, **overtime pay** and **total pay**.

Once we know the inputs and outputs, then we need to figure out the computations needed to get from the inputs to the outputs.

1 – Outline What the Screen Should Look Like When the Program Runs



We should also get an idea of what we want the screen to look like when the program runs. Draw it out on paper. As the project progresses, don't worry if the actual program output does not look exactly like your original plan. Plans can change.

Here is a sample of running the program with 41 hours at $20.00 per hour. Since there are 41 hours, 40 hours are paid at the regular pay of $20.00 per hour and there is 1 hour of overtime paid at $30.00/hour because overtime is paid at time-and-a-half.

Legal Names for Variables

Good English	Good Programming
hours worked	hours or hoursWorked
pay rate	payRate
regular hours	regHours or regularHours
overtime hours	overtimeHours or otHours
regular pay	regPay or regularPay
overtime pay	overtimePay or otPay
total pay	totalPay

When we start writing the actual program, legal names need to be used for our variables and constants. Make variable names so that they make sense both to you and anyone who might be reading the program later. The only legal characters for names are capital A-Z, small a-z, the digits 0-9 and the underscore _ character. Names you create can't start with digits 0-9 and should not start with the underscore _. Multiple English words can be combined together to form a variable name. To make it easier to see the individual words that make up a variable name, use either Camel-case or Snake-case. I am using Camel-case here. The first letter in the variable name is lower-case. Each English word that follows in the variable name is capitalized. For example the English regular hours becomes regHours.

I am giving you all of the code for the Paycheck program, including legal variable names.

The names I have chosen are **hours, payRate, regHours, overtimeHours, regPay, overtimePay** and **totalPay.**

If you want to experiment, you can choose any meaningful name for each variable. For example, you could use regularHours instead of regHours. You don't need to use the exact same names that I used, but whatever name you choose, it must be used consistently throughout the program.

Separate hours into regular hours and overtime hours

Hours worked	Regular hours	Overtime hours
35	35	0
40	40	0
45	40	5

Overtime gets paid at time and a half. Since there is a different pay rate for regular hours and overtime hours, it is necessary to separate the number of hours worked into **regHours** and **overtimeHours**.

Here are three examples of overtime for anything over 40 hours. If people work 35 hours, then there is no overtime. If people work 40 hours, still there is no overtime. However, if people work 45 hours, then the pay is computed at the their pay rate for the first 40 hours, but the program needs to make a separate computation for the 5 hours of overtime.

2 - Paycheck – Develop Algorithm

Develop an algorithm in pseudo-code. Use a HIPO (**H**ierarchical **I**nput **P**rocess **O**utput) chart to define the input, process and output for the program.

Pseudocode is fake code its code it's not really in any language. It's more in English. A lot of times, pseudocode is given to the programmer to say, “here's how I want the program to work,” and it's up to the programmer to write the code in C, C++, Java or what ever language is being used.

3 - Paycheck – Develop Algorithm

INPUT	PROCESS	OUTPUT
hours **pay rate**	Read **hours** from keyboard Read **pay rate** from keyboard Compute the number **of regular hours** (less or equal to 40) Compute the number of **overtime hours** (hours over 40) Compute the **pay** for the **regular hours** Compute the **pay** for the **overtime hours** Compute the **total pay** (regular pay + overtime pay) Display **regular pay, overtime pay and total pay**	**regular pay** **overtime pay** **total pay**

In this hierarchy, input, process, output chart, also called a HIPO chart we want to define the inputs (in this case it's hours and pay rate) and the outputs (regular pay, overtime pay and total pay)

Once we know the input and the output, we then need to decide how to get from input all the way to the output.

We're going to read the hours and pay rate from the keyboard, determine the number of regular hours, determine the number of overtime hours, compute the regular pay, overtime pay and total pay.

Finally, display the regular pay, overtime pay and total pay.

Sample Computation -- less than 40 hours

	Name	Input	Compute	Value	Output to monitor	
1	hours	41				
2	payRate	20.00				
3	regHours		up to 40 hours	40		
4	overtimeHours		anything over 40)	1		
5	regPay		regHours * payRate	800.00		
6	overtimePay		overtimeHours * payRate * 1.5	30.00		
7	totalPay		regPay + overtimePay	800.00		
8					Regular pay	800.00
9					Overtime pay	30.00
10					Total pay	830.00

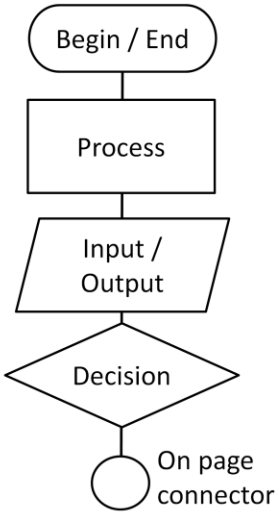
This chart shows the sequence of events that occur as the program executes with an input for hours is 41 and the input for pay rate is 20.

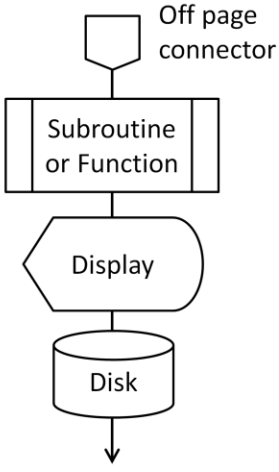
Since 41 hours were input and overtime is computed at time and a half for anything over 40, the hours are separated into regular hours equal to 40 and overtime hours at 1. Once we have separated the hours, we can compute the pay for each category. Regular pay is equal to 800 which is 40 * 20. The overtime pay is 30 hours because there is 1 hour times the pay rate times 1.5 to get time and a half. The regular pay and overtime pay are added together to get the total pay.

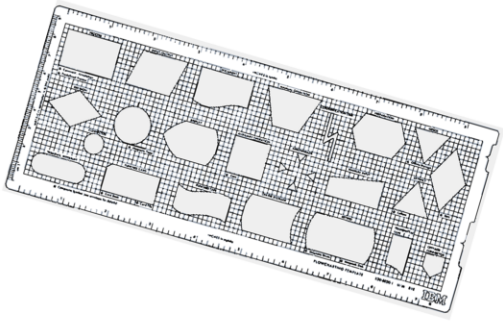
Finally, the regular pay, overtime pay and total pay can be output to the monitor.

In your mind, keep the program organized to INPUT, PROCESS and OUTPUT.

Flowchart Symbols

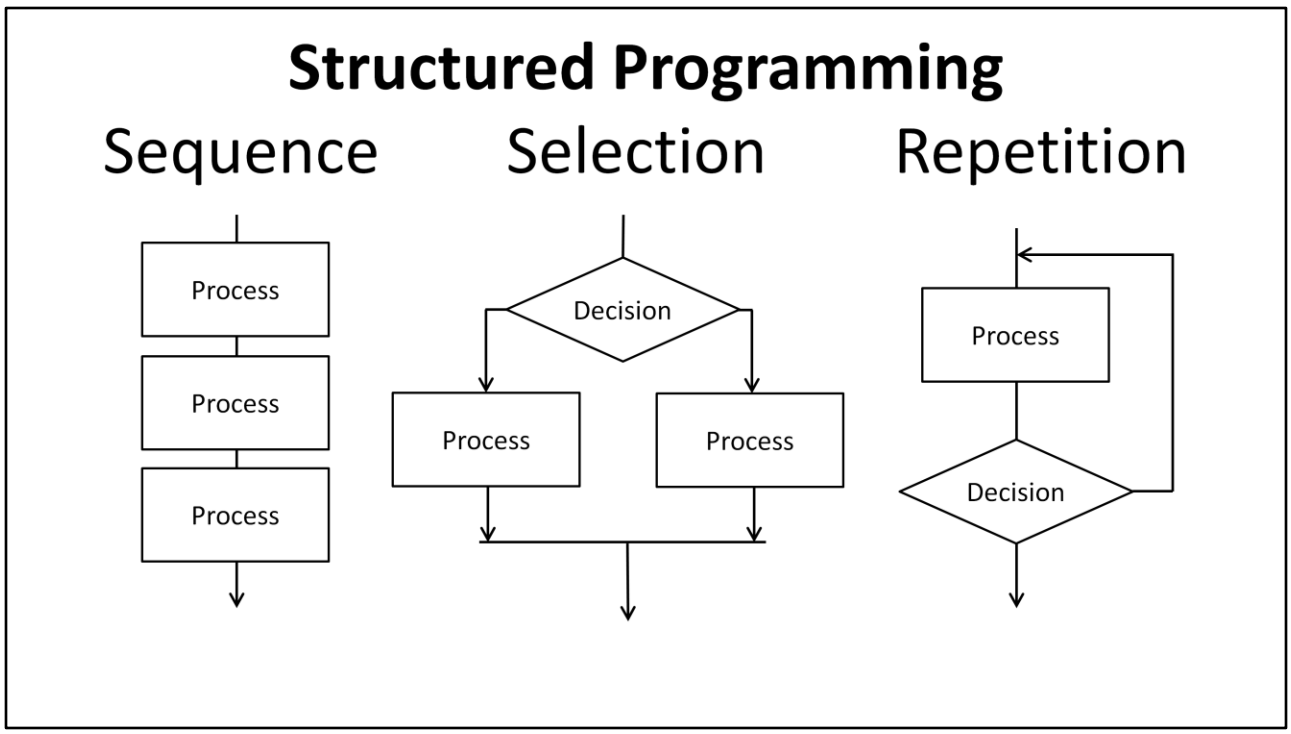




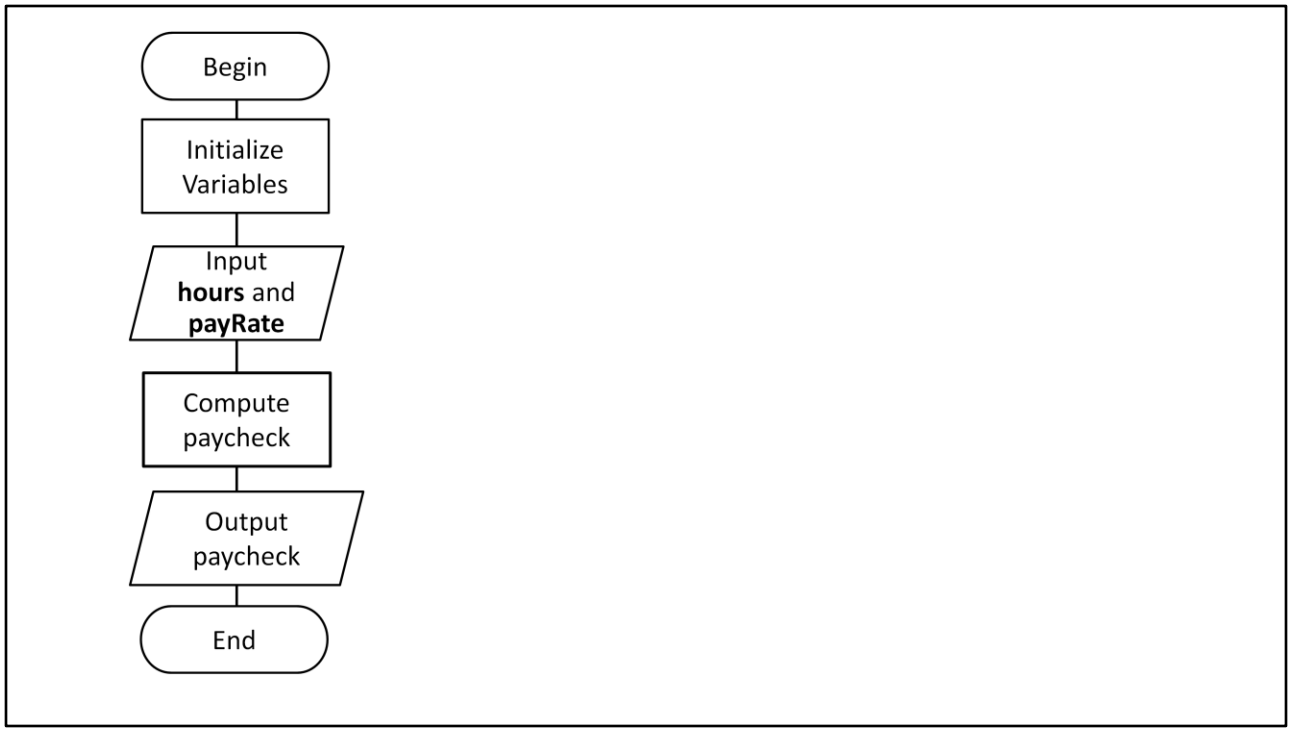


I am going to use flowcharts to continuously breakdown the program into smaller manageable parts. Flowcharts are used to represent different steps of a program. The most commonly used symbols are those in the left column. A circle or rounded rectangle is used to indicate the beginning or end. The parallelogram is used for input or output. The rectangle represents some type of process and the diamond is used to represent a decision. Other symbols can be used to represent subroutines, functions, disks, databases, etc.

Structured Programming

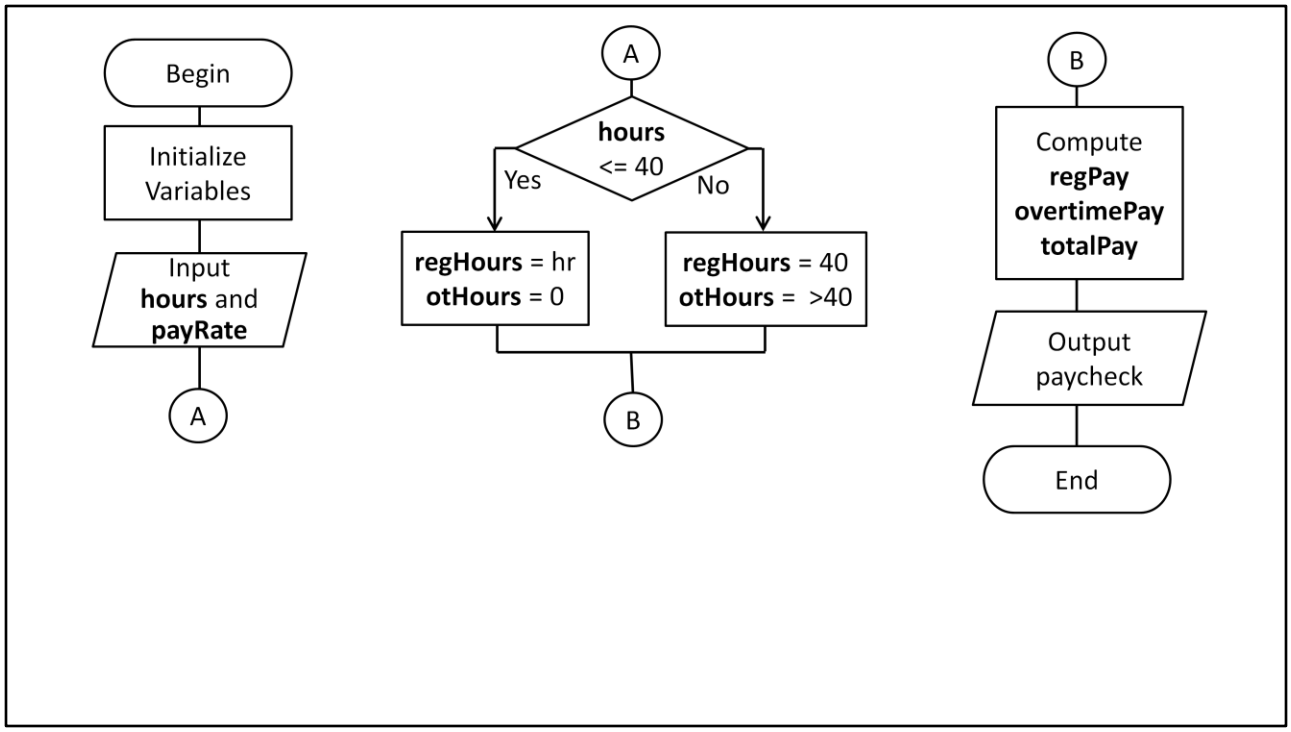


The concept of structured programming states that all programs can be created with only three constructs: Sequence, Selection and Repetition or a combination of any of them. Sequence is the easiest to understand. Program steps are executed sequentially, one after another. Many times, the order in which the steps are executed is very important. Other times it does not matter. For example, if I were baking a cake, it may not matter very much whether I put the egg in the bowl and then the flour or I reversed the two before mixing them with water. However, it would make a lot of difference if I put the cake batter in a 9 x 13 baking dish and placed it in an oven for 1 hour and 30 minutes, took it out and then turned the oven on to 350 degrees. Yuck.



This simplified flowchart shows the sequence of steps that are required for the program. There are no loops, so the repetition structure is not needed.

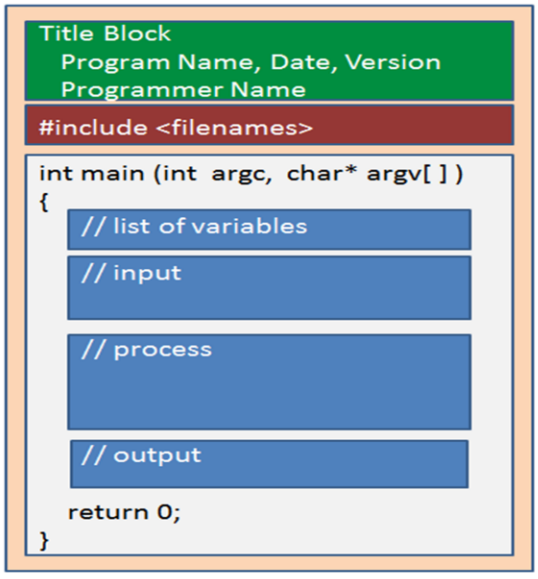
We can expand the "Compute Paycheck" box to show a lot more detail.



Here I have expanded the flowchart to show more detail when computing the paycheck. The center column uses a diamond to represent a decision. The program needs to look at all the hours worked to see if they are less or equal to 40. In this case, the **regHours** are set to the number of hours worked and the **overtimeHours** are set to zero.

However, if the test condition is NOT TRUE that the hours are less or equal to 40, that means that there is some overtime. Therefore, we set the regular hours to 40 and set the overtime hours to anything over 40.

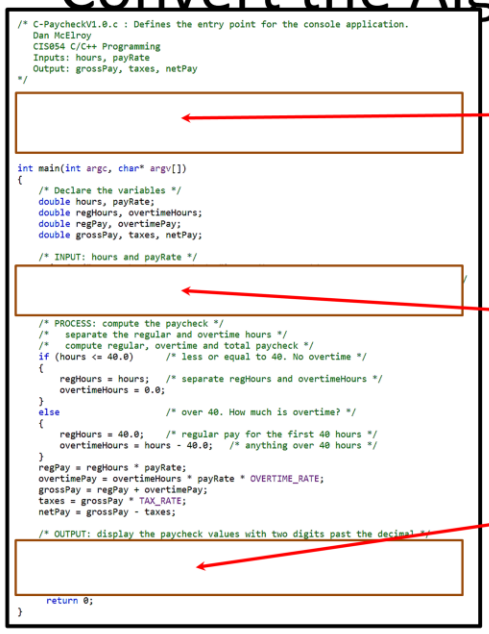
We can then proceed to compute the regular pay, the overtime pay and total pay – and then display them.



PROGRAM ORGANIZATION: Most of the programs you write will be organized like this. Use comments to place a title block at the top of each file. The title block identifies what this part of the program does as well as the date, version and programmer's name. It can be very frustrating later on when there are many files that make up one program and you look at a file that contains code and have no idea what the code does. You then need to re-analyze or 'reverse-engineer' the code instead of just looking at a short description in the title block. Many companies want the programmers to place their name in the file in case anybody has questions about the file. The date and version number also help identify the latest version of the program.

The first part of the real code contains the #include statements and any global variables and constants. Next is the main body of the program with its four sub-sections: List of variables -Input -Process -Output.

4 – Convert the Algorithm into Code



```
/* C-PaycheckV1.0.c : Defines the entry point for the console application.
   Dan McElroy
   CIS054 C/C++ Programming
   Inputs: hours, payRate
   Output: grossPay, taxes, netPay
*/

int main(int argc, char* argv[])
{
    /* Declare the variables */
    double hours, payRate;
    double regHours, overtimeHours;
    double regPay, overtimePay;
    double grossPay, taxes, netPay;

    /* INPUT: hours and payRate */

    /* PROCESS: compute the paycheck */
    /*   separate the regular and overtime hours */
    /*   compute regular, overtime and total paycheck */
    if (hours <= 40.0)      /* less or equal to 40. No overtime */
    {
        regHours = hours;   /* separate regHours and overtimeHours */
        overtimeHours = 0.0;
    }
    else                    /* over 40. How much is overtime? */
    {
        regHours = 40.0;    /* regular pay for the first 40 hours */
        overtimeHours = hours - 40.0;   /* anything over 40 hours */
    }
    regPay = regHours * payRate;
    overtimePay = overtimeHours * payRate * OVERTIME_RATE;
    grossPay = regPay + overtimePay;
    taxes = grossPay * TAX_RATE;
    netPay = grossPay - taxes;

    /* OUTPUT: display the paycheck values with two digits past the decimal */

    return 0;
}
```

The #include files and constants go here. The include files will be different for C and C++. They help the compiler build the program.

The request for input (prompt) and keyboard input go here. The C language uses the printf and scanf functions for output and input. C++ uses cout and cin.

The code to display the amount of the paycheck goes here. The C language uses printf and C++ uses cout.

Next, convert the algorithm into code. The C and C++ programs are very similar. All of the logic for determining the regular and overtime hours, computing the regular pay, overtime pay and total payare exactly the same for both languages. The big difference at this point is just the input and output statements.

I have placed square boxes where there is a difference in the C and C++ code. The first one is for the include files.
There is a slight difference between the C and C++ include files. Include files help the compiler build the program.

The second one is for inputting data, the hours worked and the pay rate.

The last box is for the output.

```c
/* C-PaycheckV1.0.c
   Dan McElroy
   CIS054 C/C++ Programming
   Inputs: hours, payRate
   Output: regPay, overtimePay, totalPay
*/

#include <stdio.h>  /* used for scanf and printf */

/* define the constants */
#define OVERTIME_RATE 1.5  /* time and a half for overtime */

int main(int argc, char* argv[])
{
    // Declare the variables
    double hours, payRate;              // values input from the keyboard
    double regHours, overtimeHours;     // the rest of the values are computed
    double regPay, overtimePay;
    double totalPay;

    /* INPUT: hours and payRate */
    printf  ("Enter the hours worked: ");    /* prompt */
    /* your compiler may need scanf instead of scanf_s */
    scanf_s ("%lf", &hours);
    printf  ("Enter the pay rate: ");
    scanf_s ("%lf", &payRate);

    /* PROCESS: compute the paycheck */
    /*   separate the regular and overtime hours */
    /*   compute regular, overtime and total paycheck */
    if (hours <= 40.0)      /* less or equal to 40. No overtime */
    {
        regHours = hours;   /* separate regHours and overtimeHours */
        overtimeHours = 0.0;
    }
    else                    /* over 40. How much is overtime? */
    {
        regHours = 40.0;    /* regular pay for the first 40 hours */
        overtimeHours = hours - 40.0;   /* anything over 40 hours */
    }
    //   compute regular, overtime and total pay
    regPay = regHours * payRate;
    overtimePay = overtimeHours * payRate * OVERTIME_RATE;
    totalPay = regPay + overtimePay;

    /* OUTPUT: display the paycheck values with two digits past the decimal */
    printf ("\n");  /* blank line before the output */
    printf ("Pay for regular hours %7.2lf\n", regPay);
    printf ("Pay for overtime      %7.2lf\n", overtimePay);
    printf ("Total pay             %7.2lf\n\n", totalPay);

    return 0;
}
```

Is a Programming Language really a Language?

Is a Programming Language really a Language? It isn't a spoken language, but neither is ASL, American Sign Language. If I was writing an English essay, I would construct it using sentences and paragraphs. I would separate the paragraphs by either indenting the first word or placing a blank line between paragraphs. A typical sentence would have a subject, verb and object. Each **statement** in a program is like a sentence in English. If I have a statement x = sqrt(25); I could express it in English as compute the square root of 25 and store it in the variable named x. This is a compound English statement. Verbs are **compute** and **store**. The object of the verb compute is the number **25**. The second part of the statement, "store it in the variable named x" has an implied subject based on the result of the square root computation. Sentences in English are ended

with a period, but the period-character is already used in C and C++ as a decimal point for numbers, so the creators of C chose to use a semicolon to end a statement.

The syntax for sentences varies with different human languages. The syntax of program statements varies between programming languages.

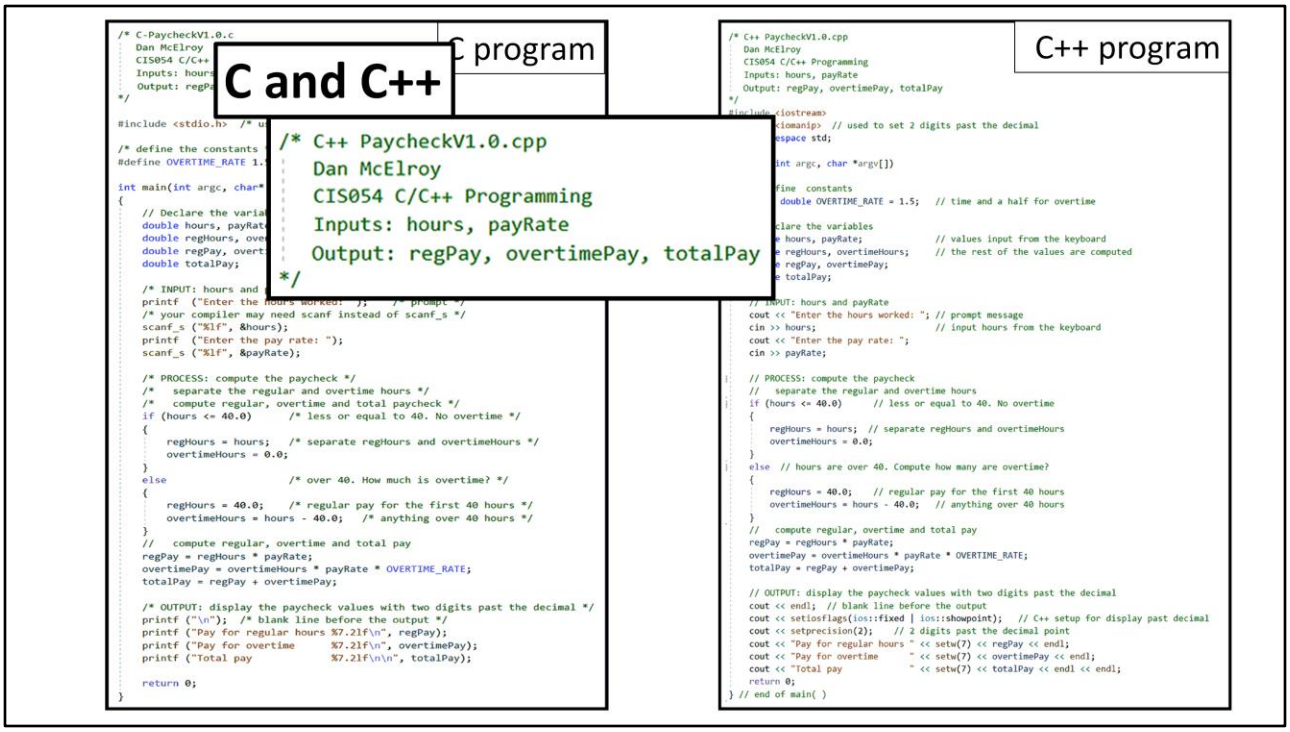
C program

```c
/* C-PaycheckV1.0.c
   Dan McElroy
   CIS054 C/C++ Programming
   Inputs: hours, payRate
   Output: regPay, overtimePay, totalPay
*/

#include <stdio.h>  /* used for scanf and printf */

/* define the constants */
#define OVERTIME_RATE 1.5  /* time and a half for overtime */

int main(int argc, char* argv[])
{
    // Declare the variables
    double hours, payRate;              // values input from the keyboard
    double regHours, overtimeHours;     // the rest of the values are computed
    double regPay, overtimePay;
    double totalPay;

    /* INPUT: hours and payRate */
    printf  ("Enter the hours worked: ");    /* prompt */
    /* your compiler may need scanf instead of scanf_s */
    scanf_s ("%lf", &hours);
    printf  ("Enter the pay rate: ");
    scanf_s ("%lf", &payRate);

    /* PROCESS: compute the paycheck */
    /*   separate the regular and overtime hours */
    /*   compute regular, overtime and total paycheck */
    if (hours <= 40.0)      /* less or equal to 40. No overtime */
    {
        regHours = hours;   /* separate regHours and overtimeHours */
        overtimeHours = 0.0;
    }
    else                    /* over 40. How much is overtime? */
    {
        regHours = 40.0;    /* regular pay for the first 40 hours */
        overtimeHours = hours - 40.0;   /* anything over 40 hours */
    }
    //   compute regular, overtime and total pay
    regPay = regHours * payRate;
    overtimePay = overtimeHours * payRate * OVERTIME_RATE;
    totalPay = regPay + overtimePay;

    /* OUTPUT: display the paycheck values with two digits past the decimal */
    printf ("\n");  /* blank line before the output */
    printf ("Pay for regular hours %7.2lf\n", regPay);
    printf ("Pay for overtime      %7.2lf\n", overtimePay);
    printf ("Total pay             %7.2lf\n\n", totalPay);

    return 0;
}
```

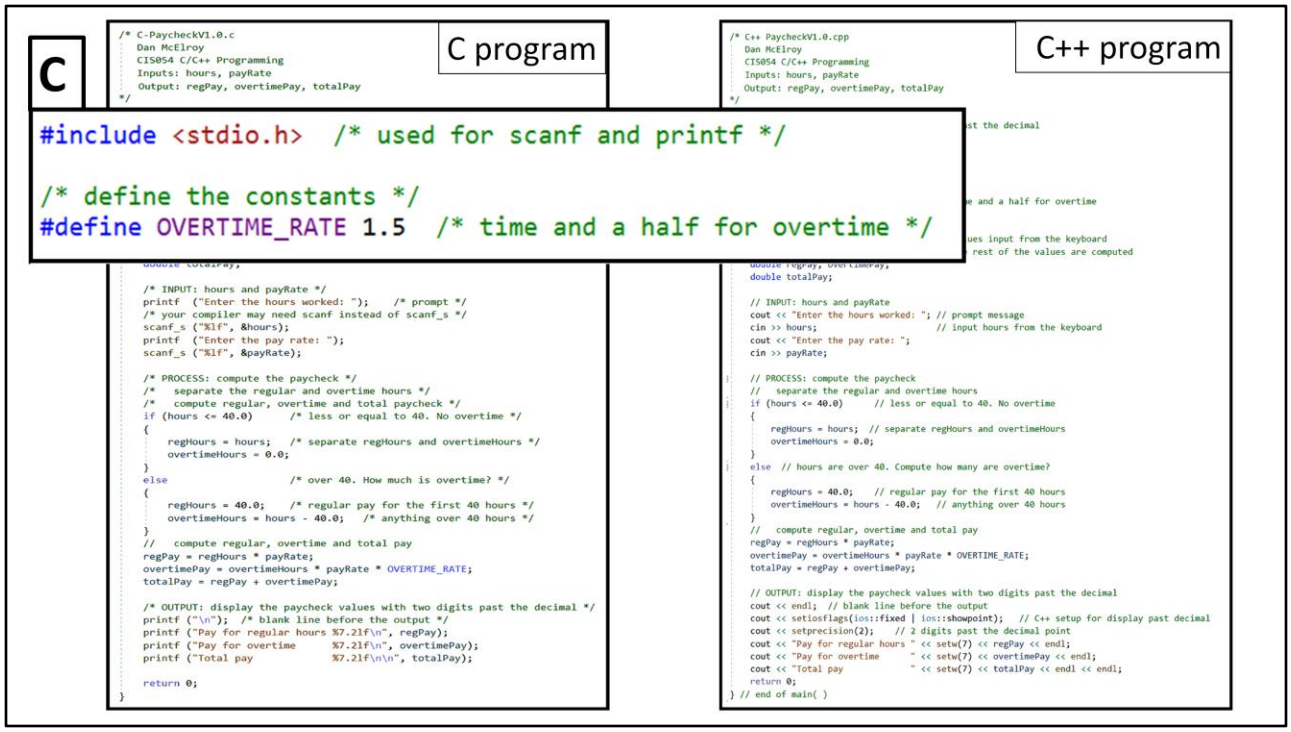
C++ program

```cpp
/* C++ PaycheckV1.0.cpp
   Dan McElroy
   CIS054 C/C++ Programming
   Inputs: hours, payRate
   Output: regPay, overtimePay, totalPay
*/
#include <iostream>
#include <iomanip>  // used to set 2 digits past the decimal
using namespace std;

int main(int argc, char *argv[])
{
    // define  constants
    const double OVERTIME_RATE = 1.5;   // time and a half for overtime

    // Declare the variables
    double hours, payRate;              // values input from the keyboard
    double regHours, overtimeHours;     // the rest of the values are computed
    double regPay, overtimePay;
    double totalPay;

    // INPUT: hours and payRate
    cout << "Enter the hours worked: "; // prompt message
    cin >> hours;                       // input hours from the keyboard
    cout << "Enter the pay rate: ";
    cin >> payRate;

    // PROCESS: compute the paycheck
    //   separate the regular and overtime hours
    if (hours <= 40.0)      // less or equal to 40. No overtime
    {
        regHours = hours;  // separate regHours and overtimeHours
        overtimeHours = 0.0;
    }
    else  // hours are over 40. Compute how many are overtime?
    {
        regHours = 40.0;    // regular pay for the first 40 hours
        overtimeHours = hours - 40.0;   // anything over 40 hours
    }
    //   compute regular, overtime and total pay
    regPay = regHours * payRate;
    overtimePay = overtimeHours * payRate * OVERTIME_RATE;
    totalPay = regPay + overtimePay;

    // OUTPUT: display the paycheck values with two digits past the decimal
    cout << endl;  // blank line before the output
    cout << setiosflags(ios::fixed | ios::showpoint);   // C++ setup for display past decimal
    cout << setprecision(2);    // 2 digits past the decimal point
    cout << "Pay for regular hours " << setw(7) << regPay << endl;
    cout << "Pay for overtime      " << setw(7) << overtimePay << endl;
    cout << "Total pay             " << setw(7) << totalPay << endl << endl;
    return 0;
} // end of main( )
```

This slide shows the program written in both the C and C++. If the text is too small, don't worry about reading it yet. I will zoom in for different parts the during the discussion.



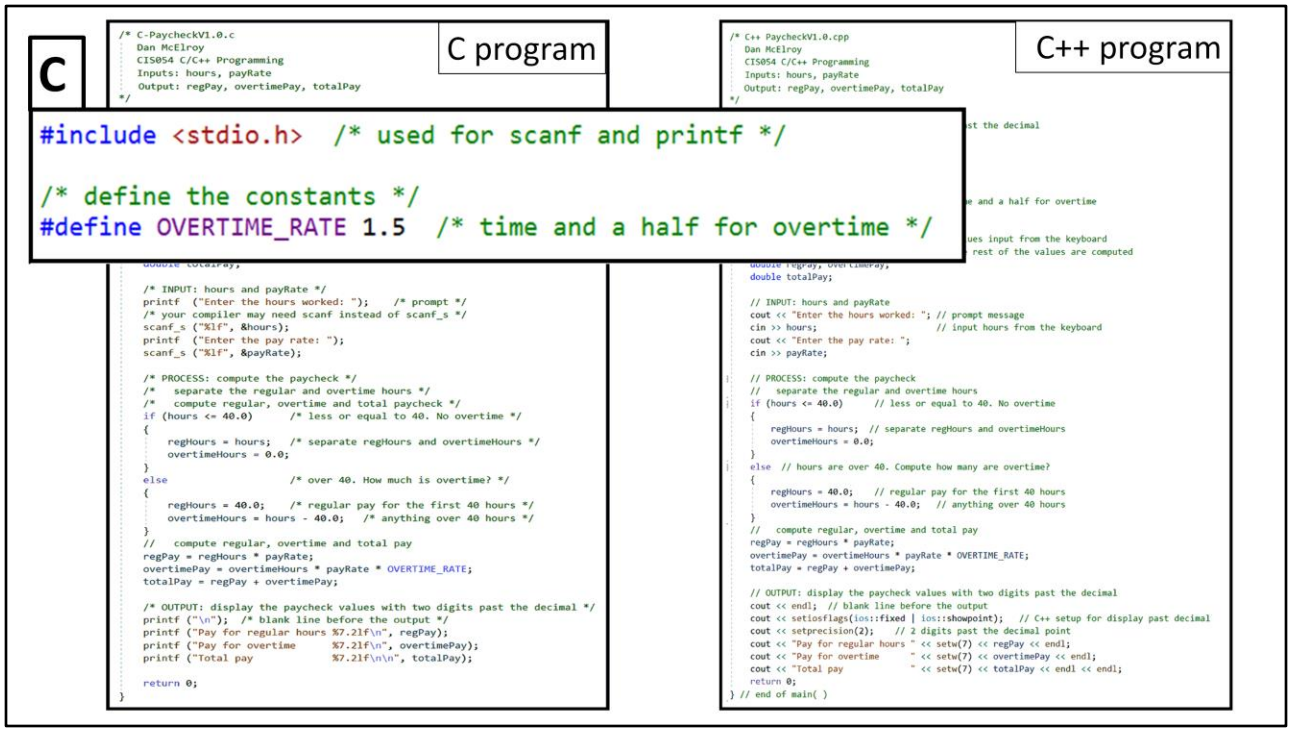
The title block at the top of the program file uses the C-language style comments, starting with a /* and ending with a */



The #include statements brings in additional code that helps the compiler build your program. Look carefully, there is no semicolon ; at the end of the #include statement.

The C-language needs #include <stdio.h> to correctly compile the scanf and printf statements used for input and output. It is proper to pronounce stdio.h either by its letters S T D I O . H or call it standard I O . H but not studio.h There is no U in stdio.h and this has nothing to do with a rock-n-roll recording studio.

Look at the C-style comments /* --- */ They are used for creating a block of comments. This style of comments can be used in both C and C++ programs.

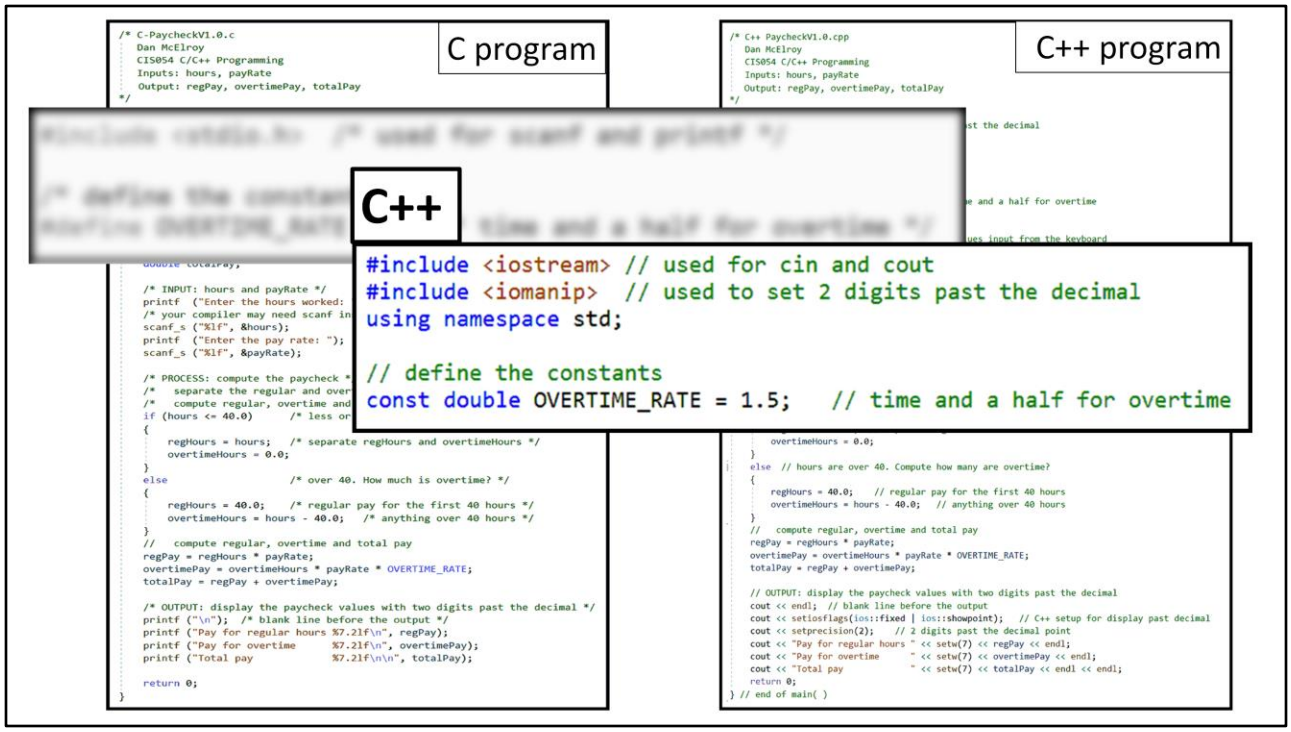


The #define statement is a pre-processor directive used in C to define constants. It causes a text replacement while the program is being compiled. This would be similar to the Find-and-Replace feature in a word processor. Anywhere that the characters OVERTIME_RATE appear in the program, they would be replaced with the characters 1.5 .

Because the #define is a text replacement, it is important that neither the equal sign = nor the semicolon character ; are used in the #define statement. Otherwise those extra characters would also be added into the program before it is compiled.

Although not required in C or C++, it is common practice to use all UPPER_CASE characters and the underscore _ when defining constants. This helps make them easier to recognize when looking at

the code later.

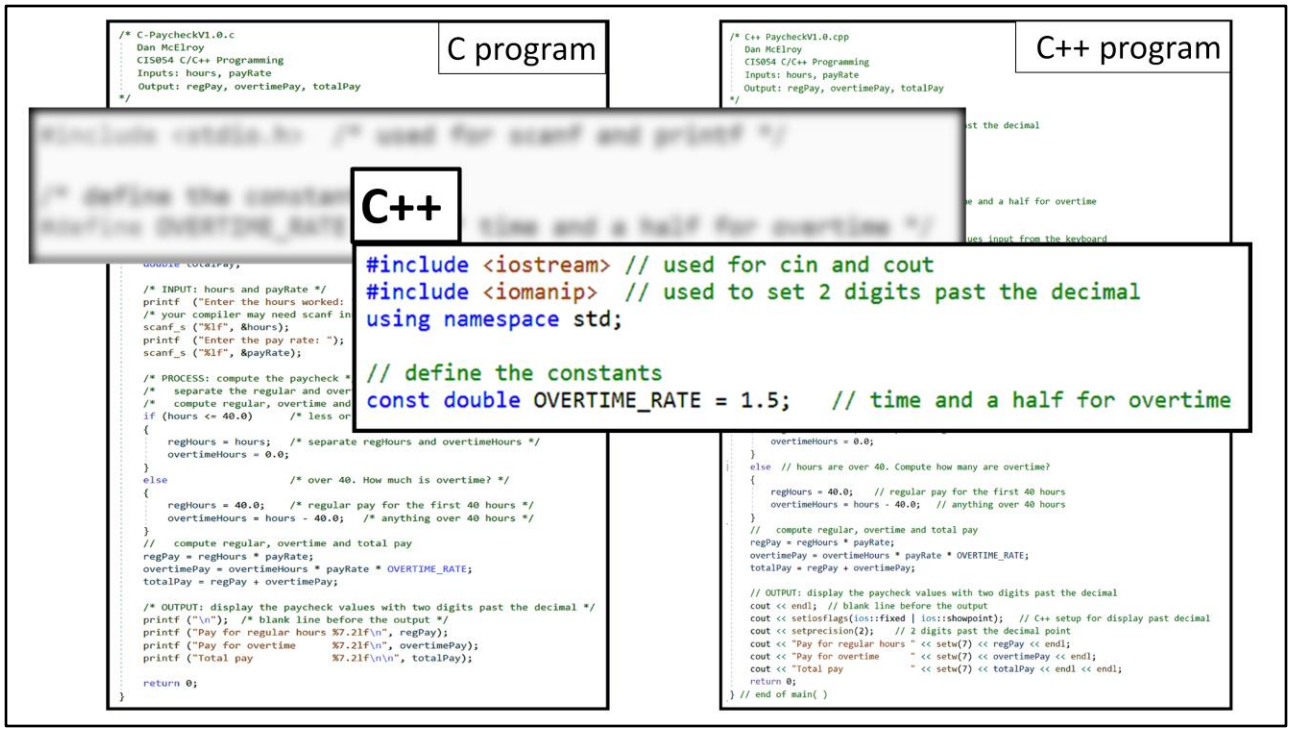


Now, look at the C++ version of the program. You will probably see the **#include <iostream>** statement in many of the C++ programs. The **iostream** library is provided by the compiler manufactures to give access to the console device's keyboard and display. We also use **#include <iomanip>** in this program because we want to display the **regular pay, overtime pay and total pay** with two digits past the decimal.

using namespace std; makes it easier when writing the console input and output statements. For example in C++, if we want to output **Hello world** to the console display, we use **std::cout << "Hello world" << std::endl;** The **std::** tells the compiler to find **cout** in the standard namespace. By placing **using namespace std;** at the top of the program,

we don't need to use **std::** with each cout, endl or cin. Now the statement can just be written as **cout << "Hello world" << endl;**

Look at the C++ style comments // C++ style comments start with the // and end at the end of the line. Although this style of comment was introduced with C++, most modern C compilers



The **const** keyword is used in C++ to prevent the defined value from being changed when the program is executed.

const double OVERTIME_RATE = 1.5; // time and a half for overtime

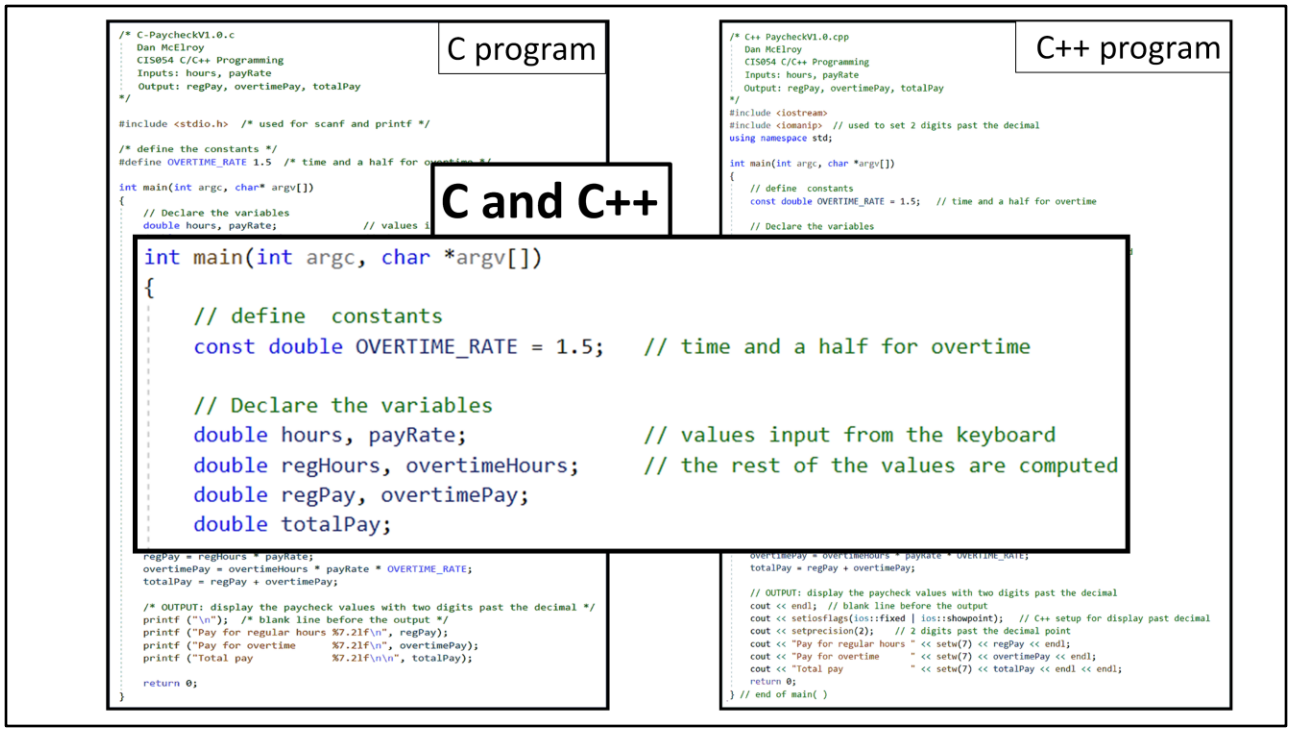
Instead of a text replacement, we are actually creating data with its associated data type, and using the equal-sign = to assign it a value. The semicolon is used at the end of the assignment statement. We now have the value 1.5 stored in memory and it has a data type of double.

Console applications start with **int main**. You can use either **int main (int argc, char* argv[ ])** or **int main ( )**. The argc and argv parameters are only needed when a different program is passing data directly into the program you are writing.

The curly-braces **{ }** are used to identify the start and end of a block of code or data. Many other programming languages use the words **Begin** and **End** instead of curly-braces. I suspect that the original developers of the C-languages did not like to do a lot of typing and also thought, "Wow, I wonder what I can use this character for?"

Several characters always occur in pairs. Examples are parentheses ( ), curly-braces { }, double-quotes ", single-quotes ', square brackets [ ] and angle brackets < > when not used as

arithmetic compare operators for less-than < or greater-than >. Most context sensitive editors used for writing programs will identify the mating character of a pair when you select one of the characters. This can be handy when looking at code.



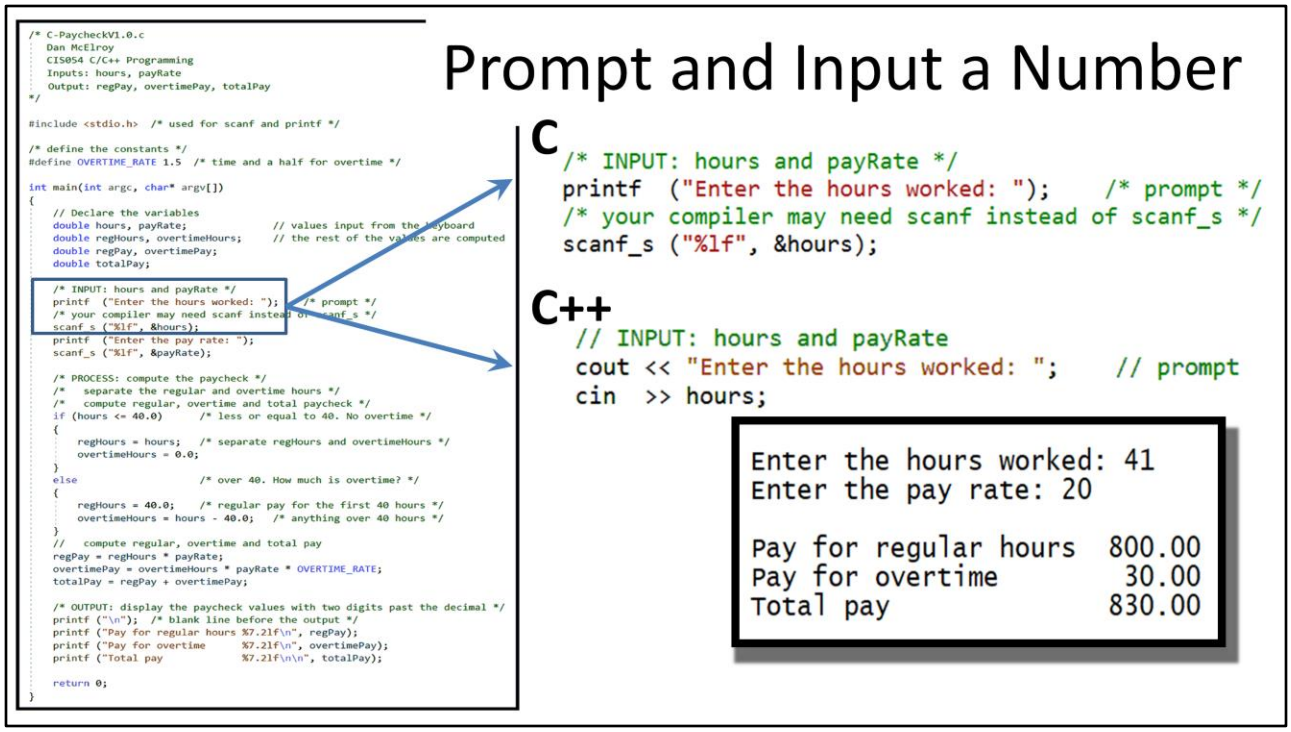
This program is using the **double** data type for variables. The **double** data type is most often used to hold floating point numbers. **double** means double-precision floating point number.

```
double  hours;
double  payRate;
double  regHours;
double  overtimeHours;
double  regPay;
double  overtimePay;
double  grossPay;
double  taxes;
double  netPay;
```

```
double hours, payRate;
double regHours, overtimeHours;
double regPay, overtimePay;
double grossPay, taxes, netPay;
```

Variables can be declared individually, one per line, or multiple variables of the same data type can be declared on the same line. Some companies require each variable on its own line. If you do place more than one variable on a line, they should at least be related to each other.

The first column shows each variable on its own line. The second column shows more than one variable declared on some of the lines. Depending on where you work, some companies require that each variable be declared on its own line.



When writing a console application, it is necessary to output a **prompt** to the screen to request data from the program's user. If you don't have a prompt message, all the user will see is a flashing cursor, won't know what to do and will probably think that the program crashed.

C-programs use the **printf** statement to output to the console screen and **scanf** or **scanf_s** to read from the console keyboard.

C++ programs use **cout** to output to the console screen and **cin** to read from the console keyboard.

C and C++ Input and Output Routines

C Language Input/Output

scanf – formatted input scan
pronounced Scan-F
printf – print formatted
pronounced Print-F
(old computer terminals used a printing Teletype®)

C++ Language Input/Output

cin – Console IN-put
pronounced See-In (not like SIN)
cout – Console OUT-put
pronounced See-Out (not KOWT)

If you want to sound like a cool programmer who is 'in the know', watch how you pronounce these names:

scanf = Scan-F
printf = Print-F
cin = See-In
cout = See-out

scanf is short for "formatted input scan"
printf is short for "formatted print"

Regardless of whether you want to concentrate in C or C++, you should learn how to use printf. It appears in Java and many other languages.

C++ uses cin and cout for inputting and outputting to the console. If you pronounce **cin** as **SIN**, people might think you are trying to lead a church congregation instead of writing a C++ program. **c-o-u-t** is pronounced as See-Out, not KOUT.

scanf vs. scanf_s

```
/* INPUT: hours and payRate */
printf  ("Enter the hours worked: ");    /* prompt */
/* your compiler may need scanf instead of scanf_s */
scanf_s ("%lf", &hours);
```

The **scanf** statement in a C-program is a little complicated to understand. The first thing to notice is that I am using **scanf_s** instead of just plain **scanf**. The plain scanf has been deprecated, which means that it is old and out of date but is still used by many versions of C, including in Apple's Xcode. The original **scanf** could cause problems when reading a string of text characters if someone typed more characters than scanf was expecting. This could cause a buffer overflow resulting in the program to crash or provide a way for evil people to insert a virus.

Microsoft has updated their version of **scanf** and called it **scanf_s**. The **_s** means 'secure'. Apple has updated their version of **scanf** but still calls it **scanf**. You may need to modify **scanf_s** to **scanf** in the example code I am

providing if you get a compiler error when trying to build your program.

Prompt and Input a Number

C

```c
/* INPUT: hours and payRate */
printf  ("Enter the hours worked: ");    /* prompt */
/* your compiler may need scanf instead of scanf_s */
scanf_s ("%lf", &hours);
```

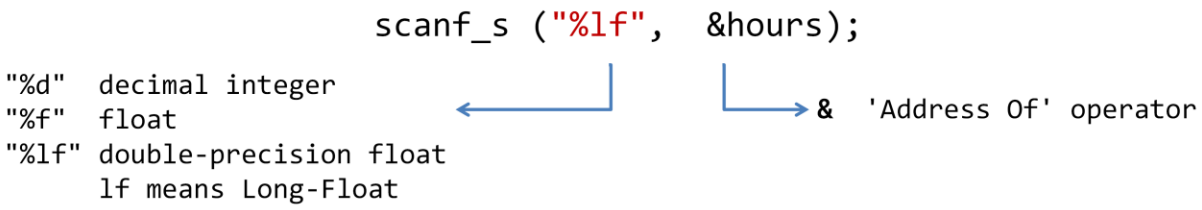
C++

```cpp
// INPUT: hours and payRate
cout << "Enter the hours worked: ";    // prompt
cin  >> hours;
```

```
Enter the hours worked: 41
Enter the pay rate: 20

Pay for regular hours  800.00
Pay for overtime        30.00
Total pay              830.00
```

When inputting data from the keyboard, it is very important to display a message on the screen so that the user knows to type something, and what should be typed. The message that asks for user input is called a **prompt.** When writing a prompt message, I like to end the prompt with a colon : and a space. The space after the colon is to make the screen look nice so that the user's typing won't be smashed right next to the prompt.

scanf Format String



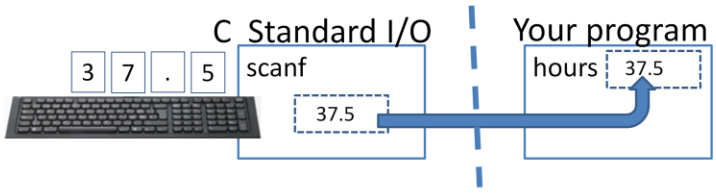
The first parameter in a call to scanf is the **format string**. It is surrounded by double-quotes. The format string tells scanf what type of data is expected to be typed at the console. The Paycheck program is using `"%lf"` in the format string because it is expecting data that may have digits past the decimal, and the input data is going to be stored in a variable named hours whose data type is **double**. Double is also known as long-float. When you are typing in the program, make sure that you type lower-case letters for %lf using a small-LF and not the number 1. Those number-1 and small-L look almost identical for most fonts.

scanf Format String

```
scanf_s ("%lf",  &hours);
```

```
"%d"  decimal integer
"%f"  float
"%lf" double-precision float
      lf means Long-Float
```

& 'Address-Of' operator



Each part of the program likes to protect its own area in memory and its own data. It is like there is a magic barrier between the scanf routine and your program that under normal circumstances can't be crossed. Since scanf needs to place the data it read from the keyboard into your part of the program, it is necessary for your program to provide the address of your variable to scanf.

For example, if the four characters 3 7 . 5 are entered at the keyboard, and the scanf control string is %lf, scanf will internally convert those four characters into a floating point number 37.5. Because your program is providing the memory address of its own variable named hours, scanf knows where to store the data in the memory location that belongs to your part of the program. As used here, the

ampersand & is called the **address-Of** operator. Just to make things confusing later, the ampersand & can have different meanings depending on where it is used.

C++ **cout** and **cin**

C

```c
/* INPUT: hours and payRate */
printf  ("Enter the hours worked: ");    /* prompt */
/* your compiler may need scanf instead of scanf_s */
scanf_s ("%lf", &hours);
```

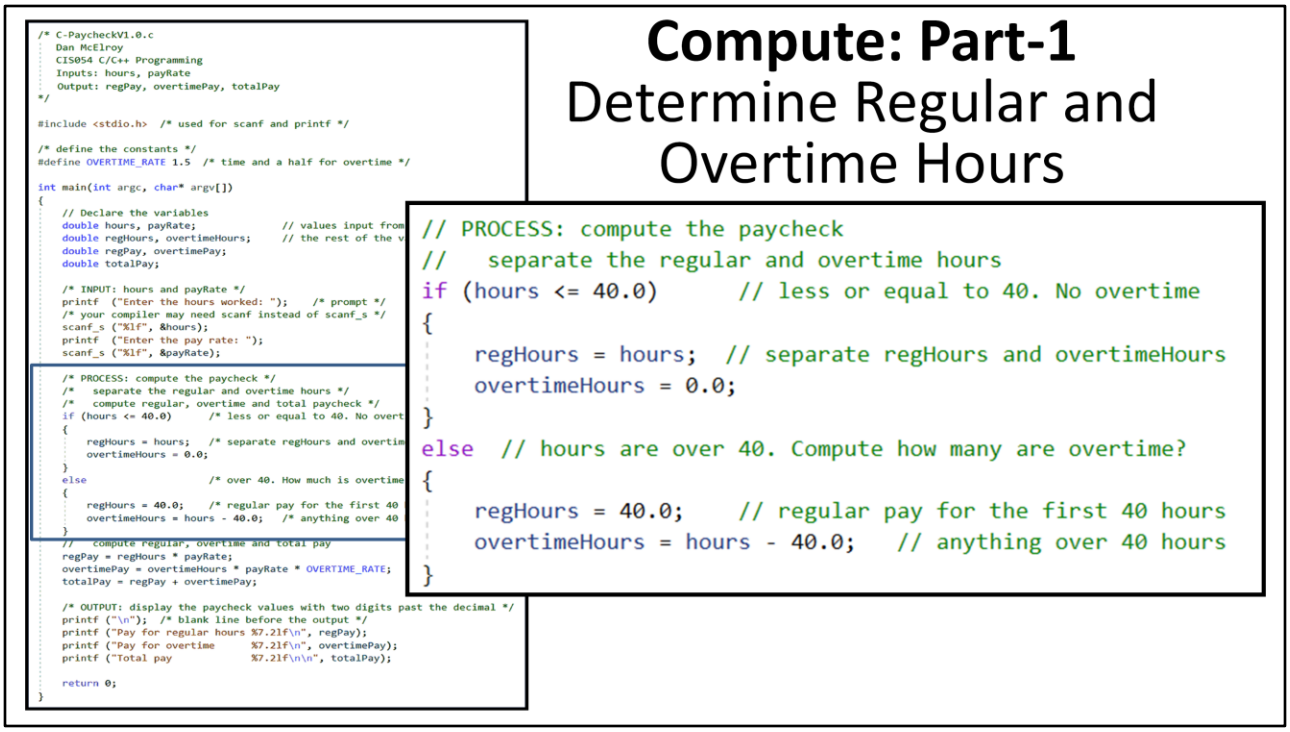
C++

```cpp
// INPUT: hours and payRate
cout << "Enter the hours worked: ";    // prompt
cin  >> hours;
```

With C++, **cout** uses the **insertion operator <<** and **cin** uses the **extraction operator >>.** They are referred by these names with the idea that cout is inserting data into the output stream and cin is extracting data from the input stream.

The easiest way to remember which ones to use is to think of them as arrows and look at the direction they are pointing. Data is going out to the console for cout.

Data is going from the keyboard when using cin and being placed into a variable.



We have finished the INPUT section of the program. Now it is time for the PROCESS part of the program. We can use the values that were input to compute the actual paycheck.

The first thing that needs to be done is to separate the overtime hours from the hours that were input at the keyboard.

The **if…else** statements are used to determine the number of regular hours worked and the number of overtime hours. The **if** statement has a logical expression enclosed within the parentheses **( )** that is testing to see if the hours are less than or equal to 40. The logical expression must evaluate to either a TRUE or a FALSE. When the expression **hours <= 40.0** evaluates TRUE, that means there is no

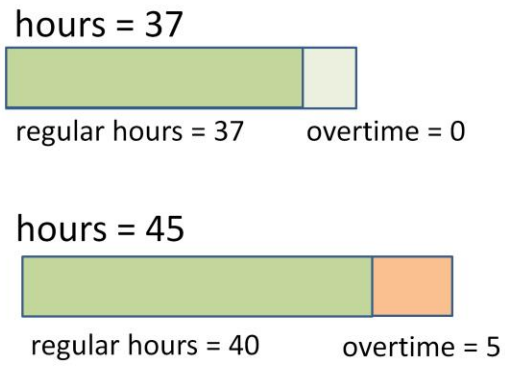
overtime and the block of code attached to the **if** statement is executed. When the expression evaluates is NOT-TRUE, the block of code attached to the **if** statement is skipped over and the code attached to the **else** statement is executed.

Be careful that you do NOT put a space in between the <= operator. I know that there are spaces in English when we say, "less than or equal", but the two characters <= form the comparison operator in C and C++.

Also be very careful that you do not put a semicolon at the end of the if statement or the end of the else statement.

Determine Regular and Overtime Hours

```
if (hours <= 40.0)
{
    regHours = hours;
    overtimeHours = 0.0;
}
else
{
    regHours = 40.0;
    overtimeHours = hours - 40.0;
}
```



There is no overtime if the logical expression evaluates to TRUE, so just set the regular hours to the number of hours worked, and the overtime to zero.

When the expression is NOT-TRUE, the block of code attached to the else statement is executed. Then there is overtime. The first 40 hours get paid at the regular rate, but anything over 40 gets paid at time and a half. Set the regular hours to 40 and the overtime hours to anything over 40.

Test the Program

Since this program can process a paycheck that does not have overtime (40 hours or less), and also a paycheck that has overtime (more than 40 hours), it is important to test both conditions.

It also important to test the program at 40 hours. For example, using these values:

Hours	payRate	grossPay
39	20	780.00
40	20	800.00
41	20	830.00

This program can process a paycheck that does not have overtime (40 hours or less), and also a paycheck that has overtime (more than 40 hours), it is important to test both conditions.

It also important to test the program at 40 hours. This way, we are testing the program for less than 40, equal to 40 and greater than 40. It is not necessary to test the program for every possible value that is less than 40 or every possible value greater than 40.

Compute: Part-2
Determine regPay overtimePay and totalPay

```
/* C-PaycheckV1.0.c
   Dan McElroy
   CIS054 C/C++ Programming
   Inputs: hours, payRate
   Output: regPay, overtimePay, totalPay
*/

#include <stdio.h>  /* used for scanf and printf */

/* define the constants */
#define OVERTIME_RATE 1.5  /* time and a half for overtime */

int main(int argc, char* argv[])
{
    // Declare the variables
    double hours, payRate;              // values input from the keyboard
    double regHours, overtimeHours;     // the rest of t
    double regPay, overtimePay;
    double totalPay;

    /* INPUT: hours and payRate */
    printf  ("Enter the hours worked: ");    /* prompt *
    /* your compiler may need scanf instead of scanf_s *
    scanf_s ("%lf", &hours);
    printf  ("Enter the pay rate: ");
    scanf_s ("%lf", &payRate);

    /* PROCESS: compute the paycheck */
    /*   separate the regular and overtime hours */
    /*   compute regular, overtime and total paycheck */
    if (hours <= 40.0)      /* less or equal to 40. No overtime */
    {
        regHours = hours;   /* separate regHours and overtimeHours */
        overtimeHours = 0.0;
    }
    else                    /* over 40. How much is overtime? */
    {
        regHours = 40.0;    /* regular pay for the first 40 hours */
        overtimeHours = hours - 40.0;   /* anything over 40 hours */
    }
    //   compute regular, overtime and total pay
    regPay = regHours * payRate;
    overtimePay = overtimeHours * payRate * OVERTIME_RATE;
    totalPay = regPay + overtimePay;

    /* OUTPUT: display the paycheck values with two digits past the decimal */
    printf ("\n");  /* blank line before the output */
    printf ("Pay for regular hours %7.2lf\n", regPay);
    printf ("Pay for overtime      %7.2lf\n", overtimePay);
    printf ("Total pay             %7.2lf\n\n", totalPay);

    return 0;
}
```

```
//   compute regular, overtime and total pay
regPay = regHours * payRate;
overtimePay = overtimeHours * payRate * OVERTIME_RATE;
totalPay = regPay + overtimePay;
```

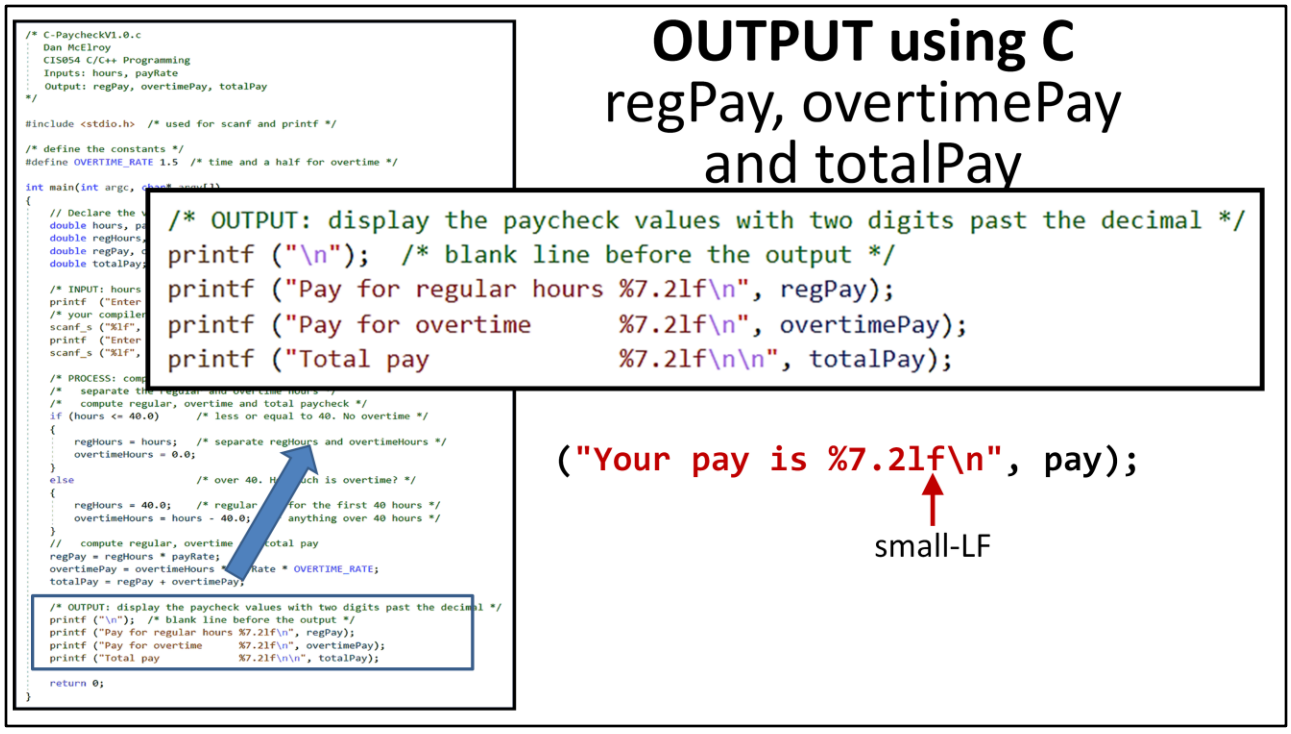
The computations for the **regPay** and **overtimePay** are fairly straight forward.

regPay = regHours * payRate

overtimePay = overtimeHours * payRate * OVERTIME_RATE;

The overtime rate is from the constant 1.5 at the top of the program.

The **totalPay** is the sum of **regPay** and **overtimePay**.



We have finished the COMPUTE part of the program. Now it is time to output the results.

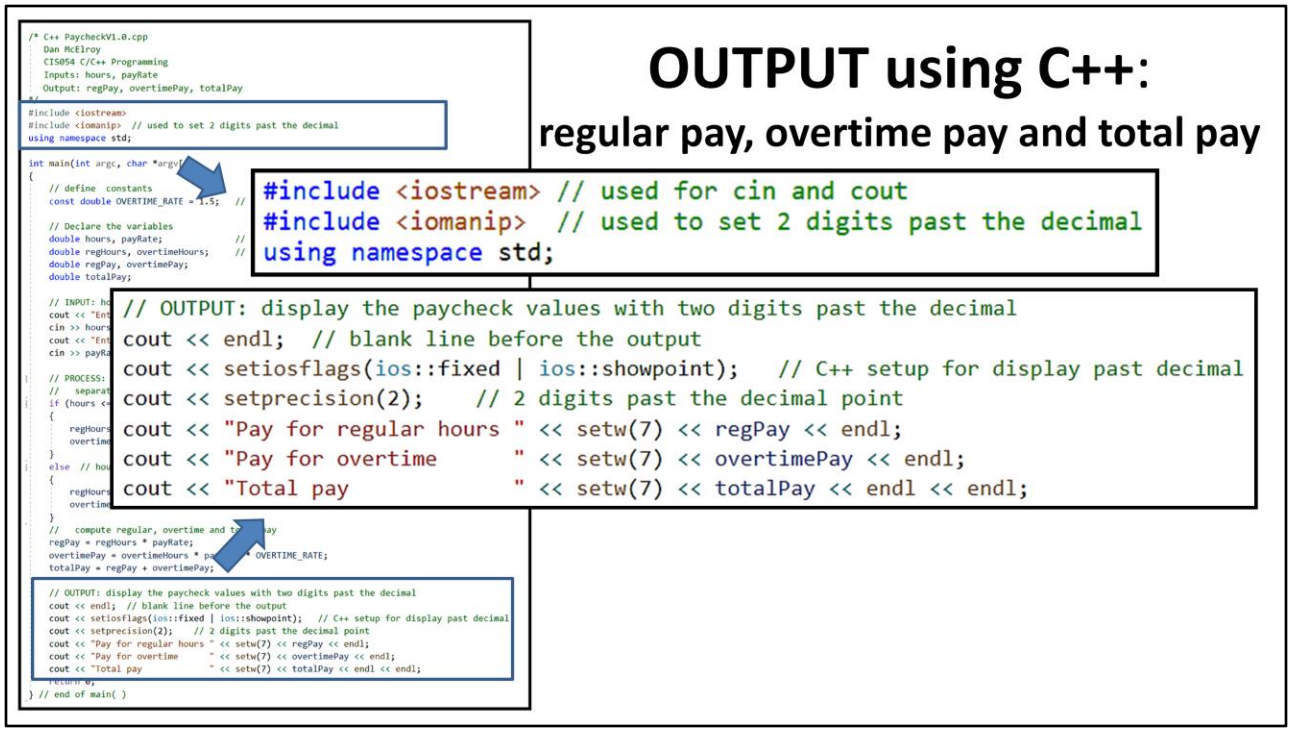
When using C, the first part of the format string in printf is just text that will be displayed "Your pay is " followed by the format specifier "%7.2lf" and the "\n" which moves the cursor down to the next line.

The **%7.2lf** tells printf to replace the **%7.2lf** with the data that is after the closing quotes of the format string. Commas separate the format string and each piece of data that is after the format string. The **lf** identifies that the data is expected to be of type **double**, and the **7.2** says a minimum of 7 characters on the screen including the decimal point and two digits past the decimal. If **pay** were computed at 803.0, then the output on the screen would show

Your pay is 803.00

and the cursor would move down one line because of the \n.

Remember how printf displays two digits past the decimal. You will need this in future lab assignments for any programs written in C.



If you thought that printf was a little complicated in displaying two digits past the decimal, look at how C++ does it. We need the **#include <iomanip>** statement at the top of the file. We also need **cout << setiosflags(ios::fixed | ios::showpoint);**
Now all we have to do is **cout << setprecision(2)** before displaying a double data type.

The cout statement uses the << insertion operator to string several pieces of the message together when forming the console output.
cout << "Pay for regular hours " << setw(7) << regPay << endl;
has four parts. The first is just a string of text characters that are displayed exactly as shown within the double-quotes. The setw(7) sets a field width to use 7 character positions on the screen for the piece of data that follows. Then we have the data that is to be displayed. And finally **endl** to cause the cursor to move to the next

line.

Make sure you type the last letter of **endl** as a lower-case L. **endl** end-of-line which moves the cursor down to the next line on the screen.

Using **endl** twice moves the cursor down two lines which gives you a blank line on the screen.

```
/* C++ PaycheckV1.0.c : Defines the entry point for the console application.
   Dan McElroy
   CIS054 C/C++ Programming
   Inputs: hours, payRate
   Output: grossPay, taxes, netPay
*/

#include <iostream> // used for cin and cout
#include <iomanip>  // used to set 2 digits past the decimal
using namespace std;

// define the constants
const double OVERTIME_RATE = 1.5;   // time and a half for overtime
const double TAX_RATE = 0.17;       // 0.17 is 17%

int main(int argc, char* argv[])
{
    // Declare the variables
    double hours, payRate;
    double regHours, overtimeHours;
    double regPay, overtimePay;
    double grossPay, taxes, netPay;

    // INPUT: hours and payRate
    cout << "Enter the hours worked: ";    // prompt
    cin  >> hours;
    cout << "Enter the pay rate: ";
    cin  >> payRate;

    // PROCESS: compute the paycheck
    //   separate the regular and overtime hours
    //   compute regular, overtime and total paycheck
    if (hours <= 40.0)       // less or equal to 40. No overtime
    {
        regHours = hours;   // separate regHours and overtimeHours
        overtimeHours = 0.0;
    }
    else                    // over 40. How much is overtime?
    {
        regHours = 40.0;    // regular pay for the first 40 hours
        overtimeHours = hours - 40.0;   // anything over 40 hours
    }
    regPay = regHours * payRate;
    overtimePay = overtimeHours * payRate * OVERTIME_RATE;
    grossPay = regPay + overtimePay;
    taxes = grossPay * TAX_RATE;
    netPay = grossPay - taxes;

    // OUTPUT: display the paycheck values with two digits past the decimal
    cout << endl;  // blank line before the output
    cout << setiosflags(ios::fixed | ios::showpoint);
    cout << "Your gross pay is $" << setprecision(2) << grossPay  << endl;
    cout << "Your taxes are $"    << setprecision(2) << taxes  << endl;
    cout << "Your net pay is $" << setprecision(2) << netPay  << endl << endl;

     return 0;
}
```

system ("PAUSE");

On some systems, you may need to type the following line before the return 0; statement to keep the output window from disappearing.

This does not need to be done for Xcode, NetBeans or even Visual Studio if the program is started without debugging.

system ("PAUSE");

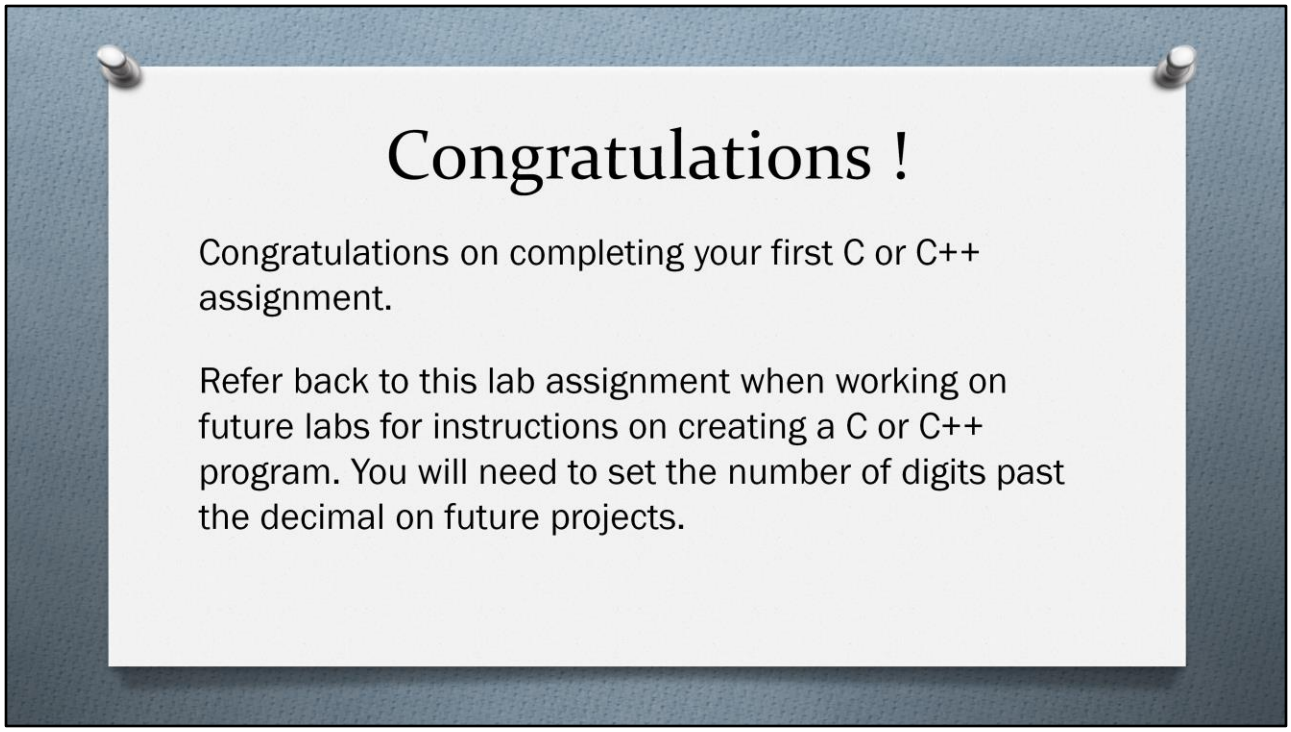
Depending on which development system you are using, you may need to insert the line **system("PAUSE");** right before the **return 0** to prevent the output window from closing immediately after the program ends and before you even have the opportunity to look at the output. This will not be necessary if you are using Xcode or NetBeans. It also won't be necessary if you use the **Ctrl+F5** to start a Microsoft Visual Studio C++ program without debugging.

What Is This Program Missing

- Check for negative hours, or pay rate
- Check for hours > 168
- Check for excessive pay rate
- Check for illegal inputs

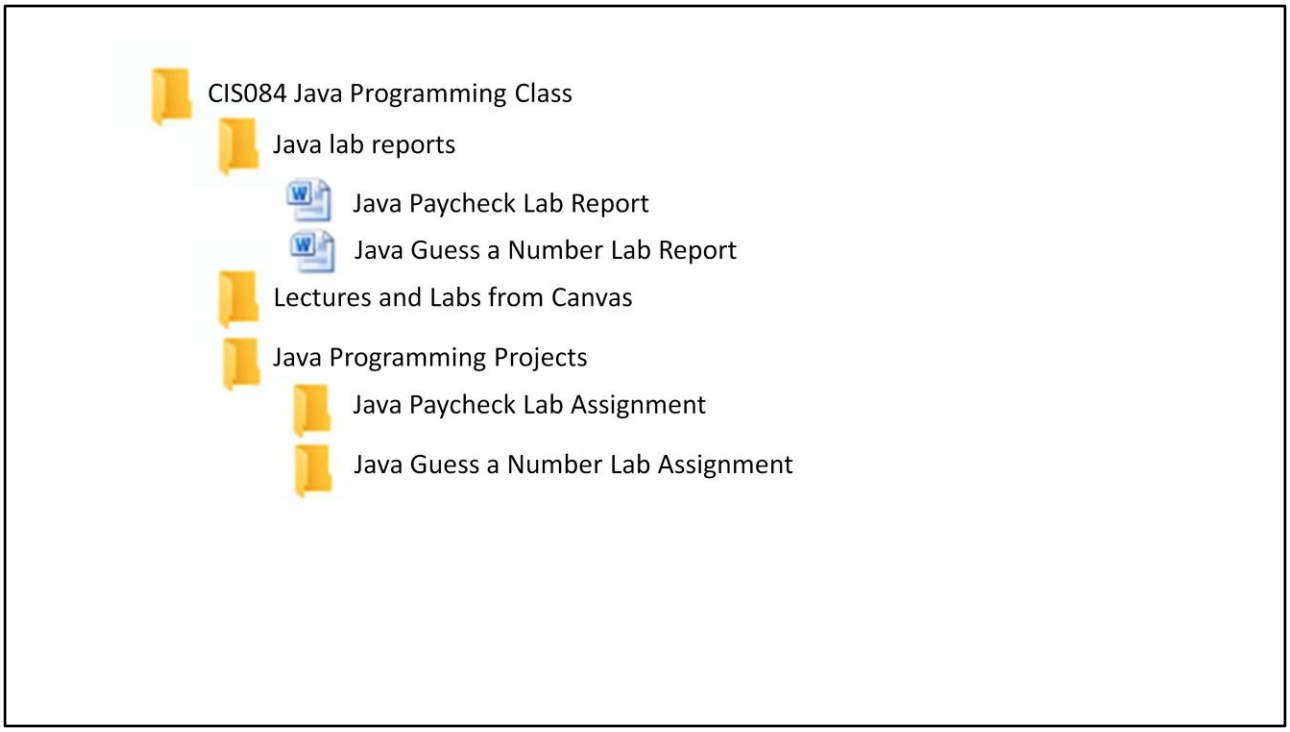
This may seem like a big task for you first program, but a lot of code is missing to make a robust program that is more user friendly and able to detect and handle bad inputs. Here are some of the things that are missing that we will need to learn how to process in later projects:

Check for negative hours or pay rate.
Check for hours > 168. There are only 168 possible hours in a week.
Check for excessive hours or pay rate.
Check for illegal inputs.



Congratulations on completing your first C or C++ assignment. I hope that you were not only able to complete just the lab report and submit it, but also that you gained an understanding of how this program works.

Refer back to this lab assignment when working on future labs for instructions on creating a C or C++ program.


CIS084 Java Programming Class
Java lab reports
Java Paycheck Lab Report
Java Guess a Number Lab Report
Lectures and Labs from Canvas
Java Programming Projects
Java Paycheck Lab Assignment
Java Guess a Number Lab Assignment