CSCI 1300 CS1: Starting Computing
Instructor: Fleming/Wong, Spring 2019
Homework 4
Due Saturday, February 9th, by 6 pm
+5% bonus if submitted by Friday, February 8th, 11:59 pm

All 4 components (Cloud9 workspace, Moodle CodeRunner attempts, multiple choice questions, and zip file) must be completed and submitted by Saturday, February 9th, 6:00 pm for your homework to receive points.

---

1. Objectives

- Understand and work with if-else conditionals and switch statements.
- Writing and testing C++ functions
    - Understand problem description
    - Design your function:
        - come up with a step by step algorithm,
        - convert the algorithm to pseudocode
        - imagine many possible scenarios and corresponding sample runs or outputs
    - Convert the pseudocode into a program written in the C++ programming language
    - Test it in the Cloud9 IDE and submit it for grading on Moodle

---

2. Background

Conditional Statements

Conditional statements in computer science are features of a programming language which execute different computations or actions based on the outcome of a *boolean condition*. Boolean conditions are expressions that evaluate a relationship between two values and then return either True or False. Conditionals are mostly used in order to alter the *control flow*, or the order in which individual instructions are carried out, of a program.

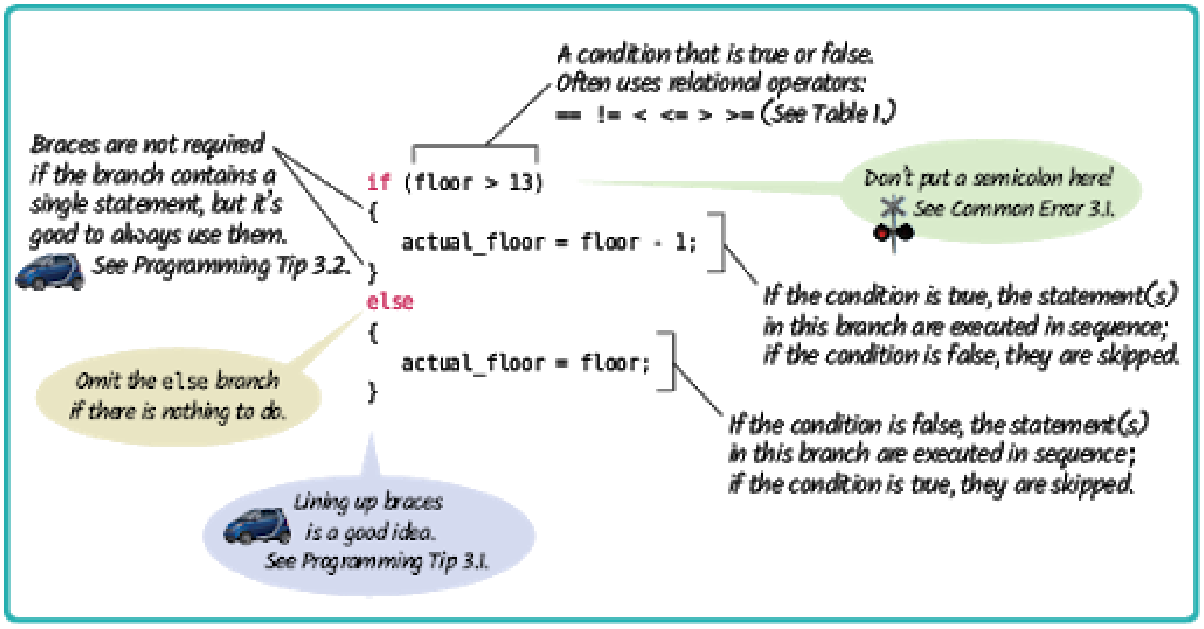
Largely, conditional statements fall into two types:

1. If-else statements, which execute one set of statements or another based on the value of a given condition
2. Switch statements, which are used for exact value matching and allow for multi-way branching

IF-ELSE statements:

The *if statement* selects and executes the statement(s) based on a given condition. If the condition evaluates to True then a given set of statement(s) is executed. However, if the condition evaluates to False, then the given set of statements is skipped and the program control passes to the statement following the if statement. If we have another *else statement* next to this, it is handled by the else block.

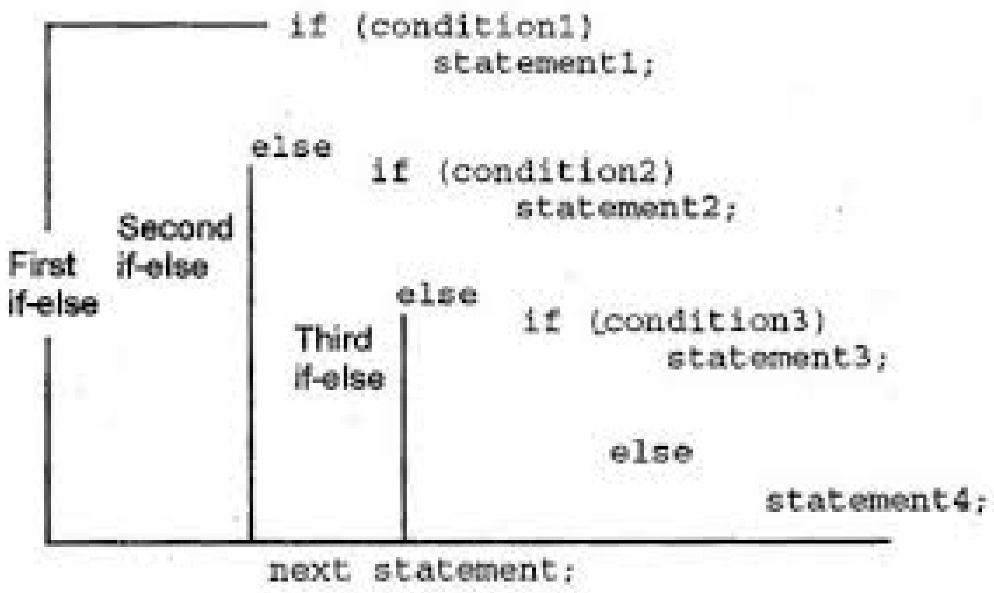
The general syntax of the if-else statement is as follows:


A condition that is true or false.
Often uses relational operators:
== != < <= > >= (See Table 1.)
Braces are not required if the branch contains a single statement, but it's good to always use them. See Programming Tip 3.2.
Don't put a semicolon here! See Common Error 3.1.
If the condition is true, the statement(s) in this branch are executed in sequence; if the condition is false, they are skipped.
Omit the else branch if there is nothing to do.
If the condition is false, the statement(s) in this branch are executed in sequence; if the condition is true, they are skipped.
Lining up braces is a good idea. See Programming Tip 3.1.

```
if (floor > 13)
{
   actual_floor = floor - 1;
}
else
{
   actual_floor = floor;
}
```

Nested IF-ELSE:

If there are more than 2 branches within your program, we can use *nested if statement*s to model this behavior within a program by putting one if statement within another. Below is an example of the syntax of a nested if statement.

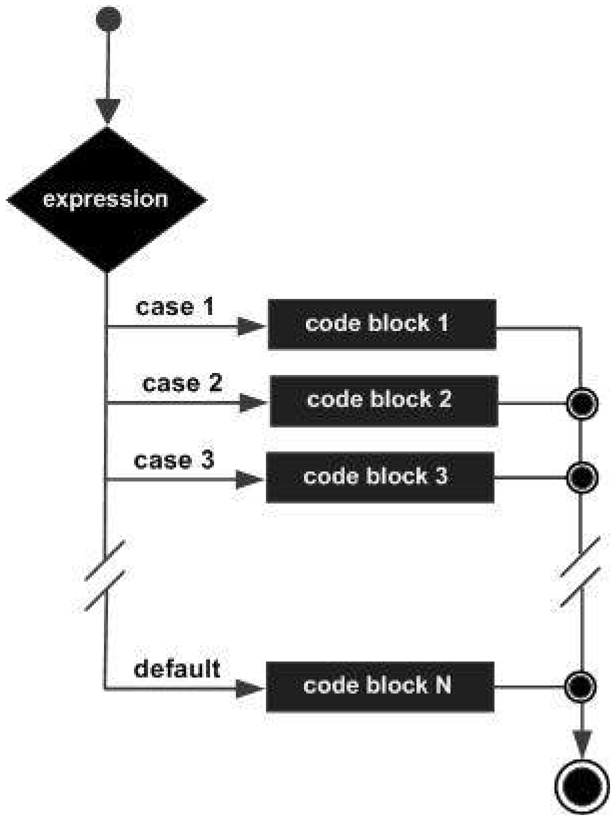


```
if (condition1)
        statement1;

else    if (condition2)
                statement2;

        else    if (condition3)
                        statement3;

                else
                        statement4;
next statement;
```

SWITCH statements:

Switch case statements are a substitute for long if statements that compare a variable to several integral values.

- The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression
- Switch is a control statement that allows a value to change control of execution.



Syntax:

With the switch statement, the variable name is used once in the opening line. A case keyword is used to provide the possible values of the variable, which is followed by a colon and a set of statements to run if the variable is equal to a corresponding value.

```
switch (n){
     case 1:
          // code to be executed if n = 1;
          break;
     case 2:
          // code to be executed if n = 2;
          break;
     default:
          // code to be executed if n doesn’t match any cases
}
```

Important notes to keep in mind while using switch statements :

1. The expression provided in the switch should result in a constant value otherwise it would not be valid.
    a. switch(num) //allowed (num is an integer variable)
    b. switch(‘a’) //allowed (takes the ASCII Value)
    c. switch(a+b) //allowed,where a and b are int variable, which are defined earlier
2. Duplicate case values are not allowed.
3. The **break** statement is used inside the switch to terminate a statement sequence. When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
4. The **default** statement is optional. Even if the switch case statement do not have a default statement,it would run without any problem.
5. Nesting of switch statements are allowed, which means you can have switch statements inside another switch. However nested switch statements should be avoided as it makes program more complex and less readable.
6. The break statement is optional. If omitted, execution will continue on into the next case. The flow of control will fall through to subsequent cases until a break is reached.

Relational Operators:

A *relational operator* is a feature of a programming language that tests or defines some kind of relation between two entities. These include numerical equality (e.g., 5 = 5) and inequalities (e.g., 4 ≥ 3). Relational operators will evaluate to either True or False based on whether the relation between the two operands holds or not. When two variables or values are compared using a relational operator, the resulting expression is an example of a *boolean condition* that can be used to create branches in the execution of the program. Below is a table with each relational operator's C++ symbol, definition, and an example of its execution.

>	greater than	5 > 4 is TRUE
<	less than	4 < 5 is TRUE
>=	greater than or equal	4 >= 4 is TRUE
<=	less than or equal	3 <= 4 is TRUE
==	equal to	5 == 5 is TRUE
!=	not equal to	5 != 4 is TRUE

Logical Operators

Logical operators are symbols that are used to compare the results of two or more conditional statements, allowing you to combine relational operators to create more complex comparisons. Similar to relational operators, logical operators will evaluate to True or False based on whether the given rule holds for the operands. Below are some examples of logical operators and their definitions.

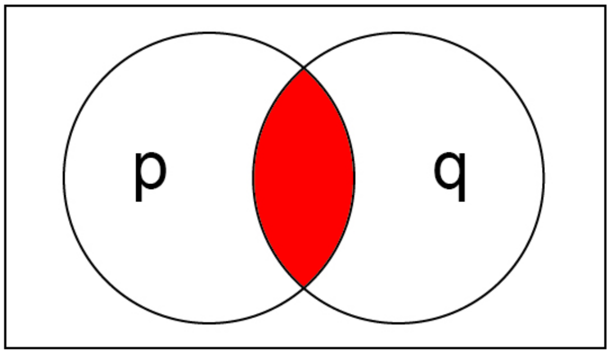
- **`&&`** (AND) returns true if and only if both operands are true
- **`||`** (OR) returns true if one or both operands are true
- **`!`** (NOT) returns true if operand is false and false if operand is true

Truth Tables

Every logical operator will have a corresponding *truth table*, which specifies the output that will be produced by that operator on any given set of valid inputs. Below are examples of truth tables for each of the logical operators specified above.

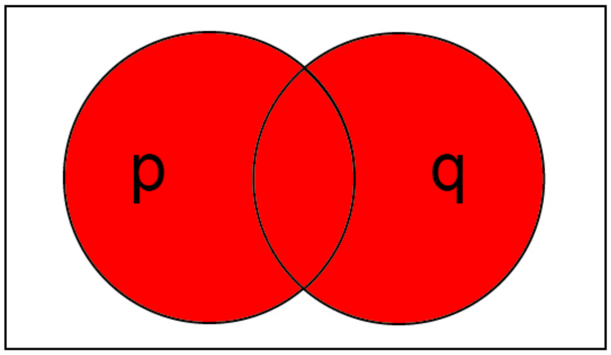
AND:
These operators return true if and only if both operands are True. This can be visualized as a venn diagram where the circles are overlapping.

`p`	`q`	`p && q`
True	True	True
True	False	False
False	True	False
False	False	False



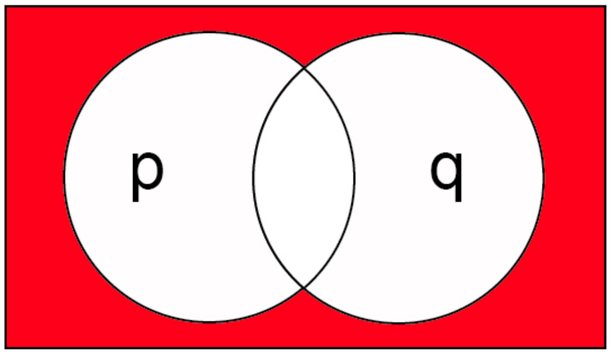
OR:
These operators return True if one or both of the operands are True. This can be visualized as the region of a venn diagram encapsulated by both circles.

`p`	`q`	`p || q`
True	True	True
True	False	True
False	True	True
False	False	False



NOR:
These operators return True if both of the operands are False. This can be visualized as the region of a venn diagram that is not within either of the circles.

`p`	`q`	`!(p || q)`
True	True	False
True	False	False
False	True	False
False	False	True



3. Submission Requirements

All three steps must be fully completed by the submission deadline for your homework to be graded.

1. ***Share your Cloud 9 workspace with your TA(if you have not done yet):*** Your recitation TA will review your code by going to your Cloud9 workspace. *TAs will check the last version that was saved before the submission deadline*.
    - Create a directory called **Hmwk4** and place all your file(s) for this assignment in this directory.
    - Share your workspace by clicking Share in the upper right hand corner and inviting your TA using their Cloud9 username.
    - Make sure to *save* the final version of your code (File > Save). Verify that this version displays correctly by going to File > File Version History.
    - The file(s) should have all of your functions, test cases for the functions in `main()` function(s), and adhere to the style guide. Please read the **Test Cases** and **Style and Comments** sections for more details.

2. ***Submit to the Moodle CodeRunner:*** Head over to Moodle to the link **Homework 4 CodeRunner**. You will find one programming quiz question for each problem in the assignment. Submit your solution for the first problem and press the Check button. You will see a report on how your solution passed the tests, and the resulting score for the first problem. You can modify your code and re-submit (press *Check* again) as many times as you need to, up until the assignment due date. Continue with the rest of the problems.

3. ***Complete Multiple Choice Questions:*** There are 4 Multiple Choice Questions (MCQs) which refer to problem 5, `countDigits`. You can find **Homework 4 MCQ Quiz** on Moodle. You can attempt at most 2 times, and you have 20 minutes for each attempt to answer 4 questions. Your highest grade will be count toward your Homework 4 grade.

4. ***Submit a .zip file to Moodle:*** After you have completed all 7 questions from the Moodle assignment, zip all 7 solution files you compiled in Cloud9 (one cpp file for each problem), and submit the zip file through the **Homework 4 (File Submission)** link on Moodle.

TA Name	Cloud 9 username	TA Name	Cloud 9 username
Shipra Behera	beherashipra	**Telly Umada**	TetsumichiUmada
Josh Ladd	joshladd	**Ashwin Sankaralingam**	ashwinsankaralingam
Chu-Sheng Ku	chusheng	**Supriya Naidu**	supriyanaidu
Punith Patil	variable314	**Sebastian Laudenschlager**	slaudens
Karthik Palavalli	karthikpalavalli	**Dhanendra Soni**	dhanendra
Thanika Reddy	thanika	**Shudong Hao**	shudonghao

---

4. Rubric

Aside from the points received from the **Homework 4 CodeRunner** quiz problems, your TA will look at your solution files (zipped together) as submitted through the **Homework 4 (File Submission)** link on Moodle and assign points for the following:

Comments **(5 points):**

- Your code should be well-commented. Use comments to explain what you are doing, especially if you have a complex section of code. These comments are intended to help other developers understand how your code works. These comments should begin with two backslashes (//) or the multi-line comments (/* … *comments here…* */) .
- Please also include a comment at the top of your solution with the following format:

```
// CS1300 Spring 2019
// Author: my name
// Recitation: 123 – Favorite TA
// Cloud9 Workspace Editor Link: https://ide.c9.io/…
// Homework 4 - Problem # ...
```

Algorithm (5 points):

- Before each function that you define, you should include a comment that describes the inputs and outputs of your function and what algorithms you are using inside the function.
- This is an example C++ solution. Look at the code and the algorithm description for an example of what is expected.

Example 1:

```
/*
 * Algorithm: convert money from U.S. Dollars (USD) to Euros.
 *    1. Take the value of number of dollars involved in the transaction.
 *    2. Current value of 1 USD is equal to 0.86 euros
 *    3. Multiply the number of dollars got with the currency
 *       exchange rate to get Euros value
 *    4. Return the computed Euro value
 * Input parameters: Amount in USD (double)
 * Output (prints to screen): nothing
 * Returns: Amount in Euros (double)
 */
```

Example 2:

```
double convertUSDtoEuros(double dollars)
{
      double exchange_rate = 0.86; //declaration of exchange rate
      double euros = dollars * exchange_rate; //conversion
      return euros; //return the value in euros
}
```

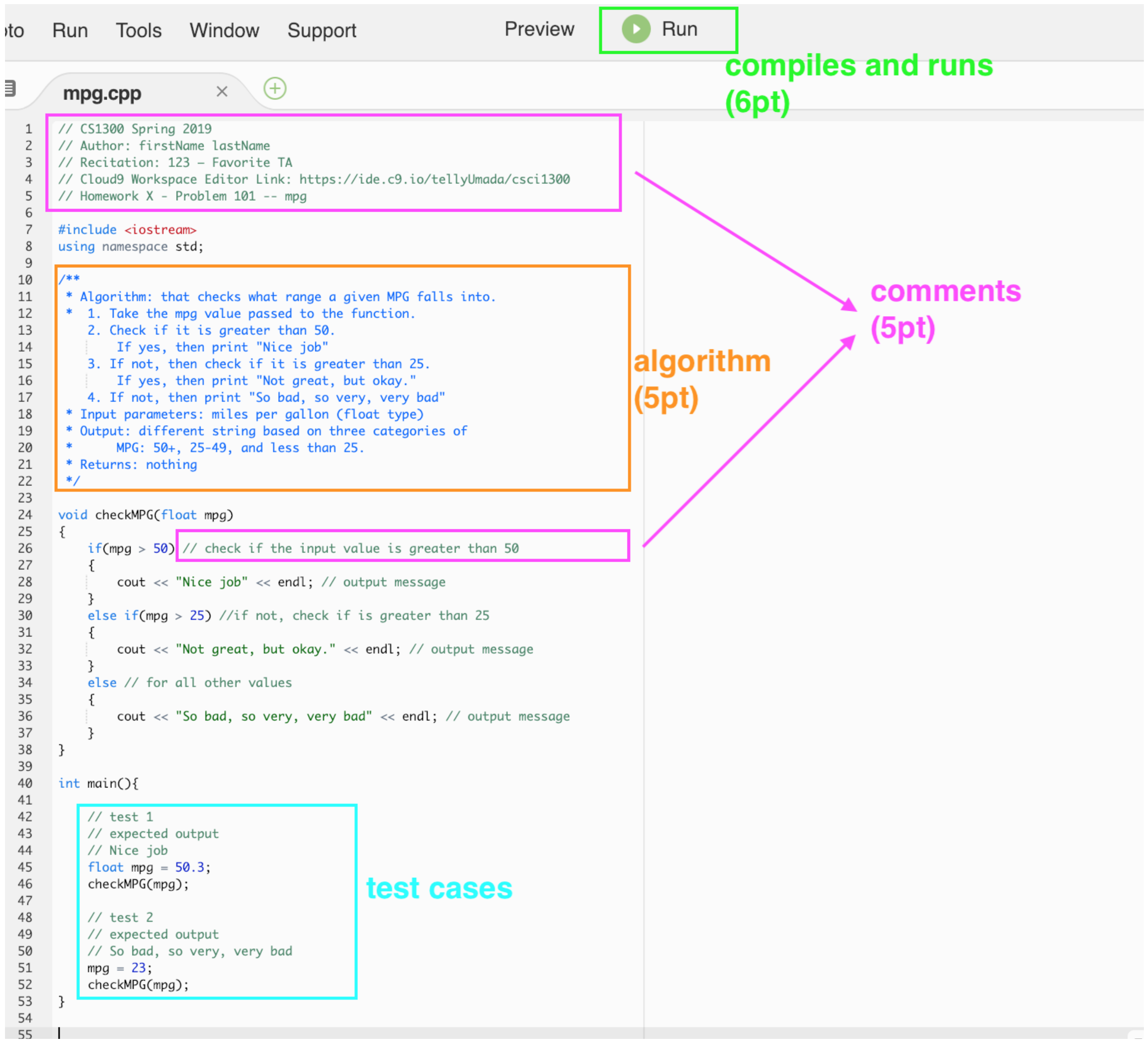
The algorithm described below does not mention in detail what the algorithm does and does not mention what value the function returns. Also, the solution is not commented. This would work properly, but would not receive full credit due to the lack of documentation.

```
/*
 * conversion
 */
double convertUSDtoEuros(double dollars)
{
      double euros = dollars * 0.86;
      return euros;
}
```

***Test Cases* (20 points):**

1. *Code compiles and runs* (**6 points**):
    - The zip file you submit to Moodle should contain **7** full programs (each with a `main()` function), saved as .cpp files. It is important that your programs can be compiled and run on Cloud9 with no errors. The functions included in these programs should match those submitted to the CodeRunner on Moodle.

2. *Test cases* (**14 points**):
   For this week's homework, all 7 problems are asking you to create a function. In your Cloud9 solution file for each function, you should have 2 test cases present in their respective `main()` function, for a total of 14 test cases (see the diagram on the next page). Your test cases should follow the guidelines, **Writing Test Cases**, posted on Moodle under Week 3.



```cpp
// CS1300 Spring 2019
// Author: firstName lastName
// Recitation: 123 – Favorite TA
// Cloud9 Workspace Editor Link: https://ide.c9.io/tellyUmada/csci1300
// Homework X - Problem 101 -- mpg

#include <iostream>
using namespace std;

/**
 * Algorithm: that checks what range a given MPG falls into.
 *  1. Take the mpg value passed to the function.
    2. Check if it is greater than 50.
        If yes, then print "Nice job"
    3. If not, then check if it is greater than 25.
        If yes, then print "Not great, but okay."
    4. If not, then print "So bad, so very, very bad"
 * Input parameters: miles per gallon (float type)
 * Output: different string based on three categories of
 *      MPG: 50+, 25-49, and less than 25.
 * Returns: nothing
 */

void checkMPG(float mpg)
{
    if(mpg > 50) // check if the input value is greater than 50
    {
        cout << "Nice job" << endl; // output message
    }
    else if(mpg > 25) //if not, check if is greater than 25
    {
        cout << "Not great, but okay." << endl; // output message
    }
    else // for all other values
    {
        cout << "So bad, so very, very bad" << endl; // output message
    }
}

int main(){

    // test 1
    // expected output
    // Nice job
    float mpg = 50.3;
    checkMPG(mpg);

    // test 2
    // expected output
    // So bad, so very, very bad
    mpg = 23;
    checkMPG(mpg);
}

```

5. Problem Set

Note: To stay on track for the week, we recommend to finish/make considerable progress on problems 1-3 by Wednesday. Students with recitation on Thursday are encouraged to come to recitation with questions and have made a start on all of the problems.

Problem 1 (5 points): numberSign

Write a function named **numberSign** that takes one integer parameter, returns nothing, and prints to the screen whether that integer is "`negative`", "`positive`", or "`zero`".

- Your function **MUST** be named **numberSign**
- Your function takes one input argument: an **integer** number
- Your function should not return any value
- Your function prints/outputs to the console window a message, as specified above

Examples:

- When the argument is equal to **`4`**, the function should print **`positive`**;
- When the argument is equal to **`-4`**, the function should print **`negative`**;
- When the argument is equal to **`0`**, the function should print **`zero`**.

In Cloud9 the file should be called **numberSign.cpp** and it will be one of 7 files you need to zip together for the **Homework 4 (File Submission)** on Moodle.

Don't forget to head over to Moodle to the link **Homework 4 CodeRunner**. For Problem 1, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 2 (5 points): collatzStep

Write a function **collatzStep** which takes a single integer argument and returns an integer. The return value should be the next value in the Collatz sequence based on the value of the input parameter. If the given value `n` is even, the next value should be `n/2` . If n is odd, the next value should be `3n+1` . If the given value is not positive the function should return `0`.

- Your function **MUST** be named **collatzStep**
- Your function takes one input argument: an **integer** number
- Your function must return an **integer**, as specified above

Examples:

- When the argument is equal to `4`, the function should return `2`;
- When the argument is equal to `7`, the function should return `22`;
- When the argument is equal to `-5`, the function should return `0`.

In Cloud9 the file should be called **collatzStep.cpp** and it will be one of 7 files you need to zip together for the **Homework 4 (File Submission)** on Moodle.

Don't forget to head over to Moodle to the link **Homework 4 CodeRunner**. For Problem 2, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 3 (10 points): checkEqual

Write a function **checkEqual** which takes three numbers as parameters, and prints

- "All same", if they are all the same
- "All different", if they are all different
- "Neither", otherwise

- Your function **MUST** be named **checkEqual**
- Your function takes three input arguments: all **integers**
- Your function **must print** one of the above three statements
- Your function should **NOT** return anything

Examples:

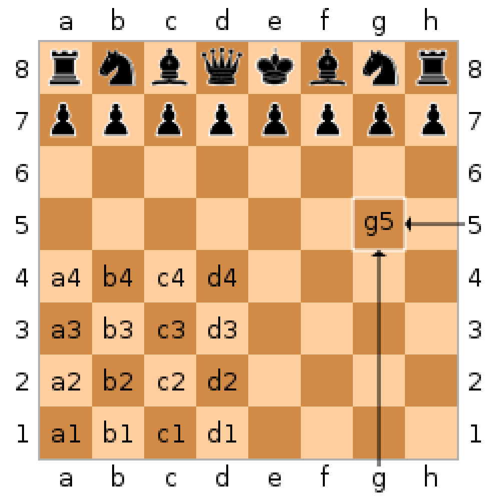
- If the input arguments are `(1, 2, 3)`, the function should print `All different`
- If the input arguments are `(2, 2, 2)`, the function should print `All same`
- If the input arguments are `(1, 1, 2)`, the function should print `Neither`

In Cloud9 the file should be called **checkEqual.cpp** and it will be one of 7 files you need to zip together for the **Homework 4 (File Submission)** on Moodle.

Don't forget to head over to Moodle to the link **Homework 4 CodeRunner**. For Problem 3, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 4 (10 points): chessBoard

Each square on a chess board can be described by a letter and number, such as g5 in this example.



Write a function **chessBoard** which determines if a square with a given letter or number on a chessboard, is dark (black) or light (white).

- Your function **MUST** be named **chessBoard**
- Your function takes two input arguments: a **character, and** an **integer**
- If your function receives as input a valid character and number, then it must print either "black" or "white", depending on the color of the square given by those coordinates. *(Note: the darker squares on the example board above should be considered as black and the lighter squares should be considered as white; non-white/black is used to distinguish between text, pieces and squares.)*
- If your function receives as input anything other than a-h for character, and 1-8 for number, then it must print "Chessboard squares can only have letters between a-h and numbers between 1-8." Note that your function should discern between the valid lowercase letters and the invalid uppercase ones.
- Your function should **NOT** return anything.

Examples:

- If the input arguments are `(‘g’, 5)`, the function should print `black`
- If the input arguments are `(‘c’, 4)`, the function should print `white`
- If the input arguments are `(‘a’, 1)`, the function should print `black`
- If the input arguments are `(‘A’, 10)`, the function should print `Chessboard squares can only have letters between a-h and numbers between 1-8`

In Cloud9 the file should be called **chessBoard.cpp** and it will be one of 7 files you need to zip together for the **Homework 4 (File Submission)**on Moodle.

Don’t forget to head over to Moodle to the link **Homework 4 CodeRunner**. For Problem 4, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 5 (10 points): countDigits

Write a function **countDigits** that takes integer as input and returns how many digits the number has. Assume that all integers are less than 1000 and greater than -1000. Suggestion: if the number is negative, you could first multiply it with –1.

- Your function **MUST** be named **countDigits**
- Your function takes one input argument: an **integer** number
- Your function must return the number of digits as an **integer**

Examples:

- If the argument is equal to `123`, the function should return `3`;
- If the argument is equal to `-75`, the function should return `2`.

In Cloud9 the file should be called **countDigits.cpp** and it will be one of 7 files you need to zip together for the **Homework 4 (File Submission)** on Moodle.

Don’t forget to head over to Moodle to the link **Homework 4 CodeRunner**. For Problem 5, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Note also that there are 4 Multiple Choice Questions (MCQs) which refer to this problem. You can find **Homework 4 MCQ Quiz** on Moodle.

Problem 6 (15 points): countHours

Write a function **countHours** that takes in a month and returns the number of hours present in the month.

- Your function **MUST** be named **countHours.**
- Your function takes one input argument: month as integer. For example, January is 1, February is 2 and so on, up to December is 12.
- Your function should not print anything.
- Your function should return the number of hours: an **integer** value**.**
- Use `switch` to solve this problem. `if-else` is not available.

Note*:* For this problem we assume all inputs fall in the category of a non leap year, i.e., February has only 28 days.

Examples:

- When the argument is equal to **`1`**, the function should return **`744`**;
- When the argument is equal to **`2`**, the function should return **`672`**;
- When the argument is equal to **`4`**, the function should return **`720`**.

In Cloud9 the file should be called **countHours.cpp** and it will be one of 7 files you need to zip together for the **Homework 4 (File Submission)** on Moodle.

Don't forget to head over to Moodle to the link **Homework 4 CodeRunner**. For Problem 6, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 7 (15 points): checkLeapYear

Write a function called **checkLeapYear** that takes a single integer argument representing a year and returns **true** if it is a leap year and **false** otherwise. This function should use a single if statement and Boolean operators.

- Your function **MUST** be named **checkLeapYear**
- Your function takes one input argument: an **integer** number
- Your function only uses `if`. `else` cannot be used to solve this problem.
- Your function should return a *boolean* value, **true** or **false**, according to the leap year definition (see below).

Here is an important question: What *is* a leap year? In general, years divisible by 4 are leap years. For dates after 1582, however, there is a *Gregorian correction*: years that are divisible by 100 are not leap years but years divisible by 400 are. So, for instance, 1900 was not a leap year but 2000 was.

Examples:

- When the argument is equal to `1900`, the function should return `false`;
- When the argument is equal to `2000`, the function should return `true`.

Note: when you print a boolean variable with the value **true**, it will display as **1**, while **false** will display as **0**.

In Cloud9 the file should be called **checkLeapYear.cpp** and it will be one of 7 files you need to zip together for the **Homework 4 (File Submission)** on Moodle.

Don't forget to head over to Moodle to the link **Homework 4 CodeRunner**. For Problem 7, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Extra Credit Problem (20 points): integerToRoman

One of the manys ways to represent a number is by using the Roman Numerals system. The system goes as follows

1. Has seven digits

Roman Digit	Integer Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

2. Numbers are formatted accordingly:
    a. Only numbers up to 3,999 are represented.
    b. As in the decimal system, the thousands, hundreds, tens, and ones are expressed separately.
    c. The numbers 1 to 9 are expressed as:

I	1
II	2
III	3
IV	4
V	5
VI	6
VII	7
VIII	8
IX	9

Note that 4 is represented as IV and 9 is represented as IX.

- Your function **MUST** be named **integerToRoman**
- Your function takes one input argument: an **integer** number
- Your function should **print**(`cout`) the Roman numeral, with appropriate capitalization
- Your function should ***not*** return any value

Examples:
- If the input argument is **`104`**, the function should print **`CIV`**
- If the input argument is **`490`**, the function should print **`CDXC`**
- If the input argument is **`3999`**, the function should print **`MMMCMXCIX`**

Your mission, should you choose to accept it, is to write a function **integerToRoman** which takes an integer argument and print its Roman equivalent.

In Cloud9 the file should be called **integerToRoman.cpp** and submit it to the **Homework 4 -- Extra Credit Problem (File Submission)** on Moodle (submit cpp file, not a zip file).

Don't forget to head over to Moodle to the link Homework 4 CodeRunner Quiz -- Extra Credit Problem. For this extra credit problem, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

6. Homework 4 checklist

Here is a checklist for submitting the assignment:

1. Complete the code Homework 4 CodeRunner
2. Complete Homework 4 MCQ Quiz
3. Submit one zip file to Homework 4 (File Submission). The zip file should be named, **<firstName>_<lastName>_homework4.zip**. It should have following 7 files:
    - **numberSign.cpp**
    - **collatzStep.cpp**
    - **checkEqual.cpp**
    - **chessBoard.cpp**
    - **countDigits.cpp**
    - **countHours.cpp**
    - **checkLeapYear.cpp**
4. If you work on the extra credit problem (integerToRoman), submit your solution to Homework 4 -- Extra Credit Problem (File Submission) and **integerToRoman.cpp to** Homework 4 -- Extra Credit Problem (File Submission) on Moodle (cpp file, not a zip file).

Note: You can earn early submission bonus points (5%) by submitting a zip file with 7 cpp files to Homework 4 (File Submission) by Friday, February 8th, 11:59 pm. Extra credit problem does not affect your early submission bonus points.

7. Homework 4 point summary

Criteria	**Pts**
CodeRunner (problem 1 - 7)	70
Multiple Choice Questions	20
Comments	5
Algorithms	5
Test cases	20
Recitation attendance (Feb 5 or Feb 7)*	-30
Total	120
5% early submission bonus	+5%
Extra credit question	+20

* if your attendance is not recorded, you will lose points. Make sure your attendance is recorded on Moodle.