CSCI 1300 CS1: Starting Computing
Ashraf, Fleming, Correll, Cox, Fall 2019
Homework 3
Due: Saturday, September 21st, by 6 pm
(5 % bonus on the total score if submitted by 11:59 pm Sep. 20th)

2 components (Moodle CodeRunner attempts, and zip file) must be completed and submitted by Saturday, September 21st, 6:00 pm for your homework to receive points.

0. Table of Contents

---

1. Objectives

- Understand and work with `if-else` conditionals and switch statements.
- Understand and work with `while` loop and `for` loops.
- 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.

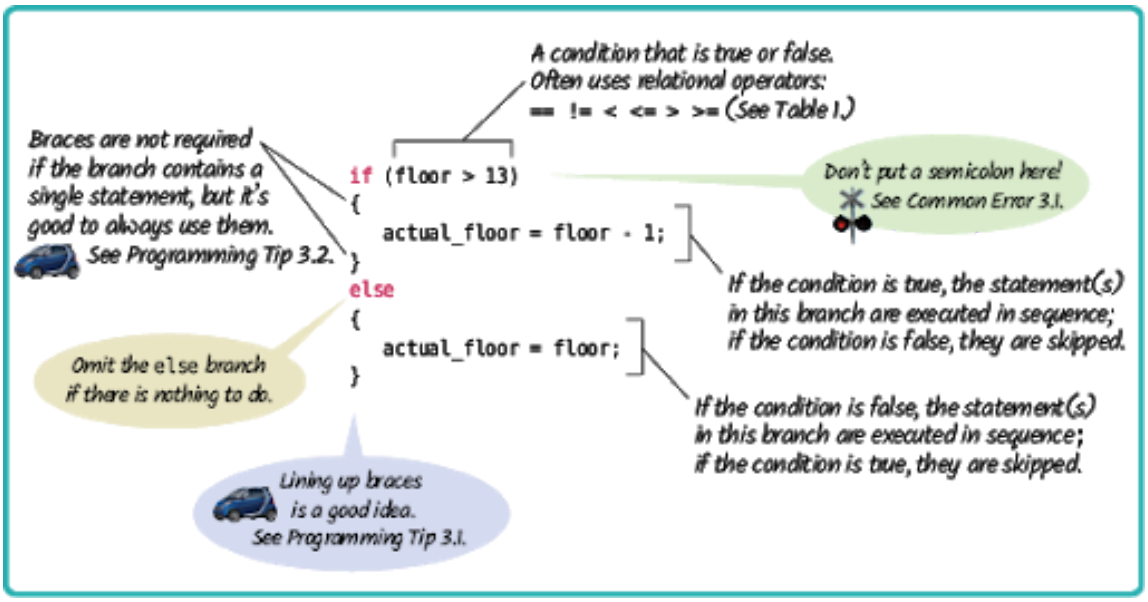
Largely, conditional statements fall into two types:

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:

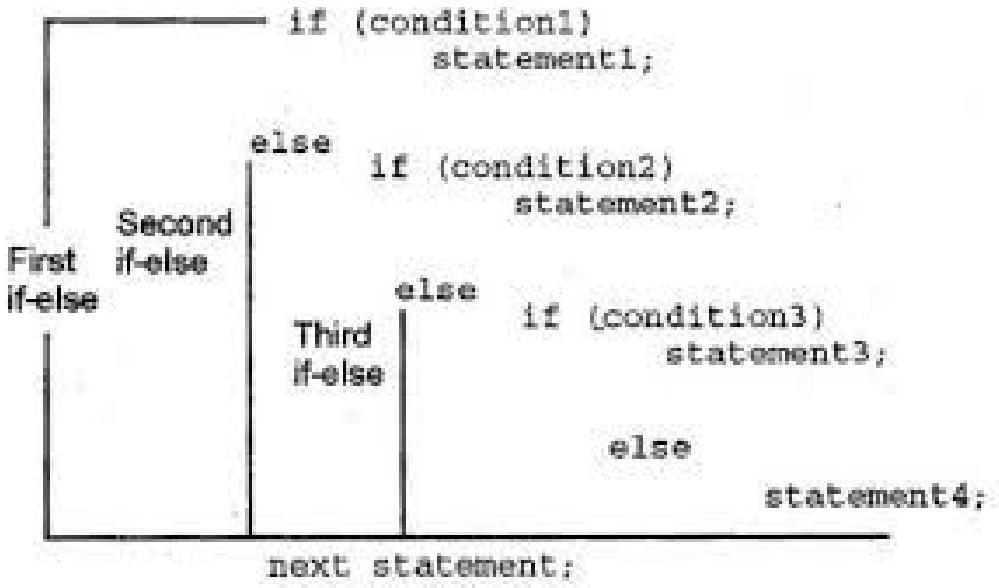
The *if statement* selects and executes the statement(s) based on a given condition. If the condition evaluates to True then a given set of statement(s) is executed. However, if the condition evaluates to False, then the given set of statements is skipped and the program control passes to the statement 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:



Nested IF-ELSE:

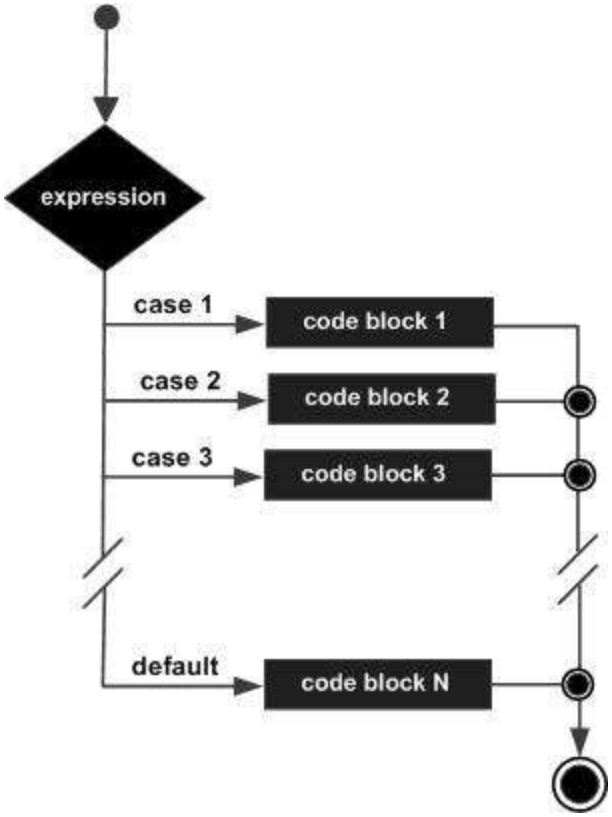
If there are more than 2 branches within your program, we can use *nested if statement*s to model this behavior within a program by putting one if statement within another. Below is an example of the syntax of a nested if statement.



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 does 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 an operand is false and false if the operand is true

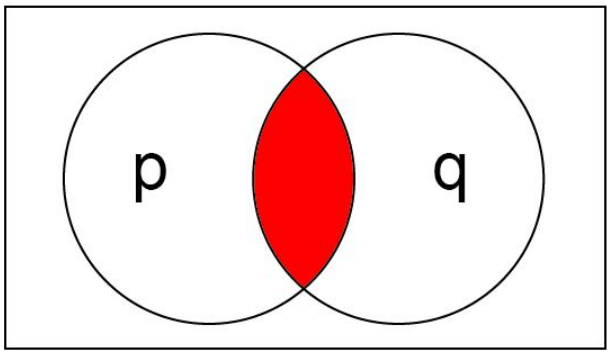
Truth Tables

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.

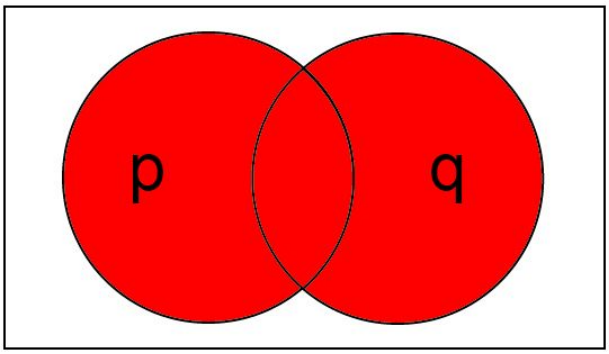
p	q	p && q
True	True	True
True	False	False
False	True	False
False	False	False



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.

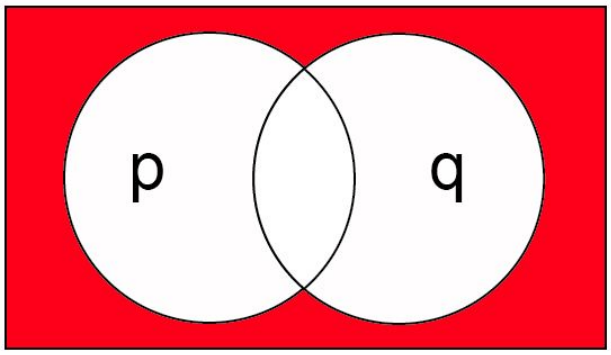
p	q	p || q
True	True	True
True	False	True
False	True	True
False	False	False



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



While Loops

Loops allow us to run a section of code multiple times. They will repeat execution of a single statement or group of statements as long as a specified condition continues to be satisfied. If the condition is not true, then the statement will not be executed.

Syntax and Form:

```
while (condition)
{
  //statement(s) to do something;
}
```

Here, *while* is a C++ reserved word, *condition* should be a Boolean expression that will evaluate to either **true** or **false**, and *statement to do something* is a set of instructions enclosed by curly brackets. If the condition is true, then the specified statement(s) within the loop are executed. After running once, the Boolean expression is then re-evaluated. If the condition is true, then the specified statement(s) are executed again. This process of evaluation and execution is repeated until the condition becomes false.

Example 1:

```
int userChoice = 1;
while (userChoice != 0)
{
   cout << "Do you want to see this question again? " << endl;
```

```
    cout << "Press 0 for no, any other number for yes." << endl;
    cin >> userChoice;
}
```

Entering '0' will terminate the loop, but any other number will cause the loop to run again. Note how we must initialize the condition before the loop starts. Setting `userChoice` equal to 1 ensures that the while loop will run at least once.

Example 2:

```
int i = 0;
while (i < 5)
{
     cout << i << endl;
     i = i + 2;
}
```

Notice how you must manually initialize `i` to equal 0 and then manually increment `i` by 2. Inserting `cout` statements into your loops is a quick way to debug your code if something isn't working, to make sure the loop is iterating over the values you want to be using. A common error is to forget to update `i` within the loop, causing it to run forever.

For loop

Sometimes you know the exact number of iterations that a loop performs. A *for* loop possess three elements:

- Initialization. It must initialize a counter variable to a starting value.
- Condition. If it is true, then the body of the loop is executed. If it is false, the body of the loop does not execute and jumps to the next statements just after the loop.
- Update. It must update the counter variable during each iteration

Syntax form:

```
for (initialization; condition; update)
{
 //statement(s) to do something;
}
```

Example 1: for loop that prints "hello" five times

```
for (int count = 0; count < 5; count++)
{
      cout << "hello" << endl;
}
```

In this loop, `count` is initialized to `0`, the test expression is `count < 5`, and `count++` to increment the count value by one.

Example 2:

```
for (int i = 0; i < 5; i = i + 2)
{
      cout << i << endl;
}
```

Notice that this example behaves in the same way as the example 2 in the `while` loop section above.

---

3. Submission Requirements

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

1. ***Work on questions on your Cloud 9 workspace:*** You need to write your code on Cloud 9 workspace to solve questions and test your code on your Cloud 9 workspace before submitting it to Moodle. (Create a directory called **hmwk3** and place all your file(s) for this assignment in this directory to keep your workspace organized)

2. ***Submit to the Moodle CodeRunner:*** Head over to Moodle to the link **Homework 3 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. ***Submit a .zip file to Moodle:*** After you have completed all 10 questions from the Moodle assignment, zip all 10 solution files you compiled in Cloud9 (one cpp file for each problem), and submit the zip file through the **Homework 3** link on Moodle.

4. Rubric

Aside from the points received from the **Homework 3 CodeRunner** quiz problems, your TA will look at your solution files (zipped together) as submitted through the **Homework 3** link on Moodle and assign points for the following:

Style, Comments, Algorithm (10 points):

Style:

- Your code should be well-styled, and we expect your code to follow some basic guidelines on whitespace, naming variables and indentation, to receive full credit. Please refer to the **CSCI 1300 Style Guide** on Moodle.

Comments:

- 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 Fall 2019
// Author: my name
// Recitation: 123 – Favorite TA
// Homework 3 - Problem # ...
```

Algorithm:

- 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
}
```

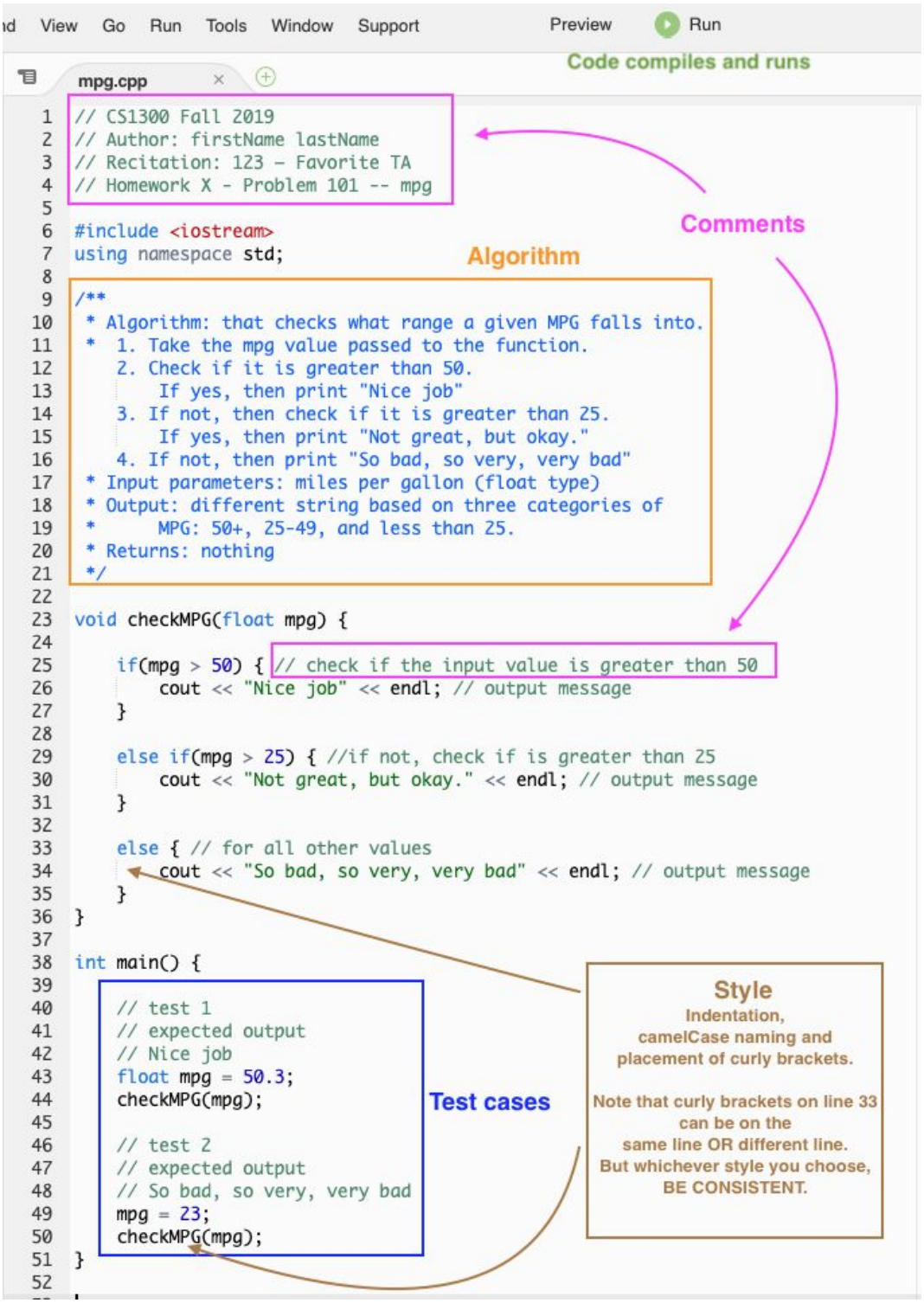
The algorithm described below does not mention in detail what the algorithm does and does not mention what value the function returns. Also, the solution is not commented. This would work properly, but would not receive full credit due to the lack of documentation.

```
/*
 * 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 **10** full programs (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 10 problems are asking you to create a function. In your solution file for each function, you should have 2 test cases present in their respective `main()` function, for a total of 20 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.




```
1  // CS1300 Fall 2019
2  // Author: firstName lastName
3  // Recitation: 123 - Favorite TA
4  // Homework X - Problem 101 -- mpg
5
6  #include <iostream>
7  using namespace std;
8
9  /**
10  * Algorithm: that checks what range a given MPG falls into.
11  *   1. Take the mpg value passed to the function.
12     2. Check if it is greater than 50.
13         If yes, then print "Nice job"
14     3. If not, then check if it is greater than 25.
15         If yes, then print "Not great, but okay."
16     4. If not, then print "So bad, so very, very bad"
17  * Input parameters: miles per gallon (float type)
18  * Output: different string based on three categories of
19  *      MPG: 50+, 25-49, and less than 25.
20  * Returns: nothing
21  */
22
23 void checkMPG(float mpg) {
24
25     if(mpg > 50) { // check if the input value is greater than 50
26         cout << "Nice job" << endl; // output message
27     }
28
29     else if(mpg > 25) { //if not, check if is greater than 25
30         cout << "Not great, but okay." << endl; // output message
31     }
32
33     else { // for all other values
34         cout << "So bad, so very, very bad" << endl; // output message
35     }
36 }
37
38 int main() {
39
40     // test 1
41     // expected output
42     // Nice job
43     float mpg = 50.3;
44     checkMPG(mpg);
45
46     // test 2
47     // expected output
48     // So bad, so very, very bad
49     mpg = 23;
50     checkMPG(mpg);
51 }
52
```

5. Problem Set

Note: To stay on track for the week, we recommend to finish/make considerable progress on problems 1-5 by Wednesday. Students with recitations on Thursday are encouraged to come to recitation with questions and have made a start on all of the problems.

Problem 1 (10 points): collatzStep

Write a function **collatzStep** which takes a single integer parameter 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 has one parameter: 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 the files you need to zip together for the Homework 3 on Moodle.

Don't forget to head over to Moodle to the link Homework 3 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 (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 has three parameters : all integers
- Your function must print one of the above three statements
- Your function should NOT return anything

Examples:

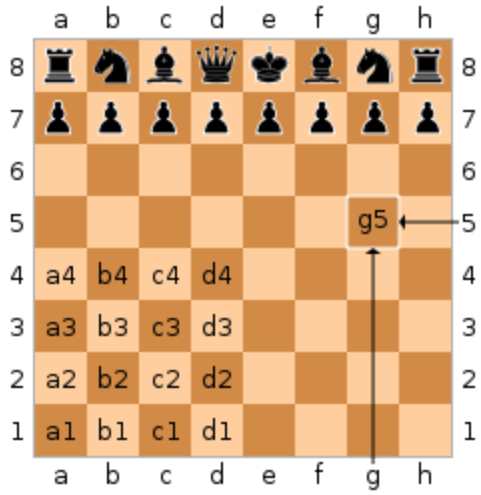
- If the parameters are (1, 2, 3), the function should print `All different`
- If the parameters are (2, 2, 2), the function should print `All same`
- If the parameters are (1, 1, 2), the function should print `Neither`

In Cloud9 the file should be called **checkEqual.cpp** and it will be one of the files you need to zip together for the Homework 3 on Moodle.

Don't forget to head over to Moodle to the link Homework 3 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): 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 has two parameters in this order: 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 arguments are `('g', 5)`, the function should print black
- If the arguments are `('c', 4)`, the function should print white
- If the arguments are `('a', 1)`, the function should print black
- If the 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 the files you need to zip together for the Homework 3 on Moodle.

Don't forget to head over to Moodle to the link Homework 3 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): countDigits

Write a function **countDigits** that takes an integer as a parameter and returns how many digits the number has. The parameter may be a positive or negative integer. Suggestion: if the number is negative, you could first multiply it with –1.

- Your function MUST be named **countDigits**
- Your function has one parameter: 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 the files you need to zip together for the Homework 3 on Moodle.

Don't forget to head over to Moodle to the link Homework 3 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 (15 points): countHours

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

- Your function MUST be named **countHours**.
- Your function has one parameter: month as an integer.
    - For example, January is 1, February is 2 and so on, up to December is 12.
- Your function should return the number of hours: an integer value.
- Your function should NOT print anything.
- Use MUST 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 the files you need to zip together for the Homework 3 on Moodle.

Don't forget to head over to Moodle to the link Homework 3 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.

Problem 6 (15 points): checkLeapYear

Write a function called checkLeapYear that takes a single integer 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 has one parameter: an integer number
- Your function should return a boolean value, true or false, according to the leap year definition (see below).
- You can NOT use `else` and `else if` statements (You can use `if` statement)

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 the files you need to zip together for the Homework 3 on Moodle.

Don't forget to head over to Moodle to the link Homework 3 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): printCollatz

In Problem 1, you just returned the next value in the Collatz sequence. In this problem, you need to print the entire sequence. Write a function to called printCollatz, which takes the starting number of the sequence and prints the entire sequence..

- Your function MUST be named **printCollatz**
- Your function has one parameter: starting number of the sequence as an integer number
- Your function should NOT return anything
- Your function must handle edge cases: when the parameter is 0 or a negative number it must print "invalid number"

Example:

1. When the parameter is equal to 4, the function should print "`4 2 1`"
    a. To find the next number in the sequence, look at the current number
        i. If the number is even, divide it by 2
        ii. If the number is odd, triple it and add 1
    b. n = 4, (since the number is even, use rule i, divide it by 2)
    c. n = 2, (since the number is even, use rule i, divide it by 2)
    d. n = 1, (we have reached 1, the end of the sequence)
    e. Therefore your output will be "`4 2 1`"
2. When the parameter is equal to 6, the function should print "`6 3 10 5 16 8 4 2 1`"
3. When the parameter is equal to -5, the function should print "invalid number"

In Cloud9 the file should be called **printCollatz.cpp** and it will be one of the files you need to zip together for the Homework 3 on Moodle.

Don't forget to head over to Moodle to the link Homework 3 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.

Problem 8(10 points): printOddNumsWhile (while-loop)

Write a function named **printOddNumsWhile** to print all positive odd integers less than or equal to a max value.

- Your function **MUST** be named **printOddNumsWhile**
- Your function has one parameter:
    - an integer parameter representing the max value.
- Your function should NOT return any value
- You should use a `while` loop to solve this problem. `For` loop is not allowed.
- Your function should print the positive odd integers less than or equal to the max value (see expected output format below)

Example:

- When the parameter is 11, the expected output should be:

```
1
3
5
7
9
11
```

In Cloud9 the file should be called **printOddNumsWhile.cpp** and it will be one of the files you need to zip together for the Homework 3 on Moodle.

Don't forget to head over to Moodle to the link Homework 3 CodeRunner. For Problem 8, 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 9(10 points): printOddNumsFor (for-loop)

Let's do the same question with a different type of loop. Write a function named **printOddNumsFor** to print all positive odd integers less than or equal to a max value.

- Your function **MUST** be named **printOddNumsFor**
- Your function has one parameter:
    - an integer parameter representing the max value.
- Your function should NOT return any value
- You should use a `for` loop to solve this problem. `While` loop is not allowed.
- Your function should print the positive odd integers less than or equal to the max value (see expected output format below)

Example:

- When the parameter is 11, the expected output should be:

```
1
3
5
7
9
11
```

In Cloud9 the file should be called **printOddNumsFor.cpp** and it will be one of the files you need to zip together for the Homework 3 on Moodle.

Don't forget to head over to Moodle to the link Homework 3 CodeRunner. For Problem 9, 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 10(15 points): printGrid

Write a function **printGrid** that takes a single integer representing the number of rows and columns in a grid, and prints a grid as shown in the example below.

- Your function **MUST** be named **printGrid**
- Your function has **one parameter**: an integer representing the size of grid
- If your function receives a parameter value `n` that is greater than zero, then it must print a grid with `n` rows and `n` columns.
- If your function receives a parameter value `n` that is less than or equal to zero, it must print "`The grid can only have a positive number of rows and columns.`"
- Your function should **NOT return** anything.

Examples:

- When the argument is equal to 3, the function should print:

```
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
```

- When the argument is equal to -4, the function should print:

```
The grid can only have a positive number of rows and columns.
```

In Cloud9 the file should be called **printGrid.cpp** and it will be one of the files you need to zip together for the Homework 3 on Moodle.

Don't forget to head over to Moodle to the link Homework 3 CodeRunner. For Problem 10, 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 (30 points): storyGenerator

Write a program that plays the Mad Libs game.

Mad Libs is a phrasal template word game where one player prompts others for a list of words to substitute for blanks in a story, before reading the – often comical or nonsensical – story aloud. - Wikipedia

Your program has a menu option that keeps asking which story the user would like to play until they opt to quit. Based on the user's story selection, prompt the user for the words for that particular story. The game has three templates:

Story1:
Be careful not to vacuum the <NOUN> when you clean under your bed.

Story2:
<NAME> is a <OCCUPATION> who lives in <PLACE>.

Story3:
In the book War of the <PLURAL NOUN>, the main character is an anonymous <OCCUPATION> who records the arrival of the <ANIMAL>s in <PLACE>.

Once the user has selected the story and input all the required words print out the story by plugging in all the words.

Sample output:

```
Which story would you like to play? Enter the number of the story (1,
2, or 3) or type 4 to quit
1
Enter a noun:
CAT
Be careful not to vacuum the CAT when you clean under your bed.

Which story would you like to play? Enter the number of the story (1,
2, or 3) or type 4 to quit
2
Enter a name:
SHIPRA
Enter an occupation:
SUPERHERO
```

```
Enter a place:
ECES112
SHIPRA is a SUPERHERO who lives in ECES112.

Which story would you like to play? Enter the number of the story (1,
2, or 3) or type 4 to quit
3
Enter a plural noun:
KITTENS
Enter an occupation:
GHOSTBUSTER
Enter an animal:
SEAL
Enter a place:
NARNIA
In the book War of the KITTENS, the main character is an anonymous
GHOSTBUSTER who records the arrival of the SEALs in NARNIA.

Which story would you like to play? Enter the number of the story (1,
2, or 3) or type 4 to quit
4
Good bye!
```

In Cloud9 the file should be called **storyGenerator.cpp** and if you attempt the extra credit it will be one of the files you need to zip together for the Homework 3 on Moodle.

Don't forget to head over to Moodle to the link Homework 3 CodeRunner(ec, storyGenerator). For this extra credit question, in the Answer Box, **paste the entire program**, including your `main()` and the other functions you created. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

6. Homework 3 checklist

Here is a checklist for submitting the assignment:

1. Complete the code Homework 3 CodeRunner
2. Submit one zip file to Homework 3. The zip file should be named, **hmwk3_lastname.zip**. It should have the following 10 files:
    - collatzStep.cpp
    - checkEqual.cpp
    - chessBoard.cpp
    - countDigits.cpp
    - countHours.cpp
    - checkLeapYear.cpp
    - printCollatz.cpp
    - printOddNumsWhile.cpp
    - printOddNumsFor.cpp
    - printGrid.cpp

3. If you have worked on the extra credit, include this file as well:
    - storyGenerator.cpp

7. Homework 3 points summary

Criteria	**Pts**
CodeRunner (problem 1 - 10)	120
Style, Comments, Algorithms	10
Test cases	20
Recitation attendance (week 4)*	-30
Total	150
Extra credit: storyGenerator	30
5% early submission bonus	+5%

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