Assignment 2

Ternary operator:

The conditional operator ? : is also another way to evaluate conditions. It operates on three operands, and is thus known as a *ternary* operator. Here is an example of its usage:

a = x < y ? x : y;

The whole expression containing this operator (the right side of the assignment statement) is called a *conditional expression*, and the sub-expression before the question mark is called the *test expression*. If this test expression is true, the entire expression takes on the value of the expression immediately after the question mark. Otherwise, it takes on the value of the expression following the colon.

Usually you only want to use this construct for choosing between two simple expressions for a particular value (both choices must be the same data type).

1. Without running, explain what the following set of statements will do:

```
bool input;
int a;
cin >> input >> a;
char *choice = ( input ? "Hello World!" : "I love C++!" );
if( ( input ? a > 1 : a < 1 ) )
{
```

```
 cout << "Why did you pick \" " << choice << "\"? " <<
endl;
}
else
 {
  cout << "Yay, you picked " << choice;
 }
```

2. Convert this for loop to a do-while loop.

```
int sum = 0;
for( int i = 0; i <5;i++ )
   sum += i;
cout << sum;
```

break and continue:

Two keywords often used with loops are *break* and *continue*.
The **break** keyword causes the entire construct to exit, and control passes to the statement immediately after the body of the loop.
For example:

```
for( n = 10; n > 0; n--)
{
        cout << n << ", ";
        if( n == 7 )
          break;
   }
```

In this example, only the first few values of n (the numbers 10, 9, 8) will be executed. As soon as the value of n becomes 7, the loop will exit.

The **continue** statement causes the program to skip the rest of the loop in the current iteration as if the end of the statement block had been reached, causing it to jump to the start of the next iteration:

```
for( int n = 10; n > 0; n--)
{
  if ( n == 7 )    continue;
  cout << n << ", ";
}
```

In this example, 10, 9, and 8 will again be displayed, but as soon as the value of n becomes 7, control will be passed to the beginning of the loop body. The rest of that iteration is skipped, and the rest of the countdown till 1, will be displayed (so the final output will be “10, 9, 8, 6, 5, 4, 3, 2, 1”). When you place a continue command within a forloop, the increase statement (in this case, n--) is executed before the loop iterates again.

3. Write a program to take in a number from a user, find its reciprocal and add it to a running sum (the running sum will be 0, when the loop initially starts). The program should repeat this procedure 10 times. However, if the user enters 0, the loop should exit, and if the user enters 1, nothing should be added. Print the final sum at the end of the program.

switch statements:

In lecture, the main type of conditional construct discussed was *if-else*. There is another type of construct, which is also sometimes used, called the *switch-case* construct. In this construct, the value of a variable is compared to a set of constants, and when a corresponding match is found, the statements associated with

that case are executed. It is like an if-else construct that can only check for equality
For example:

```
switch(number)
{
case 3: cout << "Red"; break;
case 2: cout << "Orange"; break;
case 7: cout << "Black"; break;
default: cout << "None of the above";
}
```

In this switch statement, the value of the variable numberis compared with 3, 2 and 7, and if a match is formed, the corresponding color is printed. If no match is found then “None of the above” is printed. (The break keyword is required to end each caseblock except the last.)

4. Using the *switch-case* construct write a program to input two numbers, display the following menu, and then print according to the user’s choice:

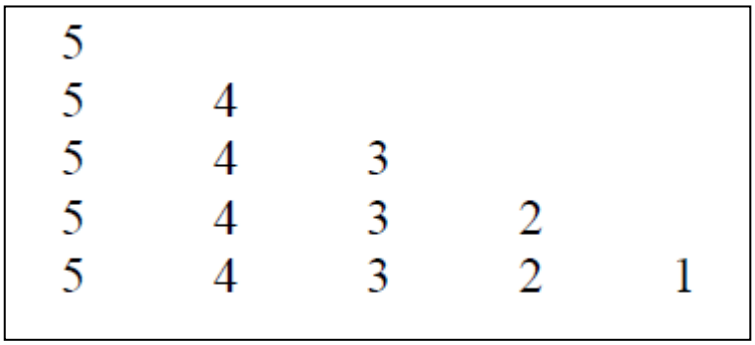
1. Difference of two numbers
2. Quotient of two numbers
3. Remainder of two numbers

For example, if the user inputs 1, then the difference of the two numbers should be printed.

5. Find the sum of the first *n* terms of the following series, where *n* is a number entered by the user:

$$-1+\left(\frac{1}{3}\right)^2-\left(\frac{1}{5}\right)^2+\ldots$$

6. Print the following pattern, using horizontal tabs to separate numbers in the same line. Let the user decide how many lines to print (i.e. what number to start at).



Series:

As specified in lecture, nested loops are used when for one repetition of a process, many repetitions of another process are needed. Similar to patterns, nested loops can be used to print the sums of nested series. For example the sum of the series 1+(1+2) +(1+2+3) +... can be found by adding one to a running sum for each execution of the inner loop, instead of printing them as you would for a pattern. As with patterns, the number of times the inner loop runs in this case depends on the value of the outer loop.

7. Write a program that inputs two numbers *x* and *n*, and find the sum of the first *n* terms of the following series:

$x+(x+x^2)+(x+x^2+x^3)+ ....$

8. An Angstrom number is one whose digits, when cubed, add up to the number itself.

For instance, 153 is an Angstrom number, since $1^3+5^3+3^3=153$.

Write a program to input a number, and determine whether it is an Angstrom number or not.
Hint: The modulus operator (%) will be useful here.