# EECE 231 — Introduction to Programming Using C++ and MATLAB
# Programming Assignment 2

September 17, 2014

- This programming assignment consists of 5 problems.
- It is due at the beginning of the next lab session by 3:00 pm. You are required to submit your solutions of the lab assignment during the the lab time. The instructor will supervise the submissions and randomly select students and check their understanding of the submission to control cheating. You are supposed to submit your own work. Cheating will not be tolerated and will be penalized (zero grade on the programming assignment, disciplinary committee, faling course grade).
- Related reading: Basic Elements of C++, Selection, and Repetition.
- *Lab structure and regulations:*
  - ⋆ The lab consists of:
    - *Occasional Solving Session (not graded but attendance mandatory)*
    - *Programming Assignment (graded)*
      Programming Assignments will be posted on Moodle on weekly basis. Typically, a Programming Assignment requires much more than the time allocated for this part in the Lab, so you are supposed to complete a part of the assignment at home. The Lab instructor will grade your assignment and help you with the problems you are facing.
    - *Occasional graded weekly quiz*
  - ⋆ Lab attendance is mandatory.

**Problem 1. (Boolean expressions).** Consider the following logical expressions involving the integer variables $i$,$j$, and $k$.

- Expression A: $i$ is less than 5.
- Expression B: $i$ is larger than or equal to 5 and less than or equal to 10.
- Expression C: $i$ is larger than 0 or $j$ is a multiple of 5.
- Expression D: $i$ is larger than 0 or $j$ is a nonnegative multiple of 5.
- Expression E: $i$, $j$, and $k$ are either all equal or all distinct.

Write a program which:

1. Declares variables for $i$, $j$, and $k$, and 5 boolean variable one for each of the above expressions
2. Prompts the user to input the values of $i$, $j$, and $k$.
3. Computes Expressions A,B,C,D, and E and stores their values in the corresponding boolean variables.
4. Prints out the values of the expressions.

**Problem 2. (Roots finder).**[1]

The roots of the quadratic equation $ax^2 + bx + c = 0$, $a \neq 0$ are given by the formula:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

In this formula, the term $b^2 - 4ac$ is called the **discriminant**. If $b^2 - 4ac = 0$, then the equation has a single (repeated) root. If $b^2 - 4ac > 0$, the equation has two real roots. If $b^2 - 4ac < 0$, the equation has two complex roots. Write a program that prompts the user to input the value of $a$ (the coefficient of $x^2$), $b$ (the coefficient of $x$), and $c$ (the constant term), and outputs the types of roots of the equation. Furthermore, if $b^2 - 4ac \geq 0$, the program should output the roots of the quadratic equation. (*Hint:* Use the function *pow* from the header file *cmath* to calculate the square root. The function *pow*, called the power function, takes two *double* input parameters $x$ and $y$ and it returns the *double* value $x^y$, i.e., $pow(x, y) = x^y$.) Do not forget to include the header file *cmath*, i.e., use the preprocessor directive $\#include < cmath >$.)

**Problem 3. (Min and Max)**

Write a program which asks the user to enter:

- a number $n$, and
- a list of $n$ integer.

Your program is supposed to find the minimum and the maximum integer in the input list.

For example, if user enters

- 5
- 10 100 2 7 5

then the min is 2 and the max is 100.

Complete the following program:

```
#include <iostream>
using namespace std;

int main()
{

    cout << "Enter the number of integers in the list: ";
    int n;
    cin >> n;




    int count = 1;// counter to keep track of the number items

    int number; // to store the next user input
    int max, min; // to store the max and the min

    cout << "Enter "<< n<<" integers: ";
```

[1]This problem is taken from "C++ Programming: Program Design Including Data Structures", Malik, 3rd edition (Exercise 4.6).

```
    while (  [... add condition ...] )
    {
       cin >> number; // read number

       if(count == 1) { // if count ==1, i.e.,  we are looking at the first number, we
                        // initialize max and min to this number
           max = number;
           min = number;
       }
       else {



          [... update max and min according to the value of number ...]


       }

       [... increment counter ....]
    }

    if (n != 0) {
         cout << "The largest number: " << max <<endl;
         cout << "The smallest number: "<< min <<endl;
    }
    else
       cout << "No input." << endl;

   return 0;
}
```

**Problem 4. (While loop: Count and add)**
Write a computer program that reads a list of integers, and then finds and prints:

1. The number of the even integers in the list
2. The sum of the even integers in the list

Your program should ask the user first to enter the number of integers in the list.

**Problem 5. (While loop and boolean variables: Testing Primality)**
A positive integer $n$ is called *a prime number* if it is not equal to 1 and if the only positive integers which divide $n$ are 1 and $n$. The first few prime numbers are $2, 3, 5, 7, 11, 13, \ldots$.

a) **(Testing Primality)** Write a program which given an integer, outputs a message indicating whether the number is prime.

*Hint:* Declare a boolean variable $isPrime$ initialized to *true*. Check the integers $2, 3, \ldots, n-1$ starting from 2 and update the variable $isPrime$ if needed after suitable considerations. At the end print the desired message depending on the value of the boolean variable $isPrime$.

b) **(A smarter test)**

Do you have to check all the integers $2, 3, \ldots, n-1$? (argue that it is enough to check if $n$ is divisible by an integer $d$ between 2 and $\sqrt{n}$ (inclusive).)

Do you have to continue your search for a divisor of $n$ if you have found one?

Inspired by the above two observations, write a more efficient version of your program in Part (a).

(*Note:* It is worth mentioning that this problem has puzzled people since ancient times. The idea underlying the test in Part (b) was discovered before 200 BC. The most efficient test known today was discovered only 12 years ago, and people are still looking for faster primality tests ...).

c) **(Density of Primes)** Write a program which given a positive integer $x$, computes the fraction $f(x)$ of prime numbers less than or equal to $x$, i.e., the number of prime numbers less than or equal to $x$ divided by $x$.

d) **(Optional)** Guess the asymptotic behavior of $1/f(x)$ as $x$ grows.

(This goes back to the 18'th century ...).