CS2211a Assignment 4

Issued on: Thursday, November 6, 2014

Due by: Thursday, 11:55 pm, November 13, 2014

- For this assignment, ***only electronic submission*** at owl.uwo.ca is required.
- ONLY user ***Courier New*** (*size = 11 pts.*)
- ***Start each question in a NEW PAGE***
- ***Write the question number is a separate line followed by an empty line***
- After finishing the assignment, you have to do the following:
  - ❖ Type your report and convert it to the PDF format (*no handwriting*),
  - ❖ The report should include:
    - Answers to *all* questions/requirements in the assignment
    - Enough test cases (as well as sample outputs) to demonstrate and cover all possible options in your program
    - A copy of all programs that you have written
    - Three *flowcharts*, one for each question
  - ❖ Prepare a soft-copy submission, including:
    - A copy of your *typed* report
    - All programs that you wrote (*each program in a file*)—*3 in total* (use meaningful program names). These files **MUST BE** text ASCII files. Do not submit them as PDF.
  - ❖ Upload the soft-copy submission file-by-file (*4 in total*), or as an archived directory.

Failure to follow the above format may cost you 10% of the total assignment mark.

- Late assignments are strongly discouraged
  - 10% will be deducted from a late assignment (up to 24 hours after the due date/time)
  - After 24 hours from the due date/time, late assignments will receive a zero grade.

Program 1 (35 marks)

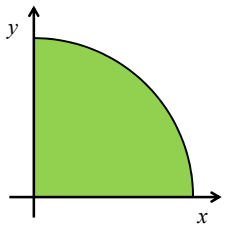
The area of a circle of radius *r* is given by

$$Area\ of\ a\ circle = \pi \times r^2.$$

Imagine that you divided this circle exactly into 4 quadrants. The area of one quadrant is then $0.25 \times \pi \times r^2$.

Let us set the radius of this circle to be $r = 1$. The equation hence becomes

$$Area\ of\ one\ quadrant\ of\ a\ circle\ of\ radius\ 1 = 0.25 \times \pi.$$



We can simply say:

$$\pi = 4 \times the\ area\ of\ one\ quadrant\ of\ a\ circle\ of\ radius\ 1.$$

To calculate the area of the shaded quadrant of the circle, use a random number generator and guess effectively the correct value of the constant π. The technique you should use is to:

- Generate a random number between 0.00 and 1.00 and assign it to *x*.
- Generate a random number between 0.00 and 1.00 and assign it to *y*.
- If $(x^2+y^2 <= 1)$, it means that this *(x,y)* coordinate lies *inside* the shaded quadrant.
- Repeat these steps *N* times, where *N* is <u>*sufficiently large number*</u> and then calculate the ratio of the points located inside the circle to an the total number of generated points, i.e., *N*. This ratio should approximate the area of one quadrant of this circle.
- Multiply the calculated ratio by 4 to find an approximation of the mathematical constant π.

Using the above procedure, draw a <u>*flowchart*</u> and write a <u>*C program*</u> to approximate the mathematical constant π.

Your program must read and validate the value of *N (a positive integer)* from the user at the beginning of execution.

Add as many inline comments as you can to make your program well documented and easy to be understood by anyone who reads it.

- Test your program using

  N = 10, N = 100, N = 1`000, N = 100`000, N = 1`000`000, N = 10`000`000, and N = 100`000`000.

- Loop inside your program to *recalculate* the value of π <u>*10 times*</u> using each of the above values of *N*. Calculate the <u>*mean*</u> (i.e., the average value) and <u>*standard deviation*</u> for <u>*results generated from the same value of N*</u>. Report all results (i.e., 70 values, 7 means and 7 standard deviations).
- Discuss your results.

$$\bar{X} = \frac{\sum_{i=1}^{i=n} X_i}{n} \qquad \sigma_n = \sqrt{\frac{1}{n}\sum_{i=1}^{i=n}(X_i - \bar{X})^2}$$

Program 2 (30 marks)

Draw a flowchart **and** write a C program that prints an `n×n` *magic square* (a square arrangement of the numbers *1, 2, ..., n^2* in which the sum of the elements in any row, column, or diagonal is the same). This program should generate a magic square of a specified size `n`, where the user will specify this size at run time. The size must be an *odd positive* integer number between 1 and 99 (to be validated by your program). You should store the magic square in a two-dimensional array.

Start by placing the number *1* in the middle of row 0. Place each of the remaining numbers *2, 3, ..., n^2* by moving *up one row* and *forward one column*. Any attempt to go outside the bounds of the array should wrap around to the opposite side of the array. For example, if *row = 0*, instead of storing the next number in *row `-1`*, we would store it in *row `n-1`* (the last row). If *column = n-1*, Instead of storing the next number in *column `n`*, we would store it in *column `0`*. In addition, if a particular array element is already occupied (filled), put the number directly below the previously stored number. Here it is a sample output:

```
Enter size of magic square: 6
Invalid size, try again...
Enter size of magic square: 3
    8    1    6
    3    5    7
    4    9    2
```

```
Enter size of magic square: 5
17       24       1        8       15
23        5       7       14       16
 4        6      13       20       22
10       12      19       21        3
11       18      25        2        9
```

Add as many inline comments in your program as you can to make it well documented and easy to be understood by anyone who reads it.

You should include enough *actual* test cases in your submission to demonstrate the functionality of your program (i.e., covering ***all*** cases).

Program 3 (35 marks)

Draw a flowchart **and** *write a C function* that determines the smallest number of $20, $10, $5, $2, and $1 bills/coins necessary to pay a dollar amount. The *function prototype* must be as follow:

```
void pay_amount(int dollars, int *twenties, int *tens, int *fives, int
*toonies, int *lonnie);
```

where the dollar amount is represented by the `dollars` parameter. The `twenties` parameter points to a variable in which the function will store the number of $20 bills required. The `tens`, `fives`, `toonies` and `lonnie` parameters are similar.

You should add as many *inline comments* as you can to make your code well documented and easy to be understood by anyone who reads it.

To test your function, *write a program* that asks the user to enter an integer value (a dollar amount), calls `pay_amount` to get the smallest number of bills/coins necessary to pay the amount, and then display the values returned by the function.

You should include enough *actual* test cases in your submission to demonstrate the functionality of your program (i.e., covering ***all*** cases).