CS2211a Assignment 3

Issued on: Tuesday, October 20, 2015

Due by: Tuesday, 11:55 pm, October 27, 2015

- For this assignment, ***only electronic submission*** at owl.uwo.ca is required.
- ONLY use **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:
    - o Answers to *all* questions/requirements in the assignment
    - o Test cases (as well as sample outputs) to demonstrate and cover all possible options in your program
    - o A copy of all programs that you have written
  - ❖ Prepare a soft-copy submission, including:
    - o A copy of your *typed* report
    - o All programs that you wrote (*each program in a separate text file*)—*3 in total* (use meaningful program names). These program files **MUST BE** text ASCII files. Do not submit them as PDF or Word.
  - ❖ 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
  - o 10% will be deducted from a late assignment (up to 24 hours after the due date/time)
  - o After 24 hours from the due date/time, late assignments will receive a zero grade.

Program 1 (35 marks)

Draw a flowchart and develop a C program that ***ask*** the user to enter two integers. The first integer value represents a *time of day* on a 24-hour clock, e.g., 1345 represents quarter to two mid-day. The second integer represents *time duration* in a similar way, e.g., 345 represents 3 hours and 45 minutes. Note that, 2520 and 2372 are not valid *time of day*. Note also that 2372 is not valid *time duration*, but 2520 is valid *time duration*. The time duration can be *positive* or *negative*. Your program should loop until getting a valid *time of day* and a *time duration*.

The program will add the *time duration* to the *time of day*, and the result is printed out in the same notation (*always as a valid 4 digit time*). For example, in the above case, the answer should be 1730, which is the time at 3 hours and 45 minutes after 1345. You should consider cases like: starting time is 2300 and duration is 200. In this case, the end time will be 0100, not 2500.

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.

Test your program using *at least* these cases (include the *actual sample outputs* in your submission)

456 -500

1234 +3750

1234 –3750

1234 –1250

123 +456

3 +4

Show what will happen when you enter the following values as a time of a day:

6420

2064

-6420

-2064

Show what will happen when you enter the following values as duration:

+2064

-2064

Program 2 (30 marks)

Draw a flowchart and develop a C program that calculates the remaining balance on a loan after each of the first *n* monthly payments. Your program should ***ask*** the user to enter the amount of loan, the yearly interest rate, the monthly payment, and *n* (the number of monthly payments). Display each balance with *two digits* after the decimal point. Your program should make sure that all entered numbers are positive and valid. Your program should loop until getting positive values.

Do not use arrays in this program.

Hint: each month, the balance is increased by the current balance times the monthly interest rate, and then decreased by the amount of the payment. Assume that the monthly interest rate is just the yearly interest rate divided by 12.

For example, if the loan is $1000, the yearly interest rate is 12%, the monthly payment is $100, and *n* is 3, then the balance after the first month is $1000 + $1000 × 1% – $100 = $910.00, the balance after the second month is $910 + $910 × 1% – $100 = $819.10, and the balance after the third payment is $819.10 + $819.10 × 1% – $100 = $727.29.

Your program should deal with cases like payment of $800 for example. In this case, the balance after the first month is $1000 + $1000 × 1% – $800 = $210.00, the balance after the second month is $210 + $210 × 1% – $212.10 = $0.00 (*yes, this payment is just $212.10, not $800*), and there will be no more payments, as the balance reached $0.00. In a case like that, you should report the amount of the last payment.

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.

Use your program to print the balance after each of the first 15 payments of a $12345 loan if the yearly interest rate is %12, and the monthly payment is $1234. Include the *actual output.*

Use your program to print the balance after each of the first 15 payments of a $12345 loan if the yearly interest rate is %12, and the monthly payment is $543.21. Include the *actual output.*

Use your program to print the balance after each of the first 15 payments of a $54321 loan if the yearly interest rate is %12, and the monthly payment is $543.21. Include the *actual output.*

Use your program to print the balance after each of the first 15 payments of a $54321 loan if the yearly interest rate is %12, and the monthly payment is $321. Include the *actual output.*

<u>**Program 3 (35 marks)**</u>

The value of the mathematical constant *e*, which is 2.718281828459045**,** can be approximated as an infinite series as follow:

$$e = 1 + 1/1! + 1/2! + 1/3! + \dots,$$

where $n! = 1\times2\times3\times4\times\dots\times n$.

Draw a flowchart and develop a C program that approximates the value of the constant *e*. Your program should keep adding terms until the value of the current term to be added becomes less than a small <u>*positive*</u> floating-point number <u>entered by the user</u>. Your program should make sure that all entered value is positive and valid. Your program should loop until getting a positive value.

Do not use recursive functions in this program.

Test your program using the following floating-point numbers *(include the input value and the actual output).*

`0.01,`

`0.001,`

`0.0001,`

`0.00001,`

`0.000001,`

`0.0000001,`

`0.00000001,`

`0.000000001,`

`0.0000000001,` and

`0.00000000001`

Your program should <u>*print the approximated value of e to 15 digits after the decimal*</u>, as well as <u>*the number of terms required to generate this value*</u>.

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.