CS2211a Assignment 3

Issued on: Thursday, October 17, 2013

Due by: Thursday, 11:55 pm, October 24, 2013

- In this assignment, only electronic submission (***attachments***) at owl.uwo.ca is required. Attachments must include:
  - ***ONE pdf*** file that has the three flowcharts and all sample outputs for all questions.
  - ***text*** soft copy of the programs that you wrote for each question (*one program attachment per question*), i.e., ***THREE*** C program files in total.

  So, in total, you will submit 1 + 3 = 4 files.

- 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 (30 marks)

~~Write a *pseudo code*~~ ***Draw a flowchart*** and a C program that reads 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 valid *time of day* and *time duration*.

The program will add the *time duration* to the *time of day*, and the result is printed out in the same notation. 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 100, not 2500.

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

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.

Program 2 (35 marks)

~~Write a *pseudo code*~~ ***Draw a flowchart*** and 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 these numbers are positive. Your program should loop until getting positive values.

Hint: each month, the balance is increased by the balance times the monthly interest rate, and 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, 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.

Use your program to print the balance after each of the first 10 payments of a \$100,000 loan if the yearly interest rate is %10, and the monthly payment is \$10,000.

What will happen if the monthly payment is \$15,000? What will happen if the monthly payment is \$750? Discuss the results.

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

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.

Program 3 (35 marks)

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

$$e = 1 + 1/1! + 1/2! + 1/3! + ...,$$

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

~~**Write a *pseudo code***~~ *Draw a flowchart* and a C program that approximate *e*. Your program should keep adding terms until the value of the current term becomes less than a small positive floating-point number entered by the user. Test your program using the following floating-point numbers.

`0.01,`

`0.001,`

`0.0001,`

`0.00001,`

`0.000001,`

`0.0000001,`

`0.00000001,`

`0.000000001,`

`0.0000000001,` and

`0.00000000001`

You should print the approximated value of *e*, as well as the number of terms required to generate this value.

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.