Exercise Session 3: iterations

October 12, 2014

E1. Write a program that prints the sum of the first n natural numbers: $1 + 2 + \ldots + n$. Your output should look like this:

Enter the number of terms: 10
The sum of the first 10 natural numbers is 55.

E2. Write a program that converts temperatures in Celsius to Fahrenheit. The program should accept a stream of temperatures from the user and stop when the user enters q.

E3. Write a program that reads a sequence of integers (stops accepting input when the user enters 0) and computes the alternating sum of all elements. For example, if the program is executed with the input data 1 4 9 16 9 7 4 9 11 0, then it computes $1 - 4 + 9 - 16 + 9 - 7 + 4 - 9 + 11 = -2$.

E4a. A prime number is one that is not divisible by any number other than 1 and itself. Write a program that asks the user to input a positive integer and checks whether the given integer is a prime number.

E4b. Extend the above program to input an integer and print all prime numbers up to that integer.

E5. Write a program to input a number and reverse the order of digits. For example if the input to the program is 12345, the output should be 54321.

E6a. Write a program that computes the factorial of a positive integer. $n! = 1 * 2 * 3 * \cdots * n$

E6b. Use the above program to calculate e^x. The value of e^x can be computed as the power series

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \tag{1}$$

Of course, you can't compute an infinite sum. Just keep adding values until an individual summand (term) is less than a certain threshold.

E7. Prove the correctness of following algorithm, which should calculate the smallest power of 7 greater than 10,000:

```
x = 1
basis = 7
bound = 10000
while x <= bound:
        x = x * basis
```

E8. Prove the correctness of following algorithm, which should calculate n^n $(n > 0)$, and stores the end result in the variable *result*.

```
# precondition: n >= 1
result = 1
count = 0
while count != n:
        result = result * n
        count++
        n++
```

E9. Credit Card Number Check: The last digit of a credit card number is the check digit, which protects against transcription errors such as an error in a single digit or switching two digits. The following method is used to verify actual credit card numbers but, for simplicity, we will describe it for numbers with 8 digits instead of 16:

1. Starting from the rightmost digit, form the sum of every other digit. For example, if the credit card number is 4358 9795, then you form the sum $5 + 7 + 8 + 3 = 23$.
2. Double each of the digits that were not included in the preceding step. Add all digits of the resulting numbers. For example, with the number given above, doubling the digits, starting with the next-to-last one, yields 18 18 10 8. Adding all digits in these values yields $1 + 8 + 1 + 8 + 1 + 0 + 8 = 27$.
3. Add the sums of the two preceding steps. If the last digit of the result is 0, the number is valid. In our case, $23 + 27 = 50$, so the number is valid.

Write a program that implements this algorithm. The user should supply an 8-digit number, and you should print out whether the number is valid or not. If it is not valid, you should print out the value of the check digit that would make the number valid.

Here are sample program runs:

Enter 8 digit credit card number: 43589794
The credit card number is not valid.
The last digit should be 5

Enter 8 digit credit card number: 43589795
The credit card number is valid.

E10. Write a program which prompts the user for a height of a pyramid (a whole number bigger than 0) and then draw one. A sample run might look like:

```
Type in the height of the pyramid: 7

      #
     ###
    #####
   #######
  #########
 ###########
#############
```