**CSci 1500 - Assignment 6 – 100 pts.**

**Due Date: Dec 10, 2019**

Write C++ programs.

**What you need to turn in:** A printed copy of your C++ code for each of the problems, arranged in order, and stapled together. Include each name of your group on the front page of what you turn in. Clearly identify C++ code solves which problem.

1. **Recursion.** Use the following definition, in which *x* is a floating-point number and *n* is an integer, to write and test a recursive exponentiation function, recPower():



**Note**: use long double for both x and the return type.

write a main() driver to test your functions that will repeatedly ask for input of arguments, read them, call the function, and display the result.

Here is an example of what output should look like from running your program (user input shown in bold):

Enter a float-point number x: 3.4

Enter an integer number n: -4

The value of x^n = 0.007483148

Continue (y or n)? y

Enter a float-point number x: -2.4

Enter an integer number n: 0

The value of x^n = 1.000000000

Continue (y or n)? y

Enter a float-point number x: 2.718

Enter an integer number n: 5

The value of x^n = 148.336238492

Continue (y or n)? y

Enter a float-point number x: -50.7

Enter an integer number n: -4

The value of x^n = 0.000000151

Continue (y or n)? n

1. **Recursion.** Write a recursive function called digits that will return the number of digits in its long parameter. Your function should work for arguments that are negative or zero as well as positive.

write a main() driver to test your functions that will repeatedly ask for input of arguments, read them, call the function, and display the result.

Here is an example of what output should look like from running your program (user input shown in bold):

Enter an integer number (negative or zero is OK): -123456789

The number of digits in number -123456789 is: 9

Continue (y or n)? y

Enter an integer number (negative or zero is OK): 0

The number of digits in number 0 is: 1

Continue (y or n)? y

Enter an integer number (negative or zero is OK): 987654321

The number of digits in number 987654321 is: 9

Continue (y or n)? y

Enter an integer number (negative or zero is OK): 1234567890

The number of digits in number 1234567890 is: 10

Continue (y or n)? n

3. Write a program that will prompt the user to enter a Canadian city and a Canadian province two-letter abbreviation (e.g., BC for British Columbia – see below for a list of all provinces and associated abbreviations) and then, using concatenation, create and display a string containing the city name followed by a comma, followed by the entire province name. For example, if the user enters the city “Vancouver” followed by the province abbreviation, “BC”, the program will create and display the string “Vancouver, British Columbia”. Read the city and the province abbreviation into two strings. Use a string-valued function, **province**, to convert a province abbreviation to its name. Also use the string concatenation operator to form the final output value. Note: strings may be compared with relational operators. Here is an example of what output should look like from running your program (user input is shown in **bold**):

Enter city: **Moose Jaw**

Enter province abbreviation: **AB**

You entered: Moose Jaw, Alberta

**Canadian provinces and abbreviations:**

Alberta (AB)British Columbia (BC)Manitoba (MB)

New Brunswick (NB)Newfoundland (NF)Nova Scotia (NS)

Northwest Territories (NT)Ontario (ON)Prince Edward Island(PE)

Quebec (PQ)Saskatchewan (SK)Yukon (YK)

4. Write a program to input a sentence, one word at a time, terminated by one of these three characters: **. ! ?** Then, output the number of words in the sentence, the average word length, and the longest word in the sentence (for ties, display the last one). **For simplicity**, assume that the only punctuation marks within the sentence are one of the two characters**, ; where each punctuation mark is separated from the preceding word by a space.**

**Note:** The user may enter the sentence over more than one input line.

Here is an example of what output should look like from running your program (user input is shown in **bold**):

Enter a sentence:

**Alas , there was no more fuel ; we began**

**losing altitude rapidly !**

Number of words = 11

Average word length = 4.54545

Longest word: altitude

**Hints:**

* Use a string class variable to read in each word and each punctuation mark. Since each word and each punctuation mark is separated by a space, use cin to read in the words and punctuation marks.
* Use the string variable’s length() function to determine the length of a word.
* **Note: You must include**  #include <string> and include a using namespace std; statement in your program.
* Use relational operators to determine when punctuation has been read.

5. Write a program that produces the binary equivalent of a non-negative integer input by the user. Following is an example of doing this conversion to generate the bits in reverse order.

Example: Convert 22 decimal to binary.

, remainder = 0

, remainder = 1

, remainder = 1

, remainder = 0

, remainder = 1

Done, since the quotient = 0.

So, 22 decimal = 10110 binary.

Note that this is a repetitive process, but the binary digits (the remainders) are determined “backwards”. So that you can display the binary digits in the right order, make use of a string class variable and the string concatenation operator. Initialize an accumulator string variable to a blank string (i.e., initialize to “”) and then repeatedly concatenate the appropriate remainder bits to the accumulator string. (**Note:** Concatenate ‘1’ or ‘0’ characters to the accumulator string, not 1 or 0 integer values.) Be careful to concatenate the remainder bits into the string in the proper way so that when you are done, the string contains the remainder bits in the correct order. **Remember:** The concatenation operator is not commutative (i.e., *string1* + *string2* does not necessarily equal *string2* + *string1*.) Then display the binary equivalent string.

Here is an example of what output should look like from running your program (user input is shown in **bold**):

Enter a positive integer: **22**

22 decimal = 10110 binary

6.When *P* dollars is deposited into a savings account that pays *r* percent annual interest, compounded annually, the amount of money in the account after one year, , is given by



If the money is left in the account for a second year, the amount in the account after the second year, , is given by



If the money is left in the account for a total of three, four, …, or *n* years, the amount in the account after that many years is given by









Write a program that prompts for and reads the values of *P*, *r*, and a desired ending amount of money and then uses a task-controlled type of loop to determine how many years it will take to accumulate that much money. Note: The user will enter the value of *r* as a percentage, but for calculation purposes, the program should use *r* in decimal form (i.e., 5% = 0.05 decimal).

Here is an example of what output should look like from running your program (user input is shown in **bold**):

Enter initial deposit: $**200**

Enter interest rate (%): **5**

Enter desired end amount: $**1000**

It will take 33 years.

Ending balance = $1000.64

7. Write a program that will prompt for and read a number of rows, *nrows* (a positive integer) and then generate a pattern of *nrows*  rows of asterisks as shown in the example run, below. Note: The first row contains *nrows*-1 blanks followed by 1 asterisk, the second row contains *nrows*-2 blanks followed by 3 asterisks, etc., until the last row contains no blanks followed by 2\**nrows* – 1 asterisks. Include a do loop to validate the user input of *nrows* – if it is not positive, the program should generate an error message and force the user to re-enter the value until it is positive.

Here is an example of what output should look like from running your program (user input is shown in **bold**):

Enter number of rows (>0): **-4**

Invalid entry – try again!

Enter a positive integer: **4**

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*