EECE 1080C/CS1021C - Laboratory 10 - Summer 2019

Pointers and File I/O

Topics covered: Pointers & File I/O

Due:

Sunday July 28, 2019 by 11:59PM uploaded on Blackboard

Objective:
This assignment is designed for the student to learn about pointers in C++ as well as basic file I/O and cryptography. It will reinforce the concepts of strings, vectors, and functions.

Collaboration:

As with most laboratory assignments in this course this laboratory assignment is to performed by an individual student. You can help each other learn by reviewing assignment materials, describing to each other how you are approaching the problem, and helping each other with syntax errors. You can get help from tutors, teaching assistants, and instructors. Having similar code for some assignments is expected but most assignments have multiple different solutions. Your code is expected to be different.

Please document any help you receive from other students, teaching assistants, or instructors. Just add the names to the top of your source file.

Having someone else code for you, sharing code with other students, or copy-pasting code from the internet or previous terms , is cheating. A helper should "teach you to fish, not feed you the fish". Laboratory assignments prepare you for exams so be smart.

Highlights:

- Please review the chapters on pointer and file I/O.
- Download the Lab10.zip starter folder.
- Review the **grading Rubric** at the end of specification.
- **Lab Attendance Milestone: Completion of Part A**
- Please ask appropriate questions when necessary.
- Take a few minutes to read this entire document, at least once, before beginning to program it.

EECE 1080C/CS1021C - Laboratory 10 - Summer 2019

Pointers and File I/O

Topics covered: Pointers & File I/O

Part A: Pointers

I. Pointer Gymnastics:

We are providing you with 3 files: Pointers.h, pointer-sandbox.cpp, and manip.h which are available in the Lab 10/Part A folder.

The sandbox file should be complete. If you compile it you will see what the output should be and what it is now. Your goal is to make them the same.

You will be editing the manip.h file and compiling the sandbox file.

You will use the Pointers class implemented in Pointers.h. You'll note that EVERY sample output calls a manipX function, with X matching the function number. These functions are located in manip.h and are currently empty. Your job is to fill in the manip functions in manip.h so your output would match the sample code!

Writing the manip functions will require you to FULLY understand what the Pointers class does, what private variables it stores, and most importantly what the member functions can do. Study the sample output and try to imagine how to setup your pointers to match the output.. Each manip function is passed parameters which allow it to change a Pointer instance somehow in order for you to get the correct output

DRAW PICTURES of what memory looks like! Each manip function can be filled in with less than or equal to 3 lines of code, with 7 of them requiring only one line. You may use ANY C/C++ tricks you can to get the tests to pass.

Your proposed changes will only affect the manip.h. File.

Sample Output

```
maninp1:
10       = 10
maninp2:
45       = 45
```

```
maninp3:
383      = 383
maninp4:
0x7fff8072b6fc   = 0x7fff8072b6fc
1       ==  1
maninp5:
45       = 45
maninp6:
10       = 10
maninp7:
15       = 15
maninp8:
199      = 199
maninp9:
0xbe9c20         = 0xbe9c20
maninp10:
199      = 199
```

II. **Simple Linked List:**

Modify LList.h to implement the functions or operator overloading listed below. Most of the functions are stubbed out so you just need to add the proper logic.

You will be editing the LList.h file and compiling the sandbox file.

Here is a list of the required methods you must fill in:

LList.h

- push_back(int) - Place the given value on the end of the linked-list.
- reverse() const - Return a new linked-list which is the current one reversed.
- size() - Returns the size (length) of the linked-list.
- empty() - Return true if the list is empty otherwise return false.
- pop_back() - Removes the last element of the linked-list. If the list is empty do nothing.
- operator==(LList) - Return true if the current linked-list contains identical values to the passed one.
- Overload the plus (+) operator to add two lists together.

Extra Credit: 10-Points Each

EECE 1080C/CS1021C - Laboratory 10 - Summer 2019

Pointers and File I/O

Topics covered: Pointers & File I/O

- getAt(unsigned) - Return the value at the given position, or throw a logic error if the index is invalid. See the overloaded [] operator for hints on how to throw an exception.
- setAt(int, unsigned) - Assign the given value to the given position (0-based). Throw a logic error if the index is invalid.

You can use list-sandbox.cpp to experiment with the linked-list classes. Compile list-sandbox.cpp to a list-sandbox executable. List-sandbox.cpp will not be graded.

Tip for Testing: Use the list-sandbox.cpp to create linked list objects and also test out your various list modifying functions. LList.h has a definition for overloaded << operator. This member function can let you print out the contents of the list and hence can be a helpful function to test your code in list-sandbox.cpp. Also, size() is called within this function so you will need to implement that one first.

Part B: File I/O & Cryptography

I. Rot13 Encoding/decoding (rot13.cpp)

Objective:

The goal of part B is to create a program to encode files and strings using the rot13 encoding method. Information about the rot13 method can be found at http://www.rot13.com/.

Implement the rot13 algorithm using functions, strings and file I/O. You program should use at least 3 to 4 functions. It should be able to read in a text file from a specified filename, encode using rot13, and write it to a specified file. It should be able to decrypt it in the same way.

Create a rot13.cpp file. You will need to provide the file I/O and the functions that implements the rot13 algorithm. Note: rot13 is very specific version of the Caesar cipher.

EECE 1080C/CS1021C - Laboratory 10 - Summer 2019

Pointers and File I/O

Topics covered: Pointers & File I/O

The rot13 algorithm just simply rotates each individual alphabetic character by 13 characters. It leaves the numbers, punctuation, etc alone. To “encrypt” you just rotate each character by 13 characters. To “decrypt” you just rotate (in either direction) the characters by 13 as well. Please preserve the case of the rotated letter.

You may want to convert all lowercase to uppercase characters before performing math on them to avoid overflows. You will want to convert them back to lowercase when done.

Example 1:

Given the letter ‘a’ the encrypted letter becomes ‘a’ + 13 = 110 or ‘n’

Given encrypted letter ‘n’ than ‘n’ + 13 (or 123) is greater than ‘z’ so it wraps around (in this case back to ‘a’).

If using a char data type, the above would “overflow” and become a negative number. It is suggested that you convert lowercase characters to uppercase before performing the rotation and then convert back to lowercase.

Example 2:

Assuming your input file contains the following text:

The quick brown fox jumps over the moon!

Your encrypted file would be:

Gur dhvpx oebja sbk whzcf bire gur zbba!

Note: The decrypted text should be the same as the original file text.

EECE 1080C/CS1021C - Laboratory 10 - Summer 2019

Pointers and File I/O

Topics covered: Pointers & File I/O

II. Implement the modified Caesar cipher

Objective:

The goal of part C is to create a program to encode files and strings using the caesar cipher encoding method. Information about the caesar method can be found at http://www.braingle.com/brainteasers/codes/caesar.php.

Note: the standard caesar cipher uses an offset of 3. We are going to use a user supplied string to calculate an offset. See below for details on how to calculate this offset from this string.

First create ***caesar.cpp***. You are required to implement this as a ***set of at least three to four functions***. It should be able to read in a specified text file, encode using a modified caesar cipher, and write it to a specified file. It should be able to decrypt it in the same way.

Using the rot13.cpp file as a template just modify the algorithm to receive a string as a key. You will use this key to calculate the rotation count. You rotate in the "right" direction for encryption and in the "left" direction for decryption.

The standard caesar cipher uses a 3 character offset for rotation. However, we are going to use an ASCII string (as a key) to determine this offset. See the following examples for details.

An example of how to calculate the key:

Given the sample key "deF":

First add all the ASCII values: ASCII_SUM = 'd' + 'e' + 'F' = 100 + 101 + 70 = 271

Than calculate the rotation count by using the following equation:

Count equals (ASCII_SUM % 23) + 3 = (271 % 23) + 3 = 21

EECE 1080C/CS1021C - Laboratory 10 - Summer 2019

Pointers and File I/O

Topics covered: Pointers & File I/O

The plus three in the above equation forces the code to rotate characters by at least three (3) positions.

Example:

Key: deF

Original file text:

This is a secret ring decoder that I found in a cracker jack box!

Encrypted file text:

Ocdn dn v nzxmzo mdib yzxjyzm ocvo D ajpiy di v xmvxfzm evxf wjs!

Note: The decrypted text should be the same as the original file text.

Lab Grading Rubric:

1. 10% Milestone
2. 50% Part A
3. 40% Part B