Exam in Programming Proficiency Part I

Directions:

Published text will be allowed (open book policy), but no lecture notes, copies of personal program listings, or electronic media (hard drives, flash drives, CD's, cell phones, etc.) will be allowed except for authorized software (compiler, word processor, IDEs like Visual Studio, XCode) already installed on the lab computers. Furthermore, no access to the Web will be allowed during the exams except for accessing the Titanium site.

You might find yourself under some time pressure in this examination.

Please check the point value for each problem so that you do not spend more time on one problem than it is worth. Please make sure to read each problem carefully before working on it.

Please PRINT your name: _____

Please SIGN your name: _____

Please make sure to read each problem carefully before working on it. Upload your C++ files to Titanium. **Every file you turn in must have your name at the beginning as documentation**. **NO OTHER DOCUMENTATION IS REQUIRED unless specifically stated.**

Problem	Max	Earned
1	20	
2	20	
3	20	
Total	60	

Problem 1. Write a C++ program that takes an array of characters and checks if it is a "palindrome" – that is if the array has the same contents whether read from the front or the end. If the array is a palindrome, it should print "True"; "False" otherwise.

For example, if the input is: 'r', 'a', 'c', 'e', 'c', 'a', 'r'., the output must be: True
Another example, if the input is: 'a', 'n', 'n', 'a', the output must be: True. (Note that palindromes can be odd or even in length.)
An array can can be checked for a palindrome by comparing its zeroth element with its last element, 1st element with second last, and so on.

Write your program in exactly the following way:

Write a Boolean function called **isPalindrome** that takes two parameters:

1. inputArray of chars
2. length of inputArray (int)

and returns True if inputArray is a palindrome, and False otherwise.
The function **isPalindrome** checks the values from the inputArray as described above.

An example: Calling **isPalindrome** (inputArray, 7) returns *True*:

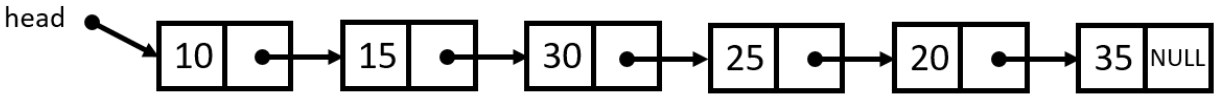
inputArray
'r'
'a'
'c'
'e'
'c'
'a'
'r'

Write a main function for your C++ program that asks the user to enter 7 chars and reads them into the inputArray. The program should then call your **inPalindrome** function. Finally, the main function prints out the return value of the function call.

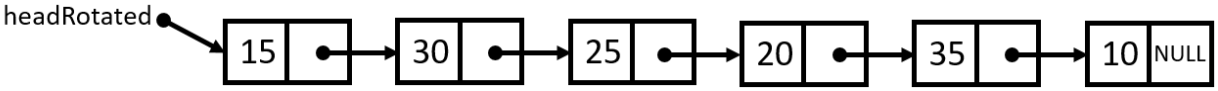
Do NOT use any STL classes (such as std::string, std::vector). Name your program **prob1.cpp**. Submit *only* prob1.cpp to the Final Part 1 - Problem 1 Assignment on the CPSC 301 Titanium site. **Remember to put in your name at the start of the file.**

Problem 2. Write a complete C++ program to demonstrate "rotating" of a singly linked lists of integers – all elements of the linked list get shifted to the left by one position and the first node becomes the last. The main component of your program must be a function called **`rotateLeft`** that takes a pointer (head of the linked list) as input parameter. This function should modify the linked list and return a pointer to the modified linked.

For example, **`headRotated = rotateLeft (head)`**
Input:



Output (return value):



You should then write a function called **print** that takes a pointer to a linked list as a parameter and prints out the integers in the linked with a space between the integers. Call the print function in your main program to show the contents of the merged list.
Your main program will have the following structure:

```
// implementation of linked list
// implementation of rotate function
// implementation of print function

int main() {
   // create linked list by reading (say, 6) integers from the user
   // declare headRotated
   headRotated = rotateLeft (head);
   print (headRotated);
}
```

The output of your program for the example input shown above should be the following:

```
15  30  25  20  35  10
```

Implementation of linked list: Keep it simple! Only implement what you need to solve this problem (i.e., just define a node and a function to add to a linked list). **You do not need to use classes.** You may define a Node as a struct as below.

```
struct Node
{
   int data;
   Node * next;
};
```

If you want to use a Node class, keep it simple with the data members public and just the constructor as a member function. **Do not** make **print** into a member function of the Node class. Also, do not bother making a List class. Do NOT use any STL classes (such as std::list, std::forward_list).

Please name the file with the program **prob2.cpp.** If you wrote a Node class in a separate file(s), you may name them using standard convention, but be sure to put your name as a comment at the start of every file. Submit all .cpp and .h files you write through the EPP Part I-Problem 2 assignment on Titanium.

Problem 3. Write a **recursive** function called **`product`** that takes a pointer to a linked list node (with integer data), and returns the product of the integers in the linked list nodes. You may assume that the list will not be empty.

The function prototype is

```
int product (Node * ptr);
```

You can reuse the code that you wrote for Problem 2. Call your recursive function **`product`** with the merged linked list and cout the return value. For the example shown in Problem 2, **`product`** should return `78750000` (35 * 15 * 30 * 25 * 20 * 10 = 78750000).

Note that even if you could not write a correct program for Problem 2, you can still do Problem 3 by writing the `product` recursive function and calling it correctly.

A non-recursive version of `product` will get no credit.

The revised program with the `product` function definition should be called **prob3.cpp**. Submit it through the Problem 3 assignment in the EPP Part 1 section on Titanium. Remember to put in your name at the start of the file.