Assignment 01

Instructions:

- **The assignment requires writing a complete header file for the 10 tasks.**
- **Accompanying these instructions are a few header files that you must use for some of the tasks. Include them in your header file.**
- **Your header file can only include the libraries string, iostream, fstream, sstream, cstdlib, ctime, cmath, cctype, and the accompanying header files.**
- **Your submission must be submitted to the Assignments directory of your github repository and/or as an attachment on Google classroom under the Assignment01 assessment. The file must remain header files.**
- **Cheating of any kind is prohibited and will not be tolerated.**
- **Violating and/or failing to follow any of the rules will result in an automatic zero (0) for the assignment.**

Grading:

Section	Maximum Points	Points Earned	Deadline
Beginning	4		02/17
Intermediate	3		03/01
Advance	3		03/22
Total	10		03/22

Beginning

1. Write an int function named `PartitionCount()` whose header is

   ```
   int PartitionCount(Array<int>& data,int target)
   ```

   It returns the number of elements of *data* whose values are less than or equal to *target*.

2. Write a double function named `Distance()` whose header is

   ```
   double Distance(Array<double>& data)
   ```

   It returns the distance between the maximum value and minimum value of the elements of *data*. If *data* contains at most 1 element, the function returns 0.

3. Write the double function named `Median()` whose header is

   ```
   double Median(const Array<int>& data)
   ```

   It returns the median of the values of the elements of *data*. If *data* is empty, it returns 0.

4. Write a generic *Node* pointer function named `ReverseList()` whose header is

   ```
   template<typename T>
   Node<T>* ReverseList(Node<T>* data)
   ```

   It returns a copy of *data* such that the values of its nodes are in the reverse order of nodes of *data*.

Intermediate

5. Write a void function named `Concatenate()` whose header is

   ```
   template<typename T>
   void Concatenate(Node<T>* data,Node<T>*& end)
   ```

   It appends *data* to the end of *end*.

6. Write a bool function named `ProperEnclosure()` whose header is

   ```
   bool ProperEnclosure(string str)
   ```

   It returns true only if *str* represents a proper enclosure of lowercase letters; otherwise, it returns false. A string represents a proper enclosure of lowercase letters if it consists only of lowercase letters and the string can be reduced to an empty string by

   - removing the outermost character on both ends of the string if they are equal, or
   - removing the outermost character and its adjacent character on both ends of the string if for both ends the character and its adjacent are equal

   For instance, the callers `ProperEnclosure("baabbccb")`, `ProperEnclosure("abcddcba")`, and `ProperEnclosure("sstt")` will all return true.

7. Define a class named *Printer* that inherits a string *QueueInterface* that is implemented with an array. It should be a queue of a fixed size of 30. The class should override all of the methods of the interface *QueueInterface*, define its special member functions, and define

   - ☐ a string constant method named `Batch()` that takes no parameters and returns the concatenation of the elements of the queue in order with a newline between each element.
   - ☐ a bool constant method named `IsFull()` that takes no parameters and returns true only if the queue is not at its capacity.

Advance

8. Write a void function named `HexadecimalCount()` whose header is

   ```
   void HexadecimalCount(int n)
   ```

   It displays the count up from 1 to the absolute value of *n* in hexadecimal. It alphabet should be uppercase letters.

9. Define a class named *RangeStack* that inherits an int *StackInterface* that is implemented with an array. It should be a stack of a fixed size of 50 that also allows the view of its maximum and minimum values. The class should override all of the methods of the interface *StackInterface*, define its special member functions and define

   - ☐ a constant int reference constant method named `Max()` that takes no parameters and returns the maximum value from the stack.
   - ☐ a constant int reference constant method named `Min()` that takes no parameters and returns the minimum value from the stack.
   - ☐ a bool constant method named `IsFull()` that takes no parameters and returns true only if the stack is not at its capacity.

   All but two methods must have a constant runtime constant (this does not include the special member functions).

10. Write the int *Node* pointer function named `Sum()` whose header is

    ```
    Node<int>* Sum(Node<int>* opr1,Node<int>* opr2)
    ```

    Given that *opr1* and *opr2* represent two positive integers such that their nodes are digits of the numbers in order, the function returns a linked list that represents the sum of *opr1* and *opr2* is the same format. For instance, if *opr1* = `[1]->[2]->[3]` and *opr2* = `[3]->[8]`, then the function would return `[1]->[6]->[1]`.