Data Structures 2021
Assignment (2) [100 Points]

Deadline: <u>5 June 2021</u>

Submission:

- This is a group assignment. The team consists of up to **THREE** students.
- At least one team member should submit the compressed group solution as a zip file containing the programs under assignments (name your assignment file "DS_A2_<Bio/SWE>_ID1_ID2_ID3.zip"). e.g. DS_A2_Bio_20179983_20179985_20179990.zip.

About this assignment:

- The assignment is marked by 100 points
- All team members should understand all assignment problems.
- Assume any missing details and search for information yourself.
- Any cheating in any part of the assignment is the responsibility of the whole team and whole team will be punished.
- Your solution should compile, run without run time errors, leave no memory leaks or dangling pointers, and handle all cases.

Problems:

Problem 1: Linked lists [40 points]

In this problem, you should develop a linked list class similar to that provided in the C++ STL. The public interface of your class should provide basic insertion and deletion functions. In addition, it should provide an iterator class as an inner class in order to access the data stored in the list. For example, if we have a list of three elements, and want to access the second element, we should

declare an iterator and initialize it to the position of the first element, and move to the second position as shown in the code below:

```
list<int> myList;
myList.push_back(1);
myList.push_back(2);
myList.push_back(3);
list<int>::iterator it = myList.begin();
it++;

cout<< *it;
```

notice the usage of the scope operator in the declaration of the iterator, this is because the iterator class is defined as an inner class inside the list class:

```
template<class type>
class myList {
public:

class iterator {
// your code for the iterator class here

};
// your code for the list class her

};
```

Your list class should be a template class. [3 points]
The list class should have the following public interface:

- `list()` − default constructor. [2 points]
- `list(type value, int initial_size)` − constructs a list having 'initial_size' elements whose values are 'value'. [3 points]
- `~list()` − a destructor to clear the list and leave no memory leaks. [3 points]
- `int size()` − returns the current number of elements in the list. [2 points]
- `void insert(type value, iterator position)`

adds an element at position specified by the iterator. For example, if the passed iterator currently points to the second element, this element will be shifted on position, and the new value should be added at the second position. [3 points]

- `iterator erase(iterator position)` – erases the element specified by the iterator and return an iterator to the next element, throws exception if position points after the last element. [3 points]
- `list<type>& operator = (list<type> another_list)` – overloads the assignment operator to deep copy a list into another list and return the current list by reference. [3 points]
- `iterator begin()` – returns an iterator pointing to the first element. [3 points]
- `iterator end()` – returns an iterator pointing after the last element. [3 points]
- You should develop an iterator class the following public interface:
    - `void operator ++ ()` – overloads the operator ++, it should advance the iterator one position towards the end of the list, throws exception if it is currently pointing after the last element. [3 points]
    - `void operator -- ()` – overloads the operator --, it should move the iterator one position toward the beginning of the list, throws exception if it is currently pointing to the first element of the list. [3 points]

- All node pointers in the list class of the iterator class should be private and inaccessible from outside of the class. [3 points]
- No memory leaks or dangling pointers. [3 points]
- It is highly recommended to implement it as a double linked list.
- **Write a main function to test all the above.**

Problem 2: Sorting [15 Points]

In this problem, we will develop classes to use for testing two sorting algorithms (Selection and Quick sort). The **TestSorting** class will have methods to support experimenting and analyzing sorting algorithms performance.

TestSorting class has the following functions that you should complete:

1. **GenerateRandomList(min, max, size)** Generate a given number of random integer data from a certain range. For example, one can generate a vector/array of 10000 integer numbers that fall in the range from 1 to 100000, e.g., [5554, 32300, 98000, 342, ...]
2. **RunOnce(sorter, data, size)** Run a given sorting algorithm (sorter is either a Selection or Quicksort) on a given set of data and calculate the time taken to sort the data.
1. **RunAndAverage(sorter, min, max, size, sets_num)** Run a given sorting algorithm on several sets of data of the same length and same attributes (from the same range) and calculate the average time.

Write a main function to test.

Problem 3: Stacks [15 Points]

In this problem, you should develop a stack class similar to that provided in the C++ STL. You can use arrays or your linked list class developed in the previous problem as an underlying data structure, however, you cannot use any of the C++ STL classes in this problem.

- Your stack class should be template. [3 Points]
- The stack class should have the following public interface:
    - `stack()` – default constructor. [2 Points}
    - `stack(type value, int intial_size)` constructs a stack having 'initial_size' elements whose values are 'value'. [3 points]
    - `~stack()` – a destructor to clear the stack and leave no memory leaks. [3 Points]
    - `type& top()` – returns the top element by reference. [3 Points]
    - `void pop()` – removes the top element. [3 Points]
    - `void push(type value)` – adds an element to the top of the stack. [3 Points]
    - `int size()` – returns the number of elements in the stack. [2 Points]
- **Write a main function to test all the above.**

Problem 4: Using stack [15 Points]

In this problem, use the stack implemented in question 3. Given an input string of brackets '(' and ')', square brackets '*' and '+', curly brackets ';' and '-', and multiple line comment token '/*' and '*/', check if this string is valid or not.
 A string is considered valid if and only if:

For each opened bracket, there should be a closing bracket of the same type.

Each closing bracket should close the lastly opened bracket.

Multiple line comment tokens consist of two characters, and any text between them should be ignored. However, they are treated the same way as brackets regarding the rules described above.

([{}]) () {} [] {[]} – valid.

({)} – invalid.

({/*)}]]]]]]]}*/}) – valid.

({/*[] [[]]]]]}) – invalid, the comment is not closed.

[{/*******/}] – valid.

Write a main function to test.

Problem 5 Priority Queue [15 Points]

Priority Queue is an extension of queue with following properties.

1. Every item has a priority associated with it.
2. An element with high priority is dequeued before an element with low priority.
3. If two elements have the same priority, they are served according to their order in the queue.

Implement:

- ```string enqueue(string value, int priority)```
- ```string dequeue() – dequeue according to the priority```
- **Write a main function to test.**

```
Example:
q.enqueue("world", 10)
q.enqueue("Hello", 5)
cout << q.dequeue() << " " << q.dequeue() <<endl;
Hello world
```

Rules:
- Cheating will be punished by giving -2 * assignment mark.
- Cheating is submitting code or report taken from any source that you did not fully write yourself (from the net, from a book, from a colleague, etc.)
- Giving your code to others is also considered cheating. Both the giver and the taker will get -10.

- People are encouraged to help others fix their codes but cannot give them their own code.
- Do not say we solved it together and we understand it. You can write the algorithm on paper together but each group should implement it alone.
- If you do not follow the delivery style (time and files names), your assignment will be rejected.