Cairo University
Faculty of Computers & Artificial Intelligent

CS214 – Data Structures – 2020
Assignment (2)

[130 Points]

Delivered Files Name:

- Assignment number with teams member IDs in zip file name A2_ID1_ID2_ID3_ID4.zip
- **Don’t** create folder inside the zip put code files directly
- **Don’t** create zip files inside the main zip file.
- Only include the .h files, .cpp files, .doc files and .txt files. **Don’t** include any other project files or exe files
- Problem code files name should be section number with problem number S#_P#.cpp
- **Any submission that doesn't follow the above rules will be ignored**

Example

- A2_2018203_2018204_2018205.zip
- Inside zip file
  - S1_P1.cpp
  - S1_P2.cpp
  - S1_P3.cpp
  - S2_P1.cpp
  - S2_P2.cpp
  - S2_P3.cpp
  - S2_P4.cpp

Section#1(90 point)

Problem 1: Linked lists [46 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 here
};
```

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]
    - type& operator * () – overloads the dereference operator to return the value contained in the current node by refence to allow its modification. [3 points]
    - bool operator == (const iterator &) – overloads the equality comparison operator, should return true if the passed operator points to the same node. [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, however, it is up to you.

As mentioned above, our .end() function should return an iterator pointing to a position after the last element as the STL .end() function does. This can be done easily by having a dummy node after the actual tail, this dummy node contains no data, but mark the end of the list. So, physically, our list is never empty, which will ease the implementation of insertion and remove operations, as now we don’t have to handle an “empty list” case. However, this dummy node should be disregarded when we return the size.

Write a main function to test all the above.

Problem 2: Stacks [22 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 3: Queues [22 Points]

In this problem, you should develop a queue 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 queue class should be template. [3 Points]
The queue class should have the following public interface:

- queue() – default constructor. [2 Points]
- queue(type value, int intial_size) – constructs a queue having ‘initial_size’ elements whose values are ‘value’. *3 points]
- ~queue() – a destructor to clear the queue and leave no memory leaks. [3 Points]

- type& front() – returns the first element by reference. [3 Points]
- void pop() – removes the first element. [3 Points]
- void push(type value) – adds an element to the back of the queue. [3 Points]
- int size() – returns the number of elements in the queue. [2 Points]

Write a main function to test all the above.

<u>Section#2 (40 point)</u>

Problem # 1 Convert Binary to Decimal [5 points]

Write a function to convert a number from base 2 to base 10. To convert a number from base 2 to base 10, we first find the weight of each bit in the binary number. The weight of each bit in the binary number is assigned from right to left. The weight of the rightmost bit is 0. The weight of the bit immediately to the left of the rightmost bit is 1, the weight of the bit immediately to the left of it is 2, and so on. Consider the binary number 1001101. The weight of each bit is as follows:
weight

```
        6543210
        1001101
```

We use the weight of each bit to find the equivalent decimal number. For each bit, we multiply the bit by 2 to the power of its weight, and then we add all of the numbers. For the binary number 1001101, the equivalent decimal number i

$1 * 2^6 + 0 * 2^5 + 0 * 2^4 + 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 =$
$64 + 0+0+8+4+0+1 =77$

To write a program that converts a binary number into the equivalent decimal number, we note two things:
(1) The weight of each bit in the binary number must be known
(2) the weight is assigned from right to left.

Because we do not know in advance how many bits are in the binary number, we must process the bits from right to left. After processing a bit, we can add 1 to its weight, giving the weight of the bit immediately to its left. Also, each bit must be extracted from the binary number and multiplied by 2 to the power of its weight. To extract a bit, you can use the mod operator. Write a

program that uses a **stack** to convert a binary number into an equivalent decimal number and test your function for the following
values: 11000101, 10101010, 11111111, 10000000, 1111100000.

Problem # 2 Balanced String [5 points]

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

Note that an empty string is also considered valid.

Input: exp = “{()}[][{()()}()]”

Output: True

Input: exp = “{(])”

Output: False

Problem#3 Sort an array according to another array [5 points]

Given two arrays A1[] and A2[], sort A1 in such a way that the relative order among the elements will be the same as those in A2. For the elements not present in A2, append them at last in sorted order.

Example:
Input: A1[] = {2, 1, 2, 5, 7, 1, 9, 3, 6, 8, 8}
A2[] = {2, 1, 8, 3}

Output: A1[] = {2, 2, 1, 1, 8, 8, 3, 5, 6, 7, 9}

Problem # 4 Sorted linked list [25 points]

You have a class called “**Student**” that contains 3 attributes: Name, ID, GPA.

Then Create a Singly linked list of class Student Called “**StudentList**”

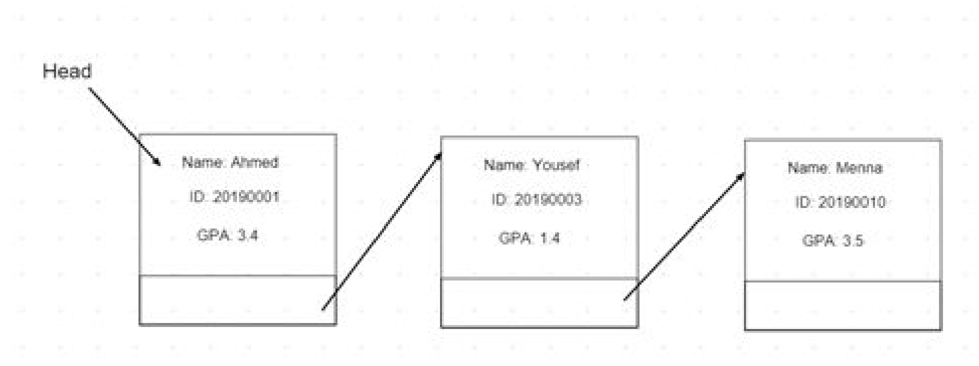
Following are the basic operations supported by a list.

- Insertion − Adds a student to the list. **While keeping the list sorted by ID**
- Delete − Deletes an element with the given ID
- Display − Displays the complete list.
- Search − Searches an element using the given key. And prints it.

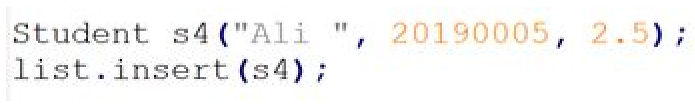
Example:

After executing the following code, Your list should look like:

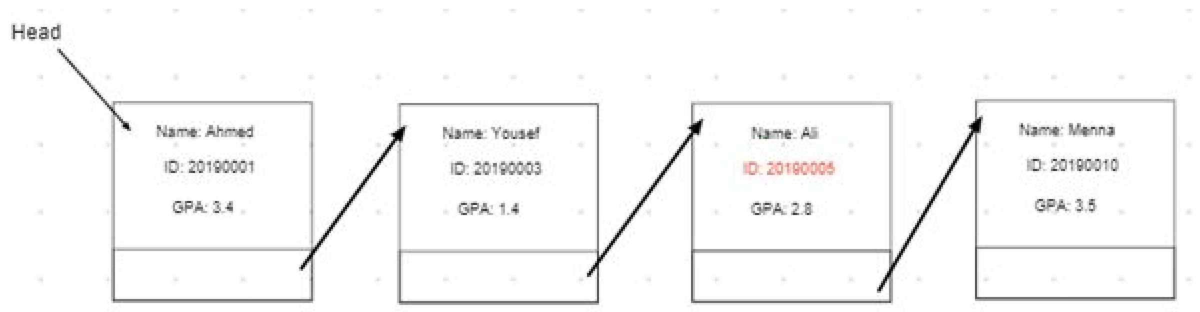
```
StudentList list;
Student s1("Ahmed", 20190001, 3.4);
Student s2("Menna ", 20190010, 3.5);
Student s3("Yousef ", 20190003, 1.4);

list.insert(s1);
list.insert(s2);
list.insert(s3);
```



Inserting a new Student should **keep the list ordered by ID**:



```
Student s4("Ali ", 20190005, 2.5);
list.insert(s4);
```


Head
Name: Ahmed
ID: 20190001
GPA: 3.4
Name: Yousef
ID: 20190003
GPA: 1.4
Name: Ali
ID: 20190005
GPA: 2.8
Name: Menna
ID: 20190010
GPA: 3.5