National University of Computer and Emerging Sciences, Lahore Campus

Course:	**Data Structures**	**Course:**	**CS 2001**
Program:	**BS(CS),BS(SE)**	**Semester:**	**Fall 2023**
Due Date	**15th-Oct-2023 at 11:59pm**	**Total Marks:**	**30**
Type:	**Assignment 2**	**Page(s):**	**3**

Important Instructions:

- **Submit all the questions separately, do NOT zip your file, submit your file with your rollnumber in cpp form (no header files needed, only one cpp file for each question).**

Question 1: **[10 Marks]**

Design a C++ program that uses the stack class for matching tags and quotes in XML extensive Markup Language. Your program will get an XML code in an input file, and it should figure out if tags and quotes are properly matched or not using stack. In case the tags are not matched, then your program should **report i) the error, ii) print the mismatched tag and iii) print the line number at which the starting tag occurred. And then continue parsing the input XML file till the XML file ends.**

What is XML? XML is a markup language somewhat like HTML. It was designed to store and transport data. XML is just information wrapped in user-defined tags which is both human- and machine-readable. The XML language has no predefined tags like HTML. The tags are "invented" by the author of the XML document. For details, see https://www.w3schools.com/xml/xml_whatis.asp

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <message>Don't forget me this weekend!</message>
</note>
```

In above example

<?xml version="1.0" encoding="UTF-8"**?>**

This is XML prolog (header). It starts with <? and ends with ?>. The header should come in the start of the document.

<note>, <from>, <heading> and <message> are user defined tags and each must have a corresponding ending tag.

Your program should handle the following features of XML:

1. **xml prolog (xml header)**
2. **xml tags (xml elements). The xml tags are case-sensitive.**
3. **xml attribute**
4. **xml comments, start with <!-- and ends with -->**

Consider another example

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
 <book category="cooking">
 <title lang="en">Everyday Italian</title>
 <author>Giada De Laurentiis</author>
 <year>2005</year>
 <price>30.00</price>
</book>
<book category="children">
 <title lang="en">Harry Potter</title>
 <author>J K. Rowling</author>
 <year>2005</year>
 <price>29.99</price>
</book>
</bookstore>
```

In the example above:

<title>, <author>, <year>, and <price> have **text content** because they contain text (like 2005).
<bookstore> and <book> have **element contents**, because they contain elements.
<book> has an **attribute** (category="children").

Note that Attribute values must always be quoted. Either single or double quotes can be used.

Your program should keep track that the attributes have opening and closing quotes.

Your code will have a template-based Node class Stack class. Implement stack using singly linked list.

```
template<class T>
class Stack {
private:

class Node {
       T  data;
       Node<T> * next;
      Node<T> * top;

 };
public:
      Stack() ;
      ~Stack();
      bool IsEmpty();
      bool push(const T & val);
      bool pop(T & val);
};    bool top(T & val);
```

Create XMLData class
Think about its attributes carefully

```
void main(){
Stack<XMLData> S1;
```

Note: (Not Mandatory , it's your choice)

You can also made simple functions in XMLData class and call them in main() as:

```
XMLData d1;
d1.checkAttribute();
```

BUT for that case your XML class must inherit your Stack class.

Question 2: ('Invertible' Stack) **[10 marks]**

Add the following methodology to the Stack Class (You must create your own Stack Class).

Add a method called flipStack. This method should work in O(1). Its effect should be such that the whole stack should be logically inverted, i.e. the oldest element becomes the newest and vice versa. So the next pop will remove the element that was at the bottom of the stack before it had been flipped. Notice that the stack may be flipped again and again by using the flipStack method repeatedly. Make sure that no slot of the array is wasted (Hint: Circular). As always, when the stack fills up we double its capacity; and when more than half of it is empty, we halve the capacity. Once again, test your code extensively.

Question 3: ('Undo' Stack) **[10 marks]**

Create a class called UndoStack. This stack is intended to be used for the undo operation in another application. It should have the property that it remembers at most the last 100 elements pushed into it. So, for example, if there were 101 push operations it would "forget" the oldest of these and if a pop happens the 2nd pushed element will be popped, etc. Write complete C++ code for this class.

CODE DESIGN GUIDLEINES

- Do template-based programming
- Code should be properly indented and commented (2 marks for this)
- Make sure there are no memory leaks or dangling pointers
- Don't cheat or take too much unnecessary help from your friends as I will check cross plagiarism.