**Data Structures and Algorithms**

**ASSIGNMENT 3**

**Spring 2024**

**PROBLEM**

**Stack of link lists**

Develop a Stack which can store multiple link lists with unique ids.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Id=1  Sum = 111  Status = 0  Data:   |  | | --- | | 1 | | 1 | | 2 | | 3 | | 3 | | 5  φ | | 6 | | 90 | |  | Id =3  Sum =95  Status = 1  Data:   |  | | --- | | 88 | | 6 | | 1 | |  | Id =5  Sum =12  Status = 0  Data:   |  | | --- | | 1 | | 1 | | 2 | | 3 | | 5 | |  | Id =6  Sum =0  Status = 0  Data:   |  | | --- | | root = NULL | |
| Previous |  | Previous |  | Previous |  | Previous |

top

Each node of this stack contains a unique ***id***, integer variable ***Sum*** & a link list implemented integer sorted list (***Data***) as shown above. Sum is an aggregate of all integer values in the list. You have to compute this sum with the insertion/deletion of an element in the sorted list.

Link list contains boolean variable ***Status*** apart from the pointer ***root***. Status is 0 if the list is sorted in ascending order & 1 if it is sorted in descending.

* 1. User can sort the link list of the top of the stack (TOS) element in ascending/ descending order at any time.
  2. User can push/pop an element in the stack. Make sure ids are unique in the list. If user tries to enter a node with already existing id, it should not be pushed. User will give the id of the node to be pushed/ popped.
  3. User can also insert/delete a complete link list in a top of the stack element. If list is not empty then new list should be concatenated with already existing list.

For example

If the top of the stack is an element with id 3 & we want to concatenate the following list

|  |
| --- |
| 0 |
| 1 |
| 2 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Id =3  Sum =95  Data:   |  | | --- | | 88 | | 6 | | 1 | | | Previous | | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Id =3  Sum =98  Data:   |  | | --- | | 88 | | 6 | | 2 | | 1 | | 1 | | 0 | | | Previous | |
| Before insertion | After insertion |

* 1. Add/drop an element from the TOS list (in that case, make sure you update the data member Sum accordingly).

Make appropriate functions of printing, isfull, isempty, constructors & destructors. There should be a proper menu.

**Question: Binary Expression Tree**

You are tasked with implementing a binary expression tree to efficiently evaluate mathematical expressions. Your task involves constructing a binary expression tree from a given postfix expression and then evaluating it.

**Binary Expression Tree Construction:**

1. Implement a function/method to construct a binary expression tree from a given postfix expression.
2. Assume that the input postfix expression is provided as a string, where operands and operators are separated by spaces.
3. Your function/method should handle the following operators: addition (+), subtraction (-), multiplication (\*), and division (/).
4. Ensure that the constructed binary expression tree reflects the correct order of operations.
5. Generate infix (includes braces to show precedence) and prefix expression

**Binary Expression Tree Evaluation:**

1. Implement a function/method to evaluate the mathematical expression represented by the constructed binary expression tree.
2. Traverse the binary expression tree to perform the necessary arithmetic operations and return the result.
3. Handle cases where division by zero occurs gracefully.

**Sample Input and Output:**

Consider the following postfix expression: **"3 4 + 2 \*"**

* + Construct the binary expression tree.
  + Evaluate the expression using the constructed binary expression tree.

**Test Cases:** Provide additional test cases to validate the correctness of your implementation. Test cases should include various arithmetic expressions involving different operators and operands.

**Efficiency Consideration:** Discuss the efficiency of constructing and evaluating the binary expression tree. Analyze the time complexity of both operations and propose any optimizations if applicable.

**Code Submission:** Submit your implementation with well-commented code and test cases demonstrating its functionality.

**VERY IMPORTANT**

* Academic integrity is expected of all the students. Plagiarism or cheating in any assessment will result in negative marking or an **F** grade in the course, and possibly more severe penalties.