NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CX1107 Data Structures and Algorithms **2020/21 Semester 2**

Assignment 5: Transform and Conquer

School of Computer Science and Engineering *Nanyang Technological University*

Q1 Given a character string of integer arithmetic expression, write a function, expressionLL() to split the operands and operators into a linked list. We assumed that the character string only contains digits, operators ('+', '-', '*' and '/') and parentheses.

The structure of LinkedList is given below. type is used to indicate the item is an operator or an operand

```
enum ExpType {OPT,OPERAND}; //OPT: OPERATOR

typedef struct _listnode
{
    int               item;
    enum ExpType      type;
    struct _listnode *next;
} ListNode;

typedef struct _linkedlist{
   int       size;
   ListNode *head;
} LinkedList;
```

The following utility functions are provided. The details can refer to the given template:

```
void insertNode(LinkedList *llPtr, int item, enum ExpType type);
int deleteNode(LinkedList *llPtr);
void removeAllNodes(LinkedList *llPtr);
int isEmptyLinkedList (LinkedList ll);
```

The function prototype is given as follows:

```
void expressionLL(char* infix, LinkedListExp* inExpLL);
```

For verification purpose, the printExpLL will print all operands after adding a seed number.

For example

Input:

1+11*111/(1111-88)
7 (seed number)

Output:
8 + 18 * 118 / (1118 - 95)

Q2 Write a function, in2preLL() to covert an infix expression into a prefix expression linked list. We assumed that the character string only contains positive integer numbers, operators ('+', '-', '*' and '/') and parentheses.

You may need the following stack structure:

```
enum ExpType {OPT,OPERAND};

typedef struct _stackNode{
    char item;
    struct _stackNode *next;
}StackNode;
```

```
typedef struct _stack{
   int size;
   StackNode *head;
}Stack;

typedef struct _listnode
{
    int  item;
    enum ExpType type;
  struct _listnode *next;
} ListNode;

typedef struct _linkedlist{
   int size;
   ListNode *head;
} LinkedList;
```

The following utility functions are provided. The details can refer to the given template:

```
void push(Stack *sPtr, char item);
int pop(Stack *sPtr);
char peek(Stack s);
int isEmptyStack(Stack s);

void insertNode(LinkedList *llPtr, int item,enum ExpType type);
int deleteNode(LinkedList *llPtr);
void removeAllNodes(LinkedList *llPtr);
int isEmptyLinkedList (LinkedList ll);
```

The function prototype is given as follows:

```
void in2preLL(LinkedListExp* ExpLL);
```

For example:
Input: 99+(88-77)*(66/(55-44)+33)
Output: + 99 * - 88 77 + / 66 - 55 44 33

Q3 Write a function, createExptree() to create an expression tree structure by using a prefix expression. Each element (operands, operators, parenthese) in the prefix expression is seperated by a white space.

You may need the following tree structure:

```
typedef struct _btnode{
  int item;
  struct _btnode *left;
  struct _btnode *right;
} BTNode;
```

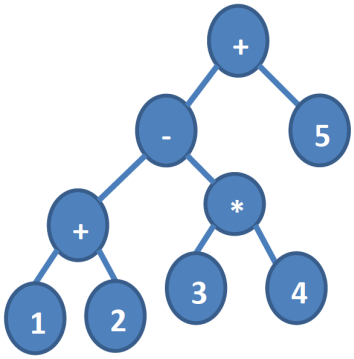
Next, the function will insert each operator in the given expression into the tree as an internal node and each operand is inserted as a leaf node. See example below.

Write another two functions, printTree() and printTreePostfix(), to print all tree nodes by in-order and post-order traversal, respectively. For your information, the in-order traversal result should be same as the original infix expression. The post-order traversal will move operator "to the back". It is known as postfix notation or Reverse Polish notation invented by Jan Łukasiewicz in 1924. For example, infix expression: a+b*c-d ⇒ postfix notation: a b c * + d -.

Lastly, write an expression evaluation function, computeTree() to calculate the given arithmetic expression. The function prototypes are given as follows:

```
void createExpTree(BTNode** root,char* prefix);
void printTree(BTNode *node);
void printTreePostfix(BTNode *node);
double computeTree(BTNode *node);
```

Example: “1+2 -3 *4 +5” → “+ - + 1 2 * 3 4 5”



Example: Input:

\+ 99 * - 88 77 + / 66 - 55 44 33

Output:

99 + 88 - 77 * 66 / 55 - 44 + 33
99 88 77 - 66 55 44 - / 33 + * +
528.00

where the first row is the infix expression (the parentheses are ignored here), the second row is the postfix expression and the last row is the evaluation answer.