Solutions to Assessment

1. Let S1 and S2 be two instances of Stack ADT. S1.push(x) and S2.push(x) are operations that push an integer element to the stacks S1 and S2 respectively. S1.pop() and S2.pop() are operations that remove an integer element from the stacks S1 and S2 respectively. S1.empty() and S2.empty() are functions which return true if the the corresponding stack is empty and false otherwise. Let Y be an ADT with two operations Y.op1(x) that adds an element x to Y and Y.op2() that removes an element from Y and returns it. Assume S1 and S2 are initially empty. Consider the following segment of code which implements the functions op1(x) and op2().

```
void op1(int x){
      S1.push(x);
}
int op2(){
      while(!S1.empty()){
            x=S1.pop ();
            S2.push(x);
      }
      x=S2.pop();
      while(!S2.empty()){
            x=S2.pop ();
            S1.push(x);
      }

}
```

Specify which one of the following statements is correct.
A)Y represents a Queue ADT where op1(x) represents an enqueue operation and op2() represents a dequeue operation.
B)Y represents a Stack ADT where op1(x) represents an push operation and op2() represents a pop operation.
C)Y represents neither a Stack ADT nor a Queue ADT.
D)None of the above.

Ans: A
Explanation: It uses two stacks S1 and S2 to simulate First-In-First-Out feature of queues. op1(x) pushes an element into S1 just like an insert is made into a queue. op2() pops elements out of S1 till it reaches the first element, all the while pushing the popped-out elements into another stack S2. Once the first element is popped out, all the remaining elements are once again pushed back from S2 into S1.

2. Let S represent an instance of the Stack ADT. Let S.push(x) push the value x on to the top of the stack, S.pop() remove the topmost element from the stack and return the value. Consider the following sequence of operations performed on S which initially contains 10 elements with 55 as the top most element (Assume that S is of sufficient capacity).

```
S.push(7);
S.push(20);
S.push(35);
S.pop();
S.push(14);
S.pop();
S.pop();
```

What will be the element at the top of the stack after the above sequence of operations?

Ans: 7

Explanation:

The topmost element after each step is displayed in bold:

```
S.push(7);      // 55, 7
S.push(20);     // 55, 7, 20
S.push(35);     // 55, 7, 20, 35
S.pop();        // 55, 7, 20
S.push(14);     // 55, 7, 20, 14
S.pop();        // 55, 7, 20
S.pop();        // 55, 7
```

3. Consider the method for conversion of an infix expression to its postfix form using stack as discussed in lecture. What will be the second last operator that gets popped off the stack by the end of conversion of the following expression using this method?
(a+b)*((c+d)/(e-f))

Ans: /

Explanation:

We use **Operator stack** to convert infix expression to postfix

	Current Symbol	Operator Stack	Postfix Expression
1.	(	(	
2.	a	(	a
3.	+	(+	a
4.	b	(+	ab

{when current symbol is "**)**", symbols "+", "(" are popped from operator stack and symbol "+" is added to postfix string}

5.	)		ab+

6.	*	*	ab+
7.	(	***(**	ab+
8.	(	***((**	ab+
9.	c	*((	ab+c
10.	+	*((+	ab+c
11.	d	*((+	ab+cd
12.	**)**	***(**	**ab+cd+** {pop “+”, add to postfix}
13.	/	*(***/**	ab+cd+
13.	(	*(/**(**	ab+cd+
13.	e	*(/(	ab+cd+e
14.	-	*(/(**-**	ab+cd+e
15.	f	*(/(-	ab+cd+ef
16.	**)**	***(/**	**ab+cd+ef-** {pop “-”, add to postfix}
17.	)	*	ab+cd+ef-/ {pop “/”, add to postfix}
18	End		ab+cd+ef-/* {pop “*”, add to postfix}

The second last operator that was popped from operator stack is “***/***”.

4. A new ADT SuperList is defined as follows:

```
class SuperList{

        List *l;
        public:
                DeleteSomewhere() {
                        l.delItem(l.first());
                }
                InsertSomeWhere(ELemType x) {
                        l.insert(l.first(), x);

                }
                int ShowEmpty () {
                        return l.empty()
                }
}
```

These operations on the SuperList make the SuperList behave like:
A)Stack
B)Queue
C)Stack and Queue
D)None of the above

Ans: A)Stack
Explanation: Both insert and delete operations are performed on the first element. This happens in implementation of **Stack**.

5. Let Q be an instance of a Queue ADT. Q.enqueue(x) adds an element x to the queue. Q.dequeue() performs a dequeue operation on the queue and returns the value that gets dequeued. Consider the following segment of code:

```
Q.enqueue(1);
int count=1;
do {
      count=count+1;
      x = Q.dequeue();
      Q.enqueue( 2*x );
      Q.enqueue( 4*x );
} while(x != 32);
```

What will be the value of the variable count, when the above segment of code completes its execution?

Ans: 12
Explanation:
Contents of queue will be: 1, 2, 4, 4, 8, 8, 16, 8, 16, 16, 32, 16, 32.. etc.

6. Let Q be an instance of a Queue ADT. Q.enqueue(x) adds an element x to the queue. Q.dequeue() performs a dequeue operation on the queue and returns the value that gets dequeued. Consider the following segment of code:

```
Q.enqueue(1);
int count=1;
do {
      count=count+1;
      x = Q.dequeue();
      Q.enqueue( 2*x );
      Q.enqueue( 2*x+1);
}while(x != 967);
```

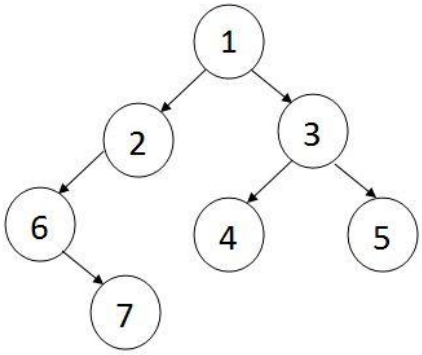
What will be the value of the variable count, when the above segment of code completes its execution?

Ans: 968
Explanation:
Contents of queue will be: 1, 2, 3, 4, 5, 6, 7, 8……

7. A tree is described as follows.
1 is root.
2,3 are the left and right children of 1 respectively.
4,5 are the left and right children of 3 respectively.
6 is the left child of 2.

7 is the right child of 6.
What is the height of the above tree ?

Ans: 3
Explanation:



8. The grand parent of 7 in the following tree is:
1 is root.
2,3 are the left and right children of 1 respectively.
4,5 are the left and right children of 3 respectively.
6 is the left child of 2.
7 is the right child of 6.

Ans: 2

9. It is possible to create a unique tree with preorder and postorder traversals.

- T
- F

Ans: F
Explanation: It is not possible to build a unique binary tree from inorder and postorder traversals. Only with the the help of inorder and one of the pre/post order traversals , a unique tree can be formed.
Example:

In the above picture both trees have same pre order and post order traversals, even

though their tree structure is not same.
(See the video lecture on tree traversal for more information.)

10. What will be the output of the following function run on a BST constructed by inserting nodes in the following sequence 5,3,1,7,6,8 ?

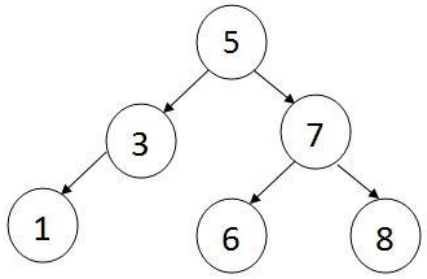
```
int fun(struct BST* root){

    if(root == NULL)
      return INT_MIN;

    while(root->left != NULL)
      root = root->left;

    return root->data;
}
```

Ans: 1
Explanation: The function traverses the BST till it reaches the leftmost leaf node of the tree shown in figure below:



11. Elements in ascending order can be obtained using a particular traversal in a Binary Search Tree.
A)Preorder
B)Inorder
C)Postorder

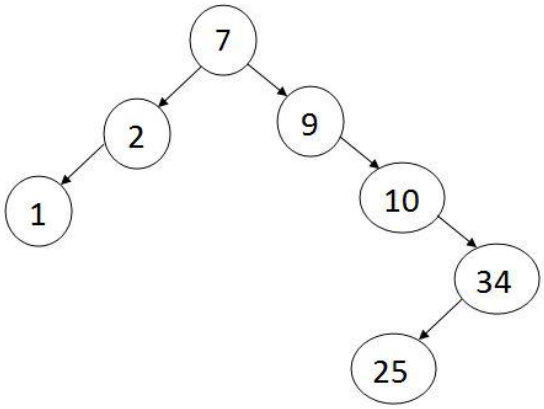
Ans: Inorder
Ascending(Or descending) order can be obtained by using Inorder(or reverse inorder) traversal in a binary search tree.

12. The preorder traversal of a BST is 7,2,1,9,10,34,25. What is the 5th element in a Post traversal after deleting 7 (assuming that 7 is replaced by its inorder-successor)?

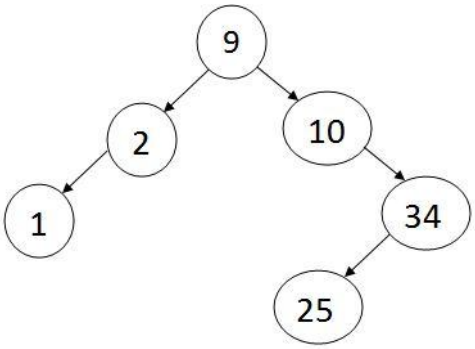
Ans: 10

Explanation:

Original tree:



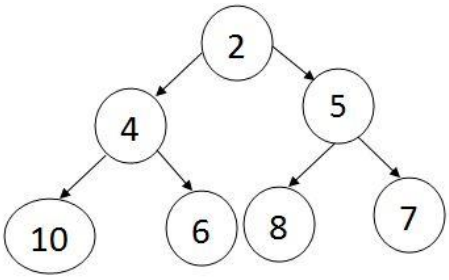
New tree after deleting 7:



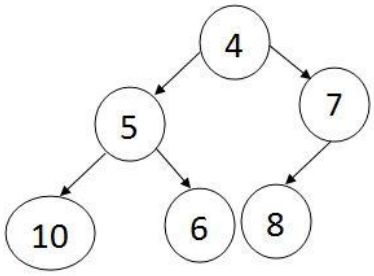
Post-order traversal of the new tree: 1, 2, 25, 34, 10, 9

13. Create a min heap with 10,8,6,2,4,5,7. The 4th element stored in the array after a "delete min" operation followed by the insertion of 1,9 is?

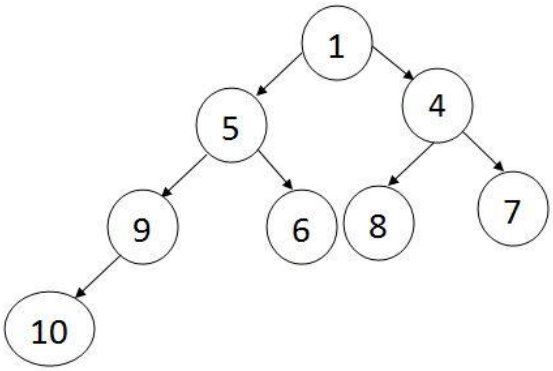
Ans: 9
Explanation:
Min heap after creation:



After ‘delete min’:



After inserting 1, 9:



Elements in the array: 1, 5, 4, 9, 6, 8, 7, 10

14. What is the maximum number of edges possible in a directed acyclic graph with 4 vertices ?
Ans: 6

15. What is the maximum number of non loop edges that can exist in an undirected graph of 10 vertices ?
(*definition:* loop is an edge connecting a vertex with itself)

Ans: $^{10}C_2$
Explanation: Each edge need 2 vertices (no loop-edges). So with total of 10 vertices, 2 vertices can be picked $^{10}C_2 = 45$ ways, which is equal to the number of edges.

Solutions to Programming Assignments

Question 1: Queue in a Hospital

In the outpatient department of an hospital, where there is a single doctor for consultation, patients are issued tokens in increasing order of their arrival. The token also carries a number indicating the severity of the case of a patient. For example, if a patient with a cold and cough comes at 9.55AM, he gets a token number 10 and priority number 3 whereas when a patient with a heart attack comes at 10 AM, he gets a token number 11 and priority number 1. The hospital staff routes the patient with highest priority number to the doctor's queue using a procedure described below:

1. There are 3 priority numbers depending on severity - 3 for casual cases, 2 for moderate cases and 1 for severe cases.
2. For every 1 patient in casual queue, 2 patients from moderate queue and 3 from severe queue are sent to the doctor's queue. Let C indicate casual queue, M the moderate queue and S the severe queue. If there are 9 patients in C, 10 in M and 9 in S, then the doctor's queue would be: 3S-2M-1C-3S-2M-1C-3S-2M-1C-2M-1C-2M-5C

Within the same queue, they can be sent in the order in which the input has been given. i.e, if token 10 and token 21 both have priority 3, then 10 will be sent first and 21 next. You do NOT have to sort based on token number for the same priority(The input will be provided in increasing order of token numbers).

Using Queue ADT provided to you, you have to develop a program that takes as input, token numbers and corresponding priority number of a set of 10 patients and output the doctor's queue based on the procedure described above. Your output has to list the token numbers as is in the doctor's queue.

INPUT:
20 integers - the first integer would be token number, the second integer would be priority number, the third would again be token number and the fourth would be corresponding priority number and so on.

OUTPUT:
List of token numbers as would be in the final doctor's queue each separated by a single space. The last token number should also be followed by a space (so as not to have a separate "cout" for the last entry).

Test Cases:

Public test cases:

Input	Output
10 3 12 2 13 1 14 2 15 2 16 1 17 3 18 2 19 3 20 1	13 16 20 12 14 10 15 18 17 19
101 1 106 2 107 3 108 1 109 2 110 3 111 1 112 2 113 3 114 1	101 108 111 106 109 107 114 112 110 113
200 3 201 3 202 3 203 3 204 3 205 3 206 2 207 2 208 2 209 2	206 207 200 208 209 201 202 203 204 205
1 3 2 2 3 1 4 3 5 2 6 1 7 3 8 2 9 1 10 3	3 6 9 2 5 1 8 4 7 10
010 2 011 2 012 2 013 2 014 2 015 3	18 19 10 11 15 12 13 16 14 17

016 3 017 3 018 1 019 1	

Private test cases:

Input	Output
11 1 22 2 34 3 41 2 52 2 56 2 87 3 90 1 92 2 99 1	11 90 99 22 41 34 52 56 87 92
82 3 83 3 84 2 85 2 86 1 87 1 88 3 89 2 90 1 91 2	86 87 90 84 85 82 89 91 83 88
33 3 41 1 52 2 62 2 70 1 84 3 88 3 91 1 92 1 99 1	41 70 91 52 62 33 92 99 84 88
100 1 200 3 300 2 400 2 500 3 600 1 700 1 800 2 900 2 1000 2	100 600 700 300 400 200 800 900 500 1000
12 1 13 1	12 13 14 15 16 17 18 19 20 21

14 1 15 1 16 1 17 1 18 1 19 1 20 1 21 1	

Solution:

```
#include <iostream>
using namespace std;

struct Node{
	int info;
	  Node *next;
};

class QueueADT{
	  private:
		Node *qFront;
		Node *qRear;

	  public:
		QueueADT(){ //constructor
			qFront = 0;
			qRear = 0;
		}
		bool isEmpty(){
			if (qFront==NULL) return(true);
			else return(false);
		}
		int dequeue(){
			int data;
			data = qFront->info;
			qFront = qFront->next;
			if (qFront==0) qRear = 0;
			return data;
		}
		void enqueue(int data){
```

```
                Node *temp = new Node();
                temp->info = data;
                temp->next = NULL;

                if (qFront == 0) {
                        qFront = qRear = temp;
                }
                else{
                        qRear->next = temp;
                        qRear = temp;
                 }
        }

};

int main(){
    int i, count, token, priority;
    QueueADT q1, q2, q3, q4;

    for (i=0;i<10;i++){
          cin>>token>>priority;
          if (priority==1){
                  q1.enqueue(token);
          }
          else if(priority==2){
                  q2.enqueue(token);
          }
          else if(priority==3){
                  q3.enqueue(token);
          }

         }
         i=0;

    while(i<10){
          count = 0;
          while (!q1.isEmpty() and count<3){
                  q4.enqueue(q1.dequeue());
                  count++;
                  i++;
          }
```

```
        count = 0;
        while (!q2.isEmpty() and count<2){
                q4.enqueue(q2.dequeue());
                count++;
                i++;
        }
        if (!q3.isEmpty()){
                q4.enqueue(q3.dequeue());
                i++;
        }
    }

/* DO NOT edit the code below. It will be appended during compilation

    for (i=0;i<10;i++)
        cout<<q4.dequeue()<<" ";
}
*/
```

Question 2:

Write a Program to implement Binary Search tree as discussed in the Lecture.

The following methods are to be implemented in the Binary Search Tree.

1)insert(key), function which inserts a new element into the tree.
2)remove(key), function which removes the key from the Binary Search Tree
follow these conditions to while removing the element.

i)if the tree-node having key is a leaf node, delete that node.
ii)replace the desired tree node with its child, if that node has only one child.
iii)If the tree node having key, has two non-empty children then use its
inorder - predecessor to replace the node. Inorder - predecessor for a tree node is the right most node in its left subtree.

3)print(), print the elements of the tree in postorder, separate one element from another with a single whitespace.

You need to build a binary search tree by inserting elements in the given order.
Then perform insert and remove operations on that tree based on the operation specified as a character (i - *insert*, d - *remove).*

Input:
*1st line has number of elements to be added to the Binary Search Tree (*Initial number elements* - N)
*2nd line of the input contains the elements each separated by single whitespace.
*3rd line has number of operations to be performed (k)
*4th line has operation, key separated by a white space (i - *insert*, d - *remove*)

Sample Input:
4
8 7 15 2
2
i 10 d 2

Output:
The program should output the **post order traversal** of BST after all the given operations are performed. Separate the elements with a single whitespace. Print “NULL”(without quotes), if the tree is empty.

Output for the above Sample Input:
7 10 15 8

Constraints:
Elements in the binary Search Tree are unique.
0<=N≤ 20
0<=key<=10000 (*Elements in Tree*)

0<=k<=10

Test Cases:

Public test cases:

Input	Output
3 8 4 10 1 d 8	10 4
4 25 12 83 5 2 i 15 d 83	5 15 12 25
5 21 16 78 1 22 2 i 20 d 16	20 1 22 78 21
2 9 12 4 i 3 d 12 d 9 d 3	NULL

Private test cases:

Input	Output
9 200 220 380 156 25 89 12 23 181 2 d 200 i 190	23 12 89 25 156 190 380 220 181
4 25 2 83 70 2 i 0 d 83	0 2 70 25
6 21 16 78 22 15 11 3 d 78 d 101 d 21	11 15 22 16
6 1 2 3 4 5 6 2 d 3 d 6	5 4 2 1
1 7 1 d 7	NULL
5 7 5 4 3 2	2 3 4 5 14 8 7

4 i 8 i 13 i 14 d 13	
10 35 200 20 18 30 25 34 100 222 210 10 i 19 d 20 i 56 i 134 d 222 i 36 d 18 d 25 i 44 i 50	34 30 19 50 44 36 56 134 100 210 200 35

Solution:

Comment [1]: solution here http://ideone.com/xkXx5q just in case(g-doc has copy,paste issues)

```
#include <iostream>

using namespace std;

class binarySearchTree;
class treeNode{
   /*
   *with the following friend declaration
   *binarySearchTree can access private members of treeNode
   */
   friend class binarySearchTree;
   private:
          int data;
          treeNode * left;
          treeNode * right;
          //treeNode * parent;

   public:
          treeNode(){
                 left=right=NULL;
                 //parent = NULL;
          }
          treeNode(int key){
                 data=key;
                 left=right=NULL    ;
                 //parent=NULL;
          }
          int getData(){ return data;}
          void setData(int key){data=key;}
};

class binarySearchTree{
   private:
          treeNode * root;
   public:
          binarySearchTree(){root=NULL;}

          /************************************
```

```
*inserts key in tree
*returns NULL, if the key already exists
*After the element is created it needs to be assigned as
*a left/right child to its parent.
*flag: is maintained to check which side to insert it on
*    flag= (1=>left) (2=>right) (0=>already existing key)
*prev: is maintained to for storing the parent of current node.
*/
treeNode * insert(int key){
        treeNode * temp, *prev;
        temp = root;
        prev = root;

        treeNode * new_node ;
        new_node = new treeNode(key);
        int flag=0;
        if(temp==NULL){
                //tree is Empty
                root = new_node;
        }
        while(temp!=NULL){

                if(key < temp->data){
                        prev = temp;
                        temp = temp->left;
                        flag=1;
                }
                else if(key > temp->data){
                        prev = temp;
                        temp = temp->right;
                        flag=2;
                }
                else{
                        //key already exists
                        cout<<"key already exists\n";
                        delete new_node;
                        return NULL;
                }
        }
        //now we assign new_node's parent
        if(flag==1) prev->left = new_node;
        else if(flag==2) prev->right = new_node;

        return new_node;
}
/*function to remove the key element in root node
**if one of the children is NULL, replace root with other child
**if both are NULL, point root to NULL
```

```
**If both left and right sub trees are not null,
**replace the root data with, its inorder-predecessor.
**Then delete the inorder predecessor.
**/
int removeRoot(){
	treeNode * temp = root;
	if(temp==NULL) return -1;
	if(temp->left==NULL){
		root = temp->right;
		delete temp;
		return 1;
	}
	if(temp->right==NULL){
		root = temp->left;
		delete temp;
		return 1;
	}
	else{
		//find the inorder-predecessor
		//right most child in the left subtree.
		treeNode * pred, *predPar;
		predPar = temp;
		pred = temp->left;
		while(pred->right!=NULL){
			predPar = pred;
			pred = pred->right;
		}
		//replace temp's data with predecessor's data
		//delete predecessor
		temp->data = pred->data;
		if(predPar->left == pred)
			predPar->left = pred->left;
		else
			predPar->right = pred->left;
		delete pred;
		return 1;
	}

}

/*************************
*return 1 on success, else returns 0 if not found, -1 if empty
*Two ways to replace the deleted node
*1)inorder successor 2)Inorder-predecessor
*we are asked for inorder-predecessor
*************************/
int remove(int key){
	treeNode * temp,*prev;
```

```
temp=root;
prev=NULL;
int flag=0;
if(root==NULL) return -1;
while(temp!=NULL){
	if(key < temp->data){
		prev=temp;
		temp = temp->left;
		flag=1;
	}
	else if(key > temp->data){
		prev=temp;
		temp = temp->right;
		flag=2;
	}
	else{
		//found the key at temp
		break;
	}
}
if(temp==NULL){
	//key not found
	return -1;
}
if(temp==root){
	//removing root
	return removeRoot();
}
// deleting leaves or internal nodes
else{
	if(temp->left==NULL){
		if(flag==1)
			prev->left = temp->right;
		else if(flag==2)
			prev->right = temp->right;
		delete temp;
		return 1;
	}
	if(temp->right==NULL){
		if(flag==1)
			prev->left = temp->left;
		else if(flag==2)
			prev->right = temp->left;
		delete temp;
		return 1;
	}
	//both children are not null
	else{
		//finding the predecessor
```

```
                                treeNode * pred, *predPar;
                                predPar = temp;
                                pred = temp->left;
                                while(pred->right!=NULL){
                                        predPar = pred;
                                        pred = pred->right;
                                }
                                //replace temp's data with predecessor's data
                                //delete predecessor
                                temp->data = pred->data;
                                if(predPar->left == pred)
                                        predPar->left = pred->left;
                                else
                                        predPar->right = pred->left;
                                delete pred;
                                return 1;
                        }

                }
        }

        void printPostOrder(treeNode * head){
                if(head==NULL) return;
                printPostOrder(head->left);
                printPostOrder(head->right);
                cout<<head->data<<" ";
        }
        /**As asked in the question**/
        void print(){
                if(root==NULL){
                        cout<<"NULL";
                        return;
                }
                printPostOrder(root->left);
                printPostOrder(root->right);
                cout<<root->data;
        }
};

int main(){

    binarySearchTree bst1;
    int N,val,NOp;
    char op;
    cin>>N;
    while(N--){
            cin>>val;
            bst1.insert(val);
```

```
    }
    cin>>NOp;
    while(NOp--){
            cin>>op;
            cin>>val;
            if(op == 'i'){
                    bst1.insert(val);
            }
            else if (op=='d'){
                    bst1.remove(val);
            }
    }

    bst1.print();
    return 0;
}
```

Question 3:

Write a Program to Evaluate a given Postfix Expression. A Postfix Expression is one in which every operator follows its operands.
For example the Postfix Expression for the Infix expression a*(b-c) is abc-*. Here a, b and c are called operands, and * ,- are called operators each representing a mathematical operation.

Use a **stack** to evaluate postfix expressions. Follow the below Algorithm as discussed in class to evaluate an expression given in postfix notation.

- Scan the Postfix Expression from left to right.
- If you encounter an operand put it into the stack. If it is an operator, calculate the value of expression with the operator and top two elements of the stack in correct order. Remove those top two elements from stack and add their result into the stack.
- Do the above step till you scanned all the elements of the postfix expression. You will be left with result of the given postfix expression.

The above algorithm can also be used to check if a given postfix expression is valid or not. For example the expressions abc-*d , a+bc- are invalid postfix expressions. Consider the expression containing division by 0 also as **invalid**.

Note that all operators are binary operators. (operator '-' can also be used as unary to represent -ve numbers, but in our postfix expression we only consider '-' as binary operator)

You will be provided with a simple Stack Implementation with methods that are needed for the task. You can use your own implementation of stack if required.

INPUT:
first line of input contains the total number of elements(operators and operands) in the postfix expression.(N)
Postfix expression will be in the second line, with elements separated from one another by a whitespace character.
Sample Input:
12 13 9 - *

OUTPUT:
Print the result of the Postfix Expression, If it is valid. Else print "**INVALID**" as output.
Sample Output:
48

CONSTRAINTS:
All operands are non-negative integers.
Allowed Operators are {*(multiplication), /(division) , +(addition), -(subtraction) } **.** All are left-associative by nature.

1 ≤ Number of Operators + Number of Operands ≤ 25
0≤ Input Operands≤ 10000

Hints:
Use *atoi(const char *)* to convert an array of characters to integer.

Test Cases:

Public Test Cases:

Input	Output
6 2 3 8 * 5 /	INVALID
11 1 2 + 3 * 6 + 2 3 + /	3
11 3 5 * 2 / 11 3 / 7 * +	28
9 6 2 7 + / 3 4 - *	0

Private Test Cases:

Input	Output
5 2 + 3 4 *	INVALID
11 12 13 - 20 6 / * 4444 4445 - /	3
5 45 24 12 * -	-243
9 45 250 * 16 3 / 4 / /	11250
11 4 2 5 - * 6 3 / 2 / /	-12
17 244 5 49 * - 67 20 / * 45 9 / 66 72 - + +	-4
15 65 64 - 4 5 - / 77 73 - * 110 20 / *	-20
4 2 3 4 +	INVALID

Solution:

Comment [2]: http://ideone.com/aJkDpp for proper indentation (code - c++)

Algorithm: We follow these steps to evaluate postfix expression

- Scan the postfix expression from left to right.
- Initialise an empty stack.
- If the scanned string is an operand, add it to the stack. If the scanned string is an operator, there must be at least two operands in the stack.
    - If the scanned character is an Operator, then we store the top most element of the stack in a variable **v2**. Pop the stack. The second most top element as **v1**. Now evaluate **v1 Operator v2**. Push result into the stack.
- Repeat this step till the given postfix expression is completely scanned.
- After all strings(operators & operators) are scanned, only one element must remain in the stack. Otherwise the given expression is invalid.

Code:

```
#include<iostream>
#include <stdlib.h>
using namespace std;

class IntStack {
   private:
        int *Data;
        int MAX_SIZE;
        int top;
   public:
        IntStack(){
                top=-1;
                MAX_SIZE = 50;
                Data = new int[MAX_SIZE];
        }
        IntStack(int size) {
                top=-1;
                MAX_SIZE=size;
                Data=new int[MAX_SIZE];
        }
        int isfull() {
                if(top==(MAX_SIZE-1)) return 1;
                else return 0;
        }
        int isempty() {
                if(top==-1) return 1;
                else return 0;
        }
        int topData() {
                if(isempty()){
                        cerr<<"Error:Stack Underflow"<<endl;
```

```
                return -1;
            }
            else return Data[top];
        }
        void push(int elem) {
            if (isfull()){
                cerr<<"Error:Stack is Full"<<endl;
            }
            else{
                Data[++top] = elem;
            }
        }
        int pop() {
            if(isempty()){
                cerr<<"Error:Stack Underflow"<<endl;
                return -1;
            }
            else {
                return Data[top--];
            }
        }
};

/**********ABOVE CODE IS PREPEND CODE*****/

/**You need to write your code here***/


/***************
**result(a,b,OP) evaluates the expression (a OP b)
**OP is operation belonging to {+, -, *, /} set
**Ex: result(2,1,'/') returns 2.
****************/

int result(int val1, int val2, char op){
    switch(op){
        case '*':
            return val1*val2;
        case '/':
            return val1/val2;
        case '+':
            return val1 + val2;
        case '-':
            return val1-val2;
        default:
            cerr<<"No such operator: "<<op<<endl;
            return -1;
    }
```

```
}
/**************
 * isOperator() takes in character pointer as input
 * and returns 1 if its one of the desired operators
 * returns -1 if the input is not an operator (ex: '9')
 * ************/
int isOperator(char * str){
    char ch=*str;
    if(ch=='*' || ch=='/' || ch=='+' || ch=='-' ){
            if(*(++str)=='\0') return 1;
    }
    return -1;
}
int main(int argc, char const *argv[])
{
    int N;
    char *str ;

    cin>>N;
    char *postfix[N];
    int i=0;
    while(i<N){
            str = new char[5];
            cin>>str;
            postfix[i++]=str;
            //cout<<"\t"<<str<<endl;
    }
    IntStack s;
    int v1, v2;
    //this for loop can be further simplified. and cleansed off couts
    for(i=0;i<N;i++){
            if (isOperator(postfix[i])==1){
                    //upon encountering an operator evaluate the result
                    //from top two elements of the stack and the operator
                    //value = stack[top] (op) stack[top-1]
                    //push the value onto the stack
                    cout<<"\tencountered an operator '"<<postfix[i]<<"'"<<endl;
                    if(s.isempty()){
                            cerr<<"INVALID"<<endl;
                            exit(1);
                    }
                    else{
                            v2 = s.pop();
                            if(s.isempty()){
                                    cerr<<"INVALID"<<endl;
                                    exit(1);
                            }
                            else{
                                    v1 = s.pop();
```

```
                                if(postfix[i][0]=='/' && v2==0){
                                        cerr<<"INVALID"<<endl;
                                        exit(1);
                                }
                                else{
                                        s.push(result(v1,v2,postfix[i][0]));
                                        cout<<"\tpushing "<<result(v1,v2,postfix[i][0])<<endl;
                                }
                        }
                }
        }
        else{
                //Upon encountering an operand,
                //push it onto  the stack
                s.push(atoi(postfix[i]));
                cout<<"\tpushing "<<atoi(postfix[i])<<endl;

        }
    }
    /**Finally pop the result of the postfix exression and after that
     * If the stack is empty then the given expression is valid
     * Otherwise declare it as invalid.
     * ******/
    int Ans;
    Ans = s.pop();
    if(s.isempty()) {
        cout<<"Answer: "<<Ans<<endl;
    }
    else{
        cout<<"INVALID"<<endl;
    }
    return 0;
}
/****end***/
```

Question 4: k-length walks in a simple graph

Important Note:

This assignment will NOT be graded. It serves as a practice problem.

A walk is an alternating sequence of vertices and edges, beginning and ending with a vertex, where each vertex is incident to both the edge that precedes it and the edge that follows it in the sequence, and where the vertices that precede and follow an edge are the end vertices of that edge. The length 'k' of a walk is the number of edges that it uses.

Hint: It might be useful to observe that the number of k-length walks between any two vertices v_i and v_j in G is the i-jth entry of A^k, where A is the adjacency matrix of G.

Given a simple graph G=(V,E) using adjacency matrix representation taught in the lectures and

a positive integer k, output the number of
k-length walks in G with the following exceptions:

1. Avoid walks that start and end at the same vertex (i.e., do not count diagonal elements).
2. The same walk should not be counted twice, e.g., treat 1 - 2 - 3 and 3 - 2 - 1 as the same walk (i.e., total number of walks divided by 2).

INPUT:
Number of vertices : n (a positive integer)
Value k (a positive integer)
Adj Matrix A of G : n x n (each entry is a 0 or a 1)

OUTPUT:
A single integer indicating the number of k-length walks in G.

Test Cases:

Public test cases:

Input	Output
5 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0
2 1 0 1 1 0	1
3 1 0 1 1 1 0 0 1 0 0	2
3 2 0 1 1 1 0 0 1 0 0	1
4 2 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0	12

Private test cases:

5 3 0 1 0 0 1 1 0 1 0 0	5

0 1 0 1 0 0 0 1 0 1 1 0 0 1 0	
1 1 0	0
6 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0	15
6 2 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0	60
3 3 0 1 1 1 0 1 1 1 0	3
5 4 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0	17

Solution:

/* You may re-write the whole code. The snippet given below is only as a guideline */

```
#include <stdio.h>
#include <stdlib.h>

typedef struct mat{
        int** matValues;
}matrixType;

matrixType initializeMatrix(matrixType anyMatrix, int numVertices){
   int i, j;

   anyMatrix.matValues = (int**) calloc(numVertices, sizeof(int*));
   for (i=0; i<numVertices; i++)
```

```
        anyMatrix.matValues[i] = (int*) calloc(numVertices, sizeof(int));
    for (i=0;i<numVertices;i++)
        for (j=0; j<numVertices;j++){
            anyMatrix.matValues[i][j] = 0;
    }
    return anyMatrix;
}

matrixType multiply(matrixType M1, matrixType M2, int numVertices){

    int i,j,k;
    matrixType resultMatrix;
      resultMatrix = initializeMatrix(resultMatrix,numVertices);
    for (i=0;i<numVertices;i++){
        for (j=0;j<numVertices;j++){
            resultMatrix.matValues[i][j] = 0;
            for (k=0;k<numVertices;k++){
                resultMatrix.matValues[i][j] =
resultMatrix.matValues[i][j]+M1.matValues[i][k]*M2.matValues[k][j];
            }
        }
    }
    return resultMatrix;
}
int main(){
int i,j,k, numVertices =0, count;
matrixType inputMatrix, powerMatrix;

scanf("%d",&numVertices);
scanf("%d",&k);

inputMatrix = initializeMatrix(inputMatrix, numVertices);
    powerMatrix = initializeMatrix(powerMatrix, numVertices);

    for(i=0; i<numVertices; i++){
        for(j=0; j<numVertices; j++){
            scanf("%d",&inputMatrix.matValues[i][j]);
        }
    }
    if (k!=1){
        powerMatrix = multiply(inputMatrix,inputMatrix,numVertices);
```

```
        for(i=2;i<k-1;i++){
                powerMatrix = multiply(powerMatrix,inputMatrix,numVertices);
        }
    }
    else{
        powerMatrix = inputMatrix;
    }
    count = 0;
    for(i=0; i<numVertices; i++){
        for(j=0; j<numVertices; j++){
                if (i!=j) count += powerMatrix.matValues[i][j];
        }
    }
    printf("%d", count/2);
}
```