ANSWER TO THE QUESTION NUMBER 1

->

Random access করতে গেলে আমদের array based stack আর linked list based stack এর মাঝে array based stack ভালো হবে ।

কারন আমরা জানি array তে প্রতিটি উপাদান একটির পর একটি থাকে। তাই শুদু index numder দিয়ে call করলেই আমরা এদেরকে access করতে পারব।

কিন্তু অপর দিকে আমরা জানি linked list এর বেলায় element গুলো momory তে একসাতে না থেকে memory এর বিভন্ন স্থানে থাকে তাই এদের random access করতে গেলে time complexity বেশি লাগে অপর দিকে array তে O(1) এ random access করা যায়।

আর ঠিক এই কারনেই array beased stack ভালো হবে linked list based stack অপেক্ষা।

ANSWER TO THE QUESTION NUMBER 2

->

1) head বা top এ push করা হলে time complexity হবে O(1)।

2) head বা top এ pop করা হলে time complexity হবে O(1)।

3)আবার আমরা যদি top element access করতে চাই তাহলেও time complexity O(1)

ANSWER TO THE QUESTION NUMBER 3

-> এক্ষেত্রে আমরা stack এর template base implemation করবো ।কারন ক্যারেক্টার, ইন্টিজার, রিয়াল নাম্বার এর stack এসব এর ক্ষেত্রে stack এর সকল code এক হবে শুদু মাত্র ডাটা এর ধরন আলাদা হবে।

এতে করে আমাদের কম code type করা লাগবে এবং code অরগানাইড হবে।

উদাহরনঃ-

```
#include<bits/stdc++.h>

using namespace std;
```

```
const int n=500;
template <class dataType>
class stacK
{
public:

        dataType* a;
        int  cap;
        int stack_size;
        stacK()
        {
        a=new dataType[1];
        cap=1;
        stack_size=0;
        }
// add an element in the stack o(1);
        void pushh(dataType val)
        {
        if(stack_size+1>n)
        {
        increase_size();
        }
        stack_size=stack_size+1;
        a[stack_size-1]=val;
        }
        //delete the topmost element of stack o(1)
```

```
        void pop()
        {
        if(stack_size==0)
        {
            cout<<"empty"<<"\n";
        return;
        }

        stack_size=stack_size-1;
        }
        // return the topmost element of stack o(1)
dataType top()
        {
        if(stack_size==0)
        {
        cout<<"empty"<<"\n";
        assert(false);
        }
        return a[stack_size-1];
        }
        //make the cpacity multiplied by 2 o(1);
        void increase_size()
        {
        dataType* temp;
        temp=new dataType[cap*2];
        for(int i=0; i<cap; i++)
```

```
        {
            temp[i]=a[i];
        }
        swap(a,temp);
        delete []temp;
        cap=cap*2;
        }
};
int main()
{
        stacK<char> l;
        l.pushh('g');
        l.pushh('j');
        cout<<l.top();
        return 0;
}
```

ANSWER TO THE QUESTION NUMBER 4

->postfix হলো এমন একটি এক্সপ্রেসন যেখনে অপারেটর অপারেন্ডের মাঝে না থেকে অপারেন্ডের পারে অবস্থান করে এবং তার পূরবের দুটি অপারেন্ডের উপর অপারেসন পরিচালনা করে।

কম্পিউটার যাতে খুব সহজে ম্যাথমেটিক অপারেটর গুলোর অপারেসন করতে পারে তাই আমরা postfix use করি।

উদাহরনঃ-

abc*+de*+

এটি stack এ যেভাবে evaluated হবে তা স্টেপ বাই স্টেপ দেখানো হলোঃ

১) a পুস করবো

২) b পুস করবো

৩) c পুস করবো

৪)b আর c pop এবং b*c (ধরি b*c=t1) এর রেজাল্ট পুস

৫)t1 pop এবং a+t1 (ধরি a+t1=t2) এর রেজাল্ট পুস

৬)) d পুস করবো

৭) e পুস করবো

8)d আর e pop এবং d*e (ধরি d*e=t3) এর রেজাল্ট পুস

৯) t2 আর t3 pop এবং t2+t3 (ধরি t2+t3 =t4) এর রেজাল্ট পুস

Final result t4.

ANSWER TO THE QUESTION NUMBER 5

(([ ][ ]{ ( ) }))

Balanced parentheses chack করার জন্য আমরা যা তা নিচে লেখা হলো

১) একটা ভেরিয়েবলে স্ট্রীং হেসাবে (([ ][ ]{ ( ) })) এটা নিবো

২) ১ম লুপ চলবে স্ট্রীং এর সাইজবার

৩) যতক্ষন open braket আসবে তাদের stack এ ভরবো

৪) close braket আসলে দেখবো stack এর top এ থাকা open braket এর ম্যাচ করে কেনা না করলে লুপ beak output no আর করলে stack এর top কে pop এইভাবে সামনে আগেবো

উক্ত ব্র্যাকেট সুকুয়েন্স এর জন্য Balanced parentheses chacker এর ধাপ গুলো দেখানো হলো

১) (([কে stack এ পুস

stack=(([

২)]এর stack top তুলোনা ম্যাচ করবে তাই stack pop

stack=((

৩) [কে stack এ পুস

stack=(([

৪)]এর stack top তুলোনা ম্যাচ করবে তাই stack pop

stack=((

৫) {(কে stack এ পুস

stack=(({(

৬)) এর stack top তুলোনা ম্যাচ করবে তাই stack pop

stack=(({

৭) } এর stack top তুলোনা ম্যাচ করবে তাই stack pop

stack=((

৮)) এর stack top তুলোনা ম্যাচ করবে তাই stack pop

stack=(

9)) এর stack top তুলোনা ম্যাচ করবে তাই stack pop

stack খালি

এভাবে লুপ শেষ হবার পর যদি দেখা যায় **stack** খালি তাহলে **answer yes** তা না হলে **no**

ANSWER TO THE QUESTION NUMBER 6

->

stack1 -> [3,4,6,2,5]

উক্ত stack টিকে আমরা অপর আর একটি stack এর সাহায্যে sort করবো

stack2 -> []

Step>>

1) Push 5 in stack2 and pop form stack1

2) pop 5 from stack2 , push 2 in stack2, pop 2 from stack1,push 5 in stack1

3) Push 5 in stack2 and pop form stack1

4) Push 6 in stack2 and pop form stack1

5) pop 6 from stack2, pop 5 from stack2, , push 4 in stack2, pop 4 from stack1, Push 6 in stack1, Push 5 in stack1

6) Push 5 in stack2 and pop form stack1

7) Push 6 in stack2 and pop form stack1

8) pop 6 from stack2, pop 5 from stack2, , push 3 in stack2, pop 3 from stack1, Push 6 in stack1, Push 5 in stack1

9) Push 5 in stack2 and pop form stack1

10) Push 6 in stack2 and pop form stack1

ANSWER TO THE QUESTION NUMBER 7

->

a+b*c+d*e কে postfix এ রুপান্তর করা হলোঃ-

১ম এ এটকা result নামক string variable এবং একটি stack নিবো।

ধাপসমূহঃ-

1) Add a in result (result=a)

2) Push + in stack (stack=+)

3) Add b in result (result=ab)

4) Push * in stack (stack=+*)

5) Add c in result (result=abc)

6) এখন stack এ + push করার সময় প্রগ্রাম দেখবে + এর থেকে বেশি শক্তিশালী * stack এর top অবস্থান করসে তাই একে আগে pop করে result এ add করবে। এখন (result=abc*) এবং (stack= +)

7) এখন stack এ + push করার সময় প্রগ্রাম দেখবে + এর সমান শক্তিশালী * stack এর top অবস্থান করসে তাই একে আগে pop করে result এ add করবে। এখন (result=abc*+) এবং (stack= +)

8) Add d in result (result=abc*+d)

9) Push * in stack (stack=+*)

10) Add d in result (result=abc*+de)

11) Pop * from stack and add in result এখন result (result=abc*+de*) এবং (stack= +)

12) Pop + from stack and add in result এখন result (result=abc*+de*+) এবং (stack= খালি)

তাহলে ফাইনাল রেজাল্ট হলো abc*+de*+