Answer Script

Question No. 01

Between array based stack implementation and linked-list based stack implementation which is better when I need random access in the stack? Explain the reasons.

Answer No. 01

I can randomly access the stack by array-based stack implementation.
In the linked list there is a problem with nodes and it takes more time for random access. I can not access the middle element directly.
On the other hand, in an array there is an index. Which can help to randomly access. Travel to the middle element easily by moving the index right or left.
If I want to explain by definition:

An array is a random access data structure, where each element can be accessed directly and in constant time. A typical illustration of random access is a book - each page of the book can be open independently of others. Random access

A linked list is a sequential access data structure, where each element can be accessed only in particular order. A typical illustration of sequential access is a roll of paper or tape - all prior material must be unrolled in order to get to the data I want.

So, it can be said that array-based stack implementation is better for random access.

Question No. 02

What is the time complexity of push, pop and top operations in a stack?

Answer No. 02

	Push	Pop	Top
Time complexity	O(1)	O(1)	O(1)

Question No. 03

Suppose you need a stack of characters, a stack of integers and a stack of real numbers. How will you implement this scenario using a single stack?

Answer No. 03

If I provide a program:

```
#include<bits/stdc++.h>
using namespace std;
template<class T>
class Stack
{
public:
   T *a;
   int sz;
   int cap;
   Stack()
   {
      a= new T[1];
      cap=1;
      sz=0;
   }
   void increase_size()
   {
      T *b;
      b= new T[cap*2];
      for(int i=0;i<cap;i++)
      {
         b[i]=a[i];
       }
         swap(a,b);
         delete []b;
         cap*=2;
   }
   void push(T val)
   {
      if(sz+1>cap)
      {
         increase_size();
      }
      sz+=1;
      a[sz-1]=val;
   }
```

```
    void pop()
    {
        if(sz==0)
        {
            return;
        }
        sz--;
    }
    T top()
    {
        return a[sz-1];
    }
};
int main()
{
    Stack<int>A;
    A.push(10);
    cout<<A.top()<<" ";
    Stack<char>B;
    B.push('B');
    cout<<B.top()<<" ";
    Stack<float>C;
    C.push(2.56);
    cout<<C.top()<<" ";
}
```

Here I use a template. As I can use a single stack for all of the data types. For integer is int, for character is char, for real number is float. I do not need to create another stack for other data types.

So, by dynamic array based stack implementation I can do that and the scenario will look like this.

Question No. 04

What is a postfix expression and why do we need it? How is it evaluated using a stack for the below example? You need to show all the steps.

abc*+de*+

Answer No. 04

In expression, operands are written before the operators to execute the operation more perfectly for a computer is called postfix expression.

Because the postfix expression works faster than the infix expression. There is no concern which operator's precedence is first . Besides, in postfix expressions we do not need to use parenthesis. For doing the operation properly on the computer we need it.

The infix expression of **abc*+de*+** is **a+b*c+d*e**. I will take the infix expression as input.

Procedure:

1. Declaration of string and input the infix expression as string .
   string s;cin>>s;
2. Create a stack of characters.
   stack<char>S;
3. Declare a blank string.
   string r="";
4. Then I start a loop from 0 to before the string size of s and make a condition if the character is a to z. If it is true then input it in the S blank string.
   for(int i=0;i<s.size();i++)
   {if(s[i]>= 'a' && s[i]<= 'z')
   {r+=s[i];}
5. Then I create a function name precedence which will return 0 if the character is + or -. Other it will return 1; int precedence(char r){if(r== '+' || r== '-'){return 0;}return 1;}
6. If it is not then I have to check if it is '+' or '-' sign. If this comes then I can push the sign in the stack.
   else{while(S.size() && precedence(S.top())>=precedence(s[i]))
   r+=S.top();
   S.pop();}
   S.push(s[i]);)}}
   Again if '*' or '/' sign come then first of all I push it in the stack, later if '+' or '-' sign come then I add the *,/ characters in r string and delete the top character of the stack and also add the +,- sign in the string until the stack size becomes 0.
7. After that if there is any value in the stack then lastly I add it to the string and pop the stack.
8. At last I print the r string.

Question No. 05

Simulate balanced parentheses check using stack for the below example. You need to show all the steps.

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

Answer No. 05

Procedure:

1. First I have to declare the string and input the string of parenthesis. string s; cin>>s;
2. Declare a character stack.
   stack<char>S;
3. Then run a loop from 0 to s.size and if I get the opening parentheses then I push it into the stack. Otherwise I will check if the stack is empty or not. If it is empty then print invalid or no.
4. If it is not empty then match the close and opening parenthesis and check it.
5. If it matches then pop the top element of the stack , otherwise print invalid.
6. After ending the loop if the stack is empty then print valid otherwise print invalid.

Checking:

0	1	2	3	4	5	6	7	8	9	10	11
(	(	[	]	[	]	{	(	)	}	)	)

1. ([as it is opening parenthesis, so push it into the stack]
2. ((
3. (([
4. (([] -> (([as top opening element and the closing element is same quality, so it is popped]
5. (([
6. (([] -> ((
7. (({
8. (({(
9. (({() -> (({
10. (({} -> ((
11. (() -> (
12. () -> valid parentheses.

Question No. 06

Sort a stack of integers using another stack for the below example. You need to show all the steps.

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

Answer No. 06

Steps:

1. Create a stack S by pushing 3, 4 ,6 , 2, 5.
2. Create another stack R .
3. Take an integer t to communicate with value.
4. Then run a while loop for S until the size becomes 0;
5. Then run a while loop for R until size becomes 0;

6. If R is empty then push the top of S to the R and pop the S.
7. After that, compare the present top of S, if it is less than R top then pop the R and push the present top of S and push the previous top of R. do this until S size becomes 0;
8. If it is greater then push it to the R stack.
9. At last swap the S and R stack and print the S.

Implement:

1. S={3,4,6,2,5},R={},t=5
2. S={3,4,6,2} ,R={5},t=2 [as 5>2]
3. S={3,4,6,5} ,R={2},t=5
4. S={3,4,6} ,R={2,5},t=6
5. S={3,4} ,R={2,5,6},t=4
6. S={3,6},R={2,5},t=4 [as 6>4]
7. S={3,6,5},R={2,4},t=5 [as 5>4]
8. S={3,6},R={2,4,5},t=6
9. S={3}, R={2,4,5,6},t=3
10. S={6},R={2,4,5},t=3 [as 3<6]
11. S={6,5},R={2,4},t=3 [as 3<5]
12. S={6,5,4},R={2,3},t=4 [as 3<4]
13. S={6,5},R={2,3,4},t=5
14. S={6},R={2,3,4,5},t=6
15. S={},R={2,3,4,5,6},t=6
16. S={2,3,4,5,6},R={},t=6

Sort in increasing order is 2,3,4,5,6.

Question No. 07

Convert the infix expression to postfix expression using a stack. You need to show all the steps.

a+b*c+d*e

Answer No. 07

The procedure is written in question no-04.

Implement:

1. s= “a+b*c+d*e”, r= “”,S={}.
2. s= “a+b*c+d*e”,r= “a”,S={}.
3. s= “a+b*c+d*e”,r= “a”, S={‘+’}.[as s[1]= ‘+’]
4. s= “a+b*c+d*e”,r= “ab”,S={‘+’}.
5. s= “a+b*c+d*e”,r= “ab”, S={‘+’, ‘*’}. [as s[3]= ‘*’]
6. s= “a+b*c+d*e”,r= “abc”,S={‘+’, ‘*’}.
7. s= “a+b*c+d*e”,r= “abc*”, S={‘+’}[s[5]= ‘+’ & the precedence of + & * is *]

8. s= “a+b*c+d*e”,r= “abc*+”,S={}
9. s= “a+b*c+d*e”,r= “abc*+”, S={‘+’}
10. s= “a+b*c+d*e”, r= “abc*+d”, S={‘+’}
11. s= “a+b*c+d*e”, r= “abc*+d”, S={‘+’, ‘*’}
12. s= “a+b*c+d*e”, r= “abc*+de”, S={‘+’, ‘*’}

As the loop is over then we have to check if there is any value in the stack.

1. r= “abc*+de*”, S={‘+’}
2. r= “abc*+de*+” ,S={}

Then if it prints, it will show in console—> abc*+de*+