Lecture 21

Stacks

November 01, 2021

Monday

RECALL ADT

Abstract Data Types allow us to delay the specific implementation of a data type until it is clearly understood what operations are required.

STACK FUNDAMENTALS

Stack is a linear Data Structure that can be accessed only at one of its ends for storing and receiving data.

- In stack the first element is accessed last, and the last element is accessed first.
- In simpler words it follows **Last In First Out (LIFO)** order.

STACK'S FUNDAMENTALS

- The pancakes are in LIFO order to get the first one we have to get all on the top.
- A pancake can only be taken if there is at least one pancake on the stack.
- A pancake can only be added to the stack if there is enough room.

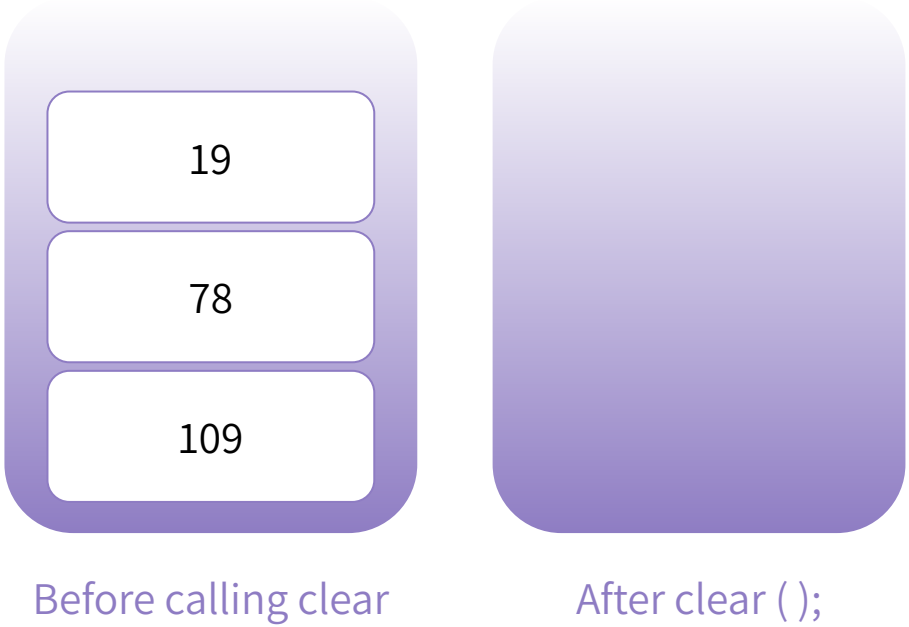
STACK'S OPERATIONS

- A stack can be defined in terms of operations
  - that change its status
  - operations that can check its status
- A stack usually consists of following operations
  - *clear ( );*
  - *isEmpty ( );*
  - *push (element);*
  - *pop ( );*
  - *peek ( );*

STACK'S OPERATIONS

clear ( );

clears the stack, all elements are deleted.



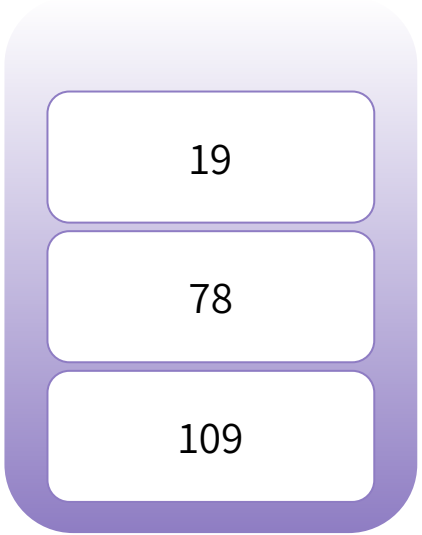
IMPLEMENTATION | CLEAR ( )

```
void clear ( )
 {
        if ( !isEmpty ( ) )
        {
                while (top != NULL)
                {
                        temp = top;
                        top = top->prev;
                        delete temp;
                        temp = NULL;
                }
        }
}
```

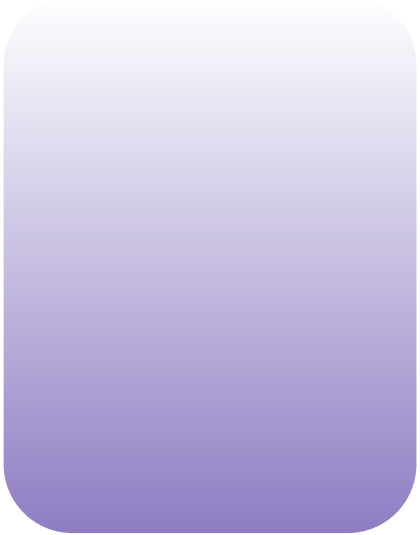
STACK'S OPERATIONS

isEmpty ( );

Check to see if the stack is empty?



Returns False / 0



Returns True / 1

IMPLEMENTATION | CLEAR ( )

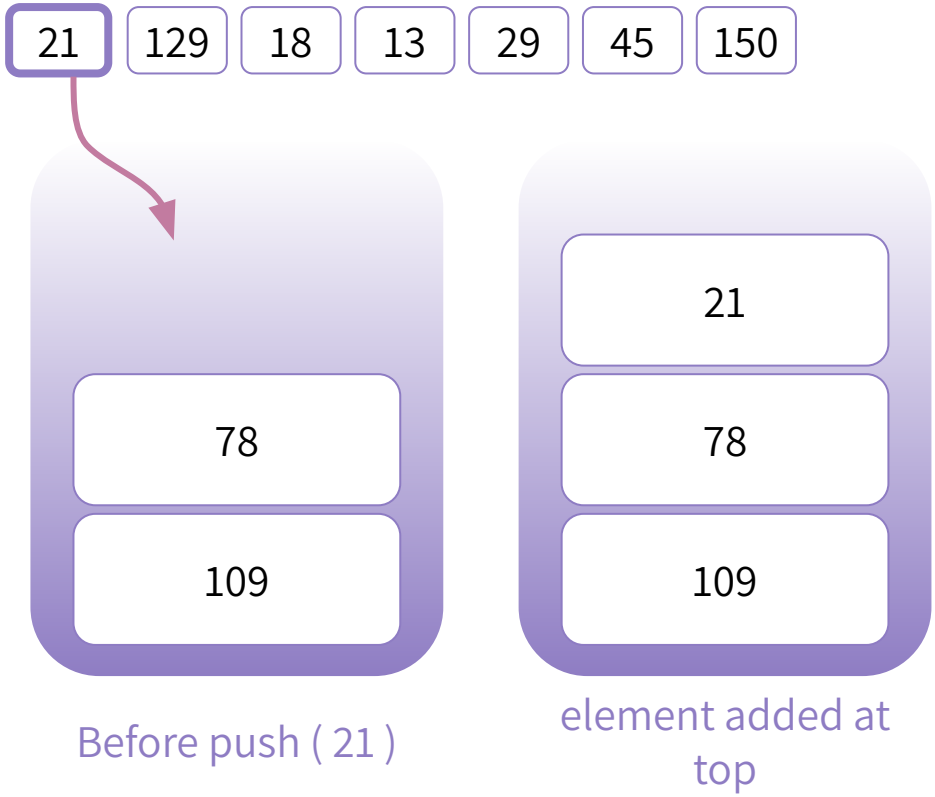
```
int isEmpty ( )
{
        return top == NULL;
}
```

STACK'S OPERATIONS

push (el);

Put the element el on

top of the stack.



IMPLEMENTATION | push (int)

```
void push ( int el )
{
        top = new Node(el, top);
        if ( !top ) {
                cout<< " Stack Overflow ";
                exit ( 1 );
        }
}
```

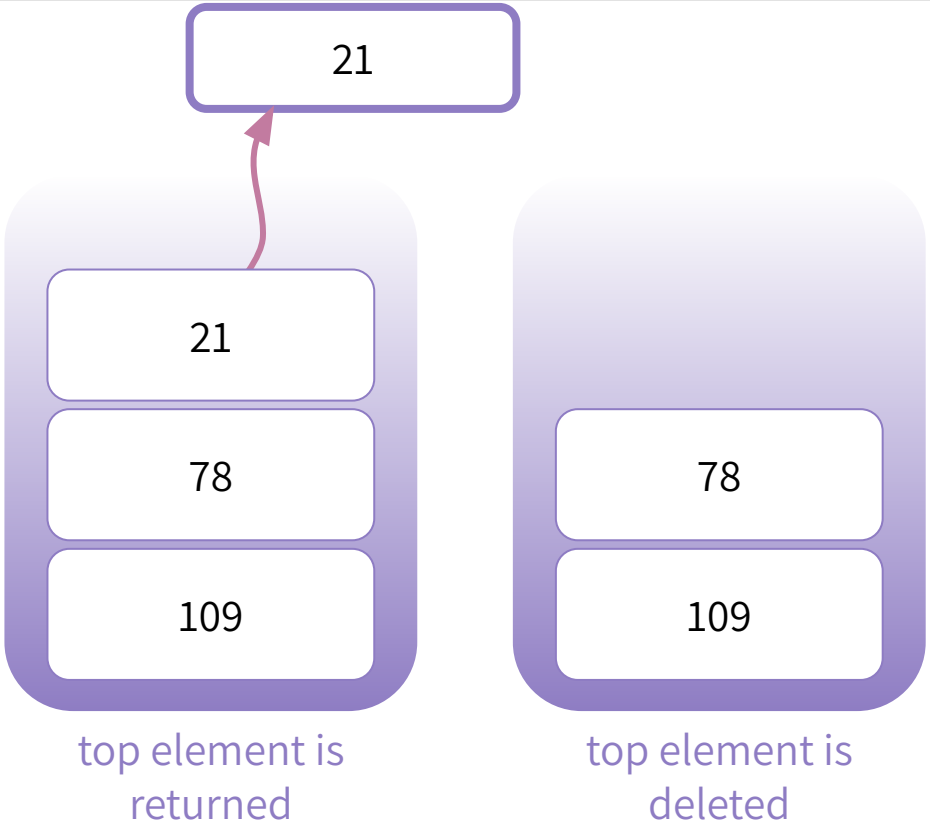
```
// Parameterized Constructor of Node
Node(int value, Node *ptr = 0)
{
        info = value;
        prev = ptr;
}
```

STACK'S OPERATIONS

pop ( );

Returns the top most element from the stack.

The top most element is removed from the stack as well.



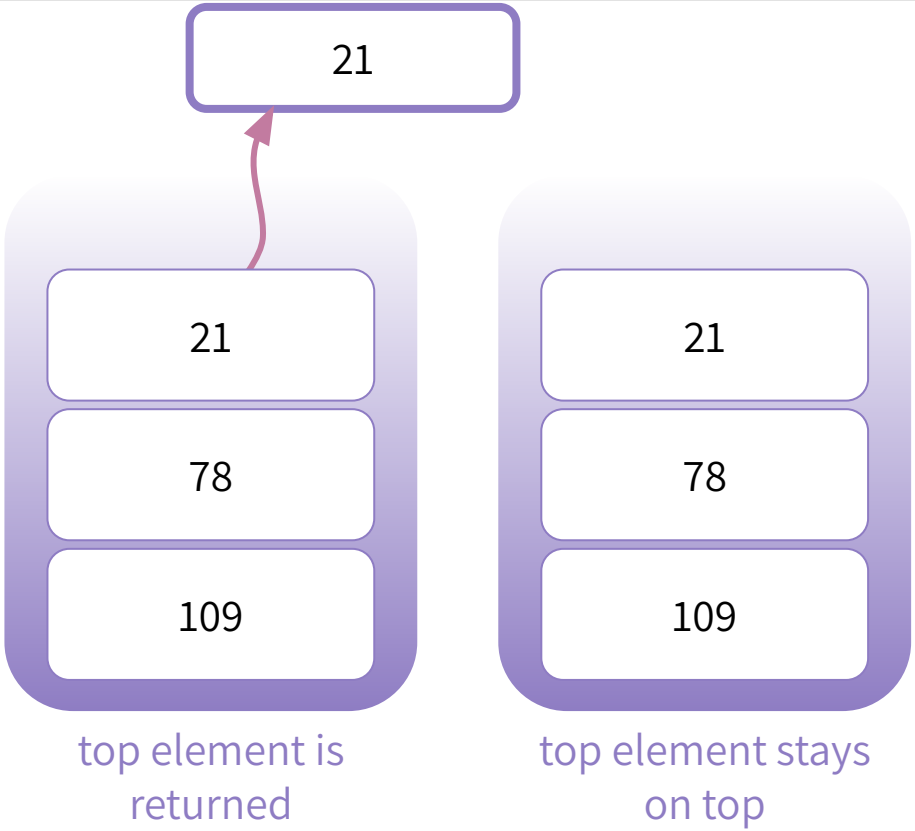
IMPLEMENTATION | pop ( )

```
int pop( )
 {
        int value = -1;

        if (!isEmpty())
        {
                value = top->info;
                temp = top;
                top = top->prev;
                delete temp;
                temp = NULL;
        }

     return value;
}
```

STACK'S OPERATIONS

peek ( );

Returns the top most element from the stack, without deleting it from the stack.



IMPLEMENTATION | peek ( )

```
int peek ( )
 {
        if (isEmpty ( ) )
        {
                cout << "Stack is empty..!";
                return -1;
        }
        else
        {
                 return top->info;
        }
 }
```

STACK ARRAY IMPLEMENTATION

CPP Attached.

DELIMITER MATCHING

- Delimiter matching algorithm reads a character from a C++ program
  - If it is an opening delimiter, push it on stack.
  - If it is a closing delimiter
    - pop the stack and match the popped element with current delimiter.
      - If they match, processing continues.
      - discontinue the processing with an error.
  - The processing is successful if
    - end of the program is reached, and the stack is empty.

DELIMITER MATCHING

```
delimiterMatching ( file )
    while not end of file
        read character ch from file
            if ch is '(', '[', or '{'
                push ( ch );
            else if ch is ')', ']', or '}'
                if ch and popped off delimiter do not match
                    return failure;
            else if ch is '/'
                read the next character
                    if this character is '*'
                        skip all characters until '*' is found
                        if end of file is reached and '*' is not found
                            return failure;
```

DELIMITER MATCHING

```
                    else ch = the character read in;
                        continue;
if stack is empty
    return success;
else
    return failure;
```

INTERESTING IDEAS ABOUT STACKS!

- We can use another stack to reverse the current stack.
- We can reverse a stack using a Queue as well
- A stack can be reversed with recursion as well.
- What about merging two stacks?
- What about adding two numbers using Stacks?

SAMPLE PROBLEM FROM LAST MID

A fellow Fastian propose a new data structures DualStack, which is an array based implementation that supports two different stacks in the same linear array. He proposed pair of functions like (Push1, Push 2), (Top1, Top2) and (Pop1, Pop2) such that these methods efficiently utilize the available space within the array. There should not be any overflow. Define the template based class for DualStack with proper functionality of each operator on this ADT that is in Bold text.