C++ Stack

Computer Engineering
Shahed University
2021 – winter

Teacher Assistants : **Amirreza Tavakoli**
amirrezatav2@gmail.com

What is a Stack?

- A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.

What is a Stack?

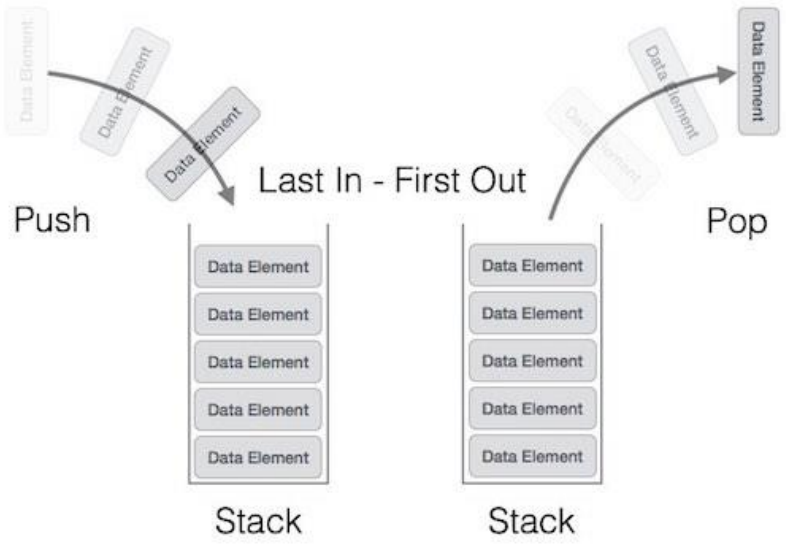
- A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from the top of the stack only. Likewise, Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack.

What is a Stack?

- this feature makes it LIFO data structure.
- LIFO stands for Last-in-first-out.
- Here, the element which is placed (inserted or added) last, is accessed first.
- In stack terminology, insertion operation is called PUSH operation
- and removal operation is called POP operation.

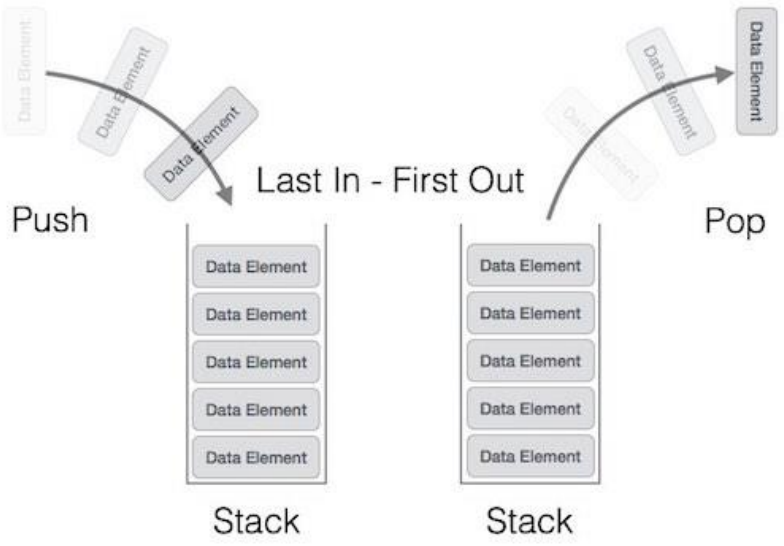
Stack Representation

- The following diagram depicts a stack and its operations −



Stack Representation

- The following diagram depicts a stack and its operations −



Stack Representation

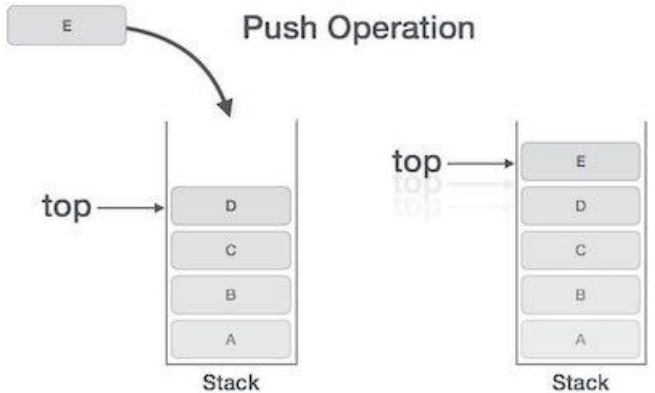
- A stack can be implemented by means of Array, Structure, Pointer, and Linked List. Stack can either be a fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using arrays, which makes it a fixed size stack implementation.

Basic Operations

- **push()** − Pushing (storing) an element on the stack.
- **pop()** − Removing (accessing) an element from the stack
- **peek()** − get the top data element of the stack, without removing it.
- **isFull()** − check if stack is full.
- **isEmpty()** − check if stack is empty.

Push Operation

- **Step 1** − Checks if the stack is full.
- **Step 2** − If the stack is full, produces an error and exit**.**
- **Step 3** − If the stack is not full, increments top to point next empty space.
- **Step 4** − Adds data element to the stack location, where top is pointing.
- **Step 5** − Returns success.



Algorithm for PUSH Operation

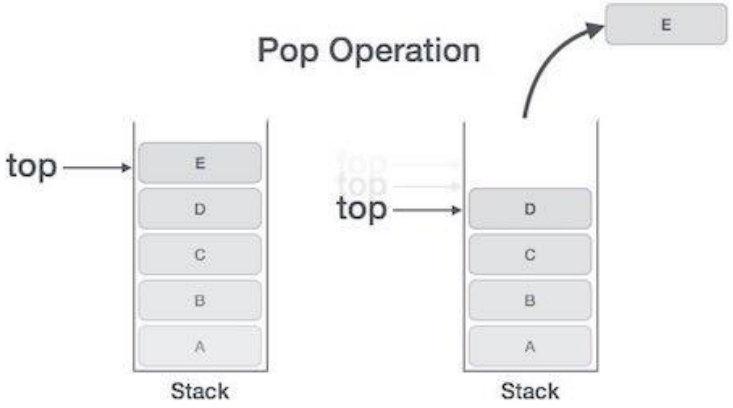
```
begin procedure push: stack, data

   if stack is full
      return null
   endif

   top ← top + 1
   stack[top] ← data

end procedure
```

Implementation of this algorithm in C++, is very easy. See the following code

```
void push(int data) {
    if (!isFull()) {
        top = top + 1;
        stack[top] = data;
    }
    else {
        throw ("Could not insert data, Stack is full.\n");
    }
}
```

Pop Operation

- **Step 1** − Checks if the stack is empty.
- **Step 2** − If the stack is empty, produces an error and exit.
- **Step 3** − If the stack is not empty, accesses the data element at which top is pointing.
- **Step 4** − Decreases the value of top by 1.
- **Step 5** − Returns success.



Algorithm for Pop Operation

amirrezatav2@gmail.com

```
begin procedure pop: stack

   if stack is empty
      return null
   endif

   data ← stack[top]
   top ← top - 1
   return data

end procedure
```

Implementation of this algorithm in C++, is very easy. See the following code

```
int pop(int data) {

    if (!isempty()) {
        data = stack[top];
        top = top - 1;
        return data;
    }
    else {
        throw ("Could not retrieve data, Stack is empty.\n");
    }
}
```

Algorithm of peek() function

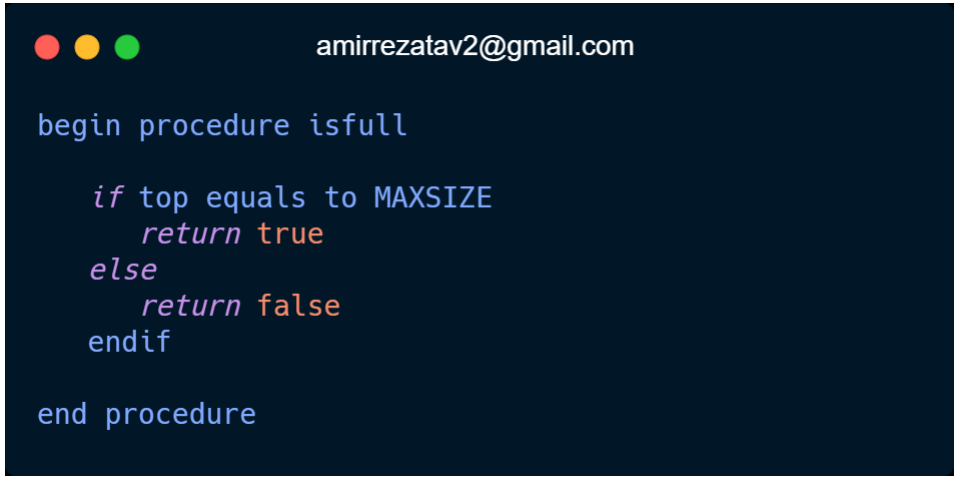
amirrezatav2@gmail.com

```
begin procedure peek
   return stack[top]
end procedure
```

Implementation of peek() function in C++ programming language −

```
int peek() {
return stack[top];
}
```

Algorithm of isfull() function –



Implementation of isfull() function in C++ programming language –

```
bool isfull() {
    if (top == MAXSIZE)
        return true;
    else
        return false;
}
```

Algorithm of isempty() function –

```
amirrezatav2@gmail.com

begin procedure isempty

   if top less than 1
      return true
   else
      return false
   endif

end procedure
```

Implementation of isempty() function in C++ programming language –

```
bool isempty() {
    if (top == -1)
        return true;
    else
        return false;
}
```

Stack Implementation C++

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

#define MAXSIZE 1000

template <class T>
class Stack {
    int top;
public:
    T a[MAXSIZE];

    Stack()
    {
        top = -1;
    }
    bool push(int x)
    {
        if (isfull()) {
            throw ( "Stack Overflow");
        }
        else {
            a[++top] = x;
            cout << x << " pushed into stack\n";
            return true;
        }
    }
int pop()
    {
        if (isEmpty()) {
            throw ("Stack Underflow");
        }
        else {
            int x = a[top--];
            return x;
        }
    }
    int peek()
    {
        if (isEmpty()) {
            throw ("Stack is Empty");
        }
        else {
            int x = a[top];
            return x;
        }
    }
    bool isEmpty()
    {
        return (top == -1);
    }
    bool isfull() {
        return top == MAXSIZE;
    }
};
// Driver program to test above functions
int main()
{
    class Stack<int> s;
    s.push(10);
    s.push(20);
    s.push(30);
    // print top element in stack and remove it
    cout << s.pop() << " Popped from stack\n";
    //print all elements in stack :
    cout << "Elements present in stack : ";
    while (!s.isEmpty())
    {
        // print top element in stack
        cout << s.peek() << " ";
        // remove top element in stack
        s.pop();
    }
    return 0;
}
```

Stack Library

```
template<class T, class Container = deque<T> > class stack;
```

Function	Description
(constructor)	The function is used for the construction of a stack container.
empty	The function is used to test for the emptiness of a stack. If the stack is empty the function returns true else false.
size	The function returns the size of the stack container, which is a measure of the number of elements stored in the stack.
top	The function is used to access the top element of the stack. The element plays a very important role as all the insertion and deletion operations are performed at the top element.
push	The function is used for the insertion of a new element at the top of the stack.
pop	The function is used for the deletion of element, the element in the stack is deleted from the top.
emplace	The function is used for insertion of new elements in the stack above the current top element.

Stack Library

```
#include <iostream>
#include <stack>
using namespace std;
void newstack(stack <int> ss)
{
    stack <int> sg = ss;
    while (!sg.empty())
    {
        cout << '\t' << sg.top();
        sg.pop();
    }
    cout << '\n';}
int main()
{
    stack <int> newst;
    newst.push(55);
    newst.push(44);
    newst.push(33);
    newst.push(22);
    newst.push(11);
    cout << "The stack newst is : ";
    newstack(newst);
    cout << "\n newst.size() : " << newst.size();
    cout << "\n newst.top() : " << newst.top();
    cout << "\n newst.pop() : ";
    newst.pop();
    newstack(newst);
    return 0;}
```

```
The stack newst is :    11    22    33    44    55

 newst.size() : 5
 newst.top() : 11
 newst.pop() :   22    33    44    55
```

Data Structure - Expression Parsing

The way to write arithmetic expression is known as a **notation**. An arithmetic expression can be written in three different but equivalent notations, i.e., without changing the essence or output of an expression. These notations are −

- Infix Notation
- Prefix (Polish) Notation
- Postfix (Reverse-Polish) Notation

These notations are named as how they use operator in expression. We shall learn the same here in this chapter.

Infix Notation

We write expression in infix notation, e.g. a - b + c, where operators are used in-between operands. It is easy for us humans to read, write, and speak in infix notation but the same does not go well with computing devices. An algorithm to process infix notation could be difficult and costly in terms of time and space consumption.

Prefix Notation

In this notation, operator is prefixed to operands, i.e. operator is written ahead of operands. For example, +ab. This is equivalent to its infix notation a + b. Prefix notation is also known as Polish Notation.

Postfix Notation

This notation style is known as Reversed Polish Notation. In this notation style, the operator is postfixed to the operands i.e., the operator is written after the operands. For example, ab+. This is equivalent to its infix notation a + b.

The following table briefly tries to show the difference in all three notations −

Sr.No.	Infix Notation	Prefix Notation	Postfix Notation
1	a + b	+ a b	a b +
2	(a + b) ∗ c	∗ + a b c	a b + c ∗
3	a ∗ (b + c)	∗ a + b c	a b c + ∗
4	a / b + c / d	+ / a b / c d	a b / c d / +
5	(a + b) ∗ (c + d)	∗ + a b + c d	a b + c d + ∗
6	((a + b) ∗ c) - d	- ∗ + a b c d	a b + c ∗ d -

Parsing Expressions

As we have discussed, it is not a very efficient way to design an algorithm or program to parse infix notations. Instead, these infix notations are first converted into either postfix or prefix notations and then computed.

To parse any arithmetic expression, we need to take care of operator precedence and associativity also.

Precedence
When an operand is in between two different operators, which operator will take the operand first, is decided by the precedence of an operator over others. For example −

Associativity

Associativity describes the rule where operators with the same precedence appear in an expression. For example, in expression a + b − c, both + and – have the same precedence, then which part of the expression will be evaluated first, is determined by associativity of those operators. Here, both + and − are left associative, so the expression will be evaluated as (a + b) − c.

Sr.No.	Operator	Precedence	Associativity
1	Exponentiation ^	Highest	Right Associative
2	Multiplication (∗) & Division (/)	Second Highest	Left Associative
3	Addition (+) & Subtraction (−)	Lowest	Left Associative

Postfix Evaluation Algorithm

amirrezatav2@gmail.com

```
Step 1 - scan the expression from left to right
Step 2 - if it is an operand push it to stack
Step 3 - if it is an operator pull operand from stack and perform operation
Step 4 - store the output of step 3, back to stack
Step 5 - scan the expression until all operands are consumed
Step 6 - pop the stack and perform operation
```

Queue (FIFO)?


LAST IN
LAST OUT
ONE WAY
FIRST IN
FIRST OUT
In
Data
Data
Data
Data
Data
Data
Out
Last In Last Out
First In First Out
Queue

Good luck!