Parul® University

Unit 2
Stacks and queue

Prof. varsha Naregalkar, Assistant Professor
Computer Science & Engineering

Parul®
University

DIGITAL LEARNING CONTENT

Topic-1

Stacks

ADT Stack and its operations:

- 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.
- ADT - The Data Type is basically a type of data that can be used in different computer program. It signifies the type like integer, float etc, the space like integer will take 4-bytes, character will take 1-byte of space etc.

Image source : Google

ADT Stack and its operations:

- The abstract data type is special kind of data type, whose behavior is defined by a set of values and set of operations. The keyword “Abstract” is used as we can use these data types, we can perform different operations. But how those operations are working that is totally hidden from the user. The ADT is made of with primitive data types, but operation logics are hidden.

- 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.

- 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

- 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.

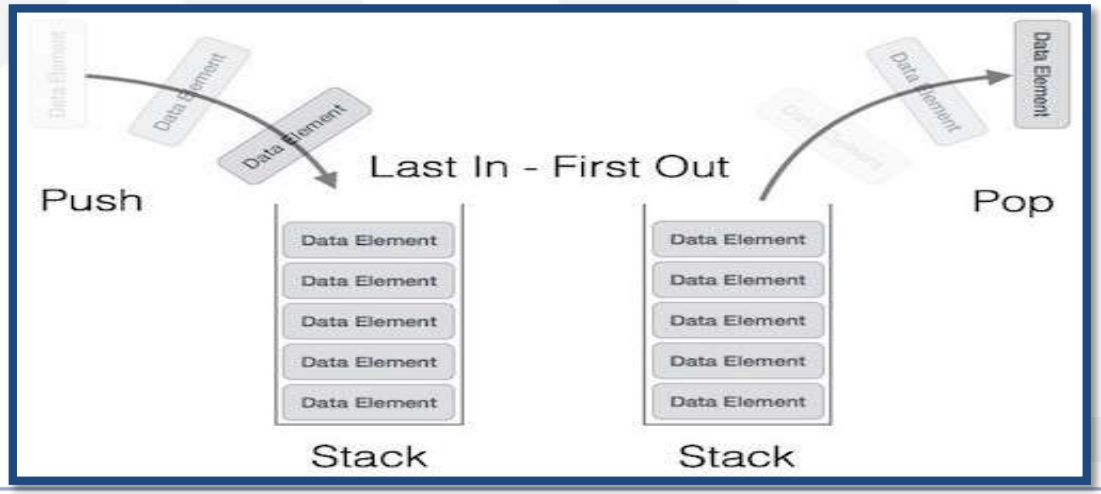


Image source : Google

Basic Operations

- Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations −

push() − Pushing (storing) an element on the stack.

pop() − Removing (accessing) an element from the stack.

Stack Operations

•To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks −

peek() − get the top data element of the stack, without removing it.

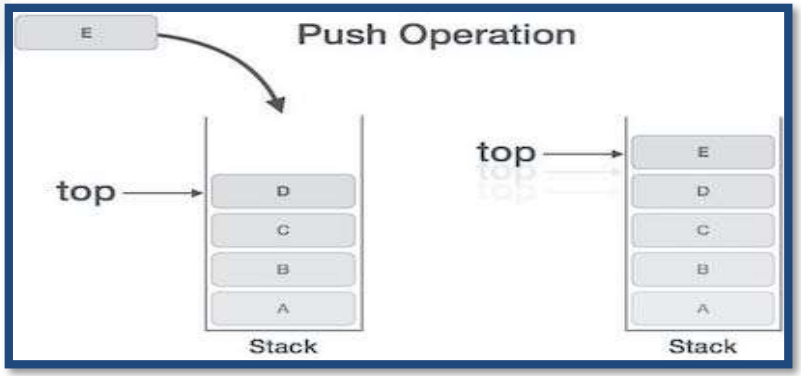
isFull() − check if stack is full.

isEmpty() − check if stack is empty.

At all times, we maintain a pointer to the last PUSHed data on the stack. As this pointer always represents the top of the stack, hence named **top**.
The **top** pointer provides top value of the stack without actually removing it.

Push Operation

•The process of putting a new data element onto stack is known as a Push Operation.
•Push operation involves a series of steps.
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.



Push Function in ‘C’

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

Pop Operation

- Accessing the content while removing it from the stack, is known as a Pop Operation. In an array implementation of pop() operation, the data element is not actually removed, instead **top** is decremented to a lower position in the stack to point to the next value. But in linked-list implementation, pop() actually removes data element and deallocates memory space.
- A Pop operation may involve the following steps –

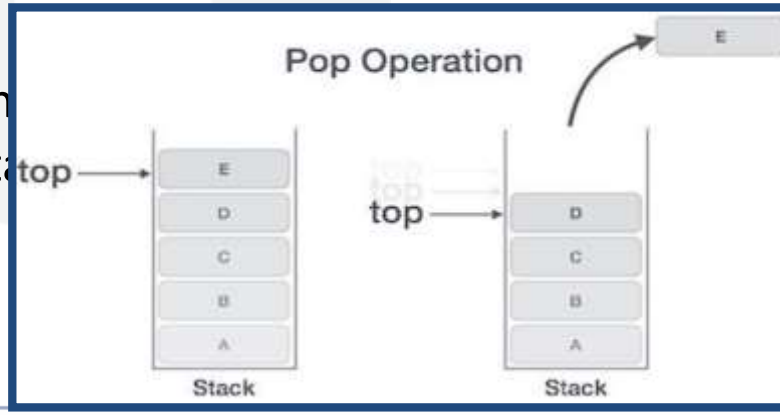
Step 1 – Checks if the stack is empty.

Step 2 – If the stack is empty, produces an error an

Step 3 – If the stack is not empty, accesses the dat which **top** is pointing.

Step 4 – Decreases the value of top by 1.

Step 5 – Returns success.



Pop Function in ‘C’

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

Analysis of Stack Operations

Below mentioned are the time complexities for various operations that can be performed on the Stack data structure.

Push Operation : O(1)
Pop Operation : O(1)
Top Operation : O(1)
Search Operation : O(n)

The time complexities for push() and pop() functions are O(1) because we always have to insert or remove the data from the **top** of the stack, which is a one step process.

Application's Of Stack

- Recursion
- Polish Expression & their Complication
- Reverse Polish Expression & their Complication
- Stacks are used by compilers to check weather the contents of "string" belong to a particular language

Recursion

- Recursion is a programming Technique in which a function contains a function call to itself
- Recursion and iteration are different processes. Recursion is a process in which the function contains a function to call itself. Where iteration is a process where the group of statement is executed repeatedly.

Eg:

```
factorial(int x) {
    int f;
   if(x==1)
     return(1);
  else
    f=x*factorial(x-1);   //This is recursion(recursive call)
      return (f);
}
```

Types Of Recursion

- Direct Recursion

Eg:-factorial(int x){
factorial(int x-1); }

- Indirect Recursion

Eg:-

```
int a()
 {
   b();
  }
 int b()
  {
    a();
  }
```

Tower Of Hanoi

- Tower of Hanoi basically comes from a Chinese or Japanese source
- It's a simple game concept which can be applied in stacks. The concept is that the bigger value should not be kept upon the smaller value
- It is necessary to stack all disks onto a third tower in decreasing order of size
- The second tower may be used as temporary storage
- The conditions are
  1) only one value may be moved at a time
  2) the larger value never rest upon smaller value
- Formula for finding minimum number of moves is 2^n-1

Algorithm For Tower Of Hanoi

Step 1:- if n=1 then print”disc -1 from A->C & return”

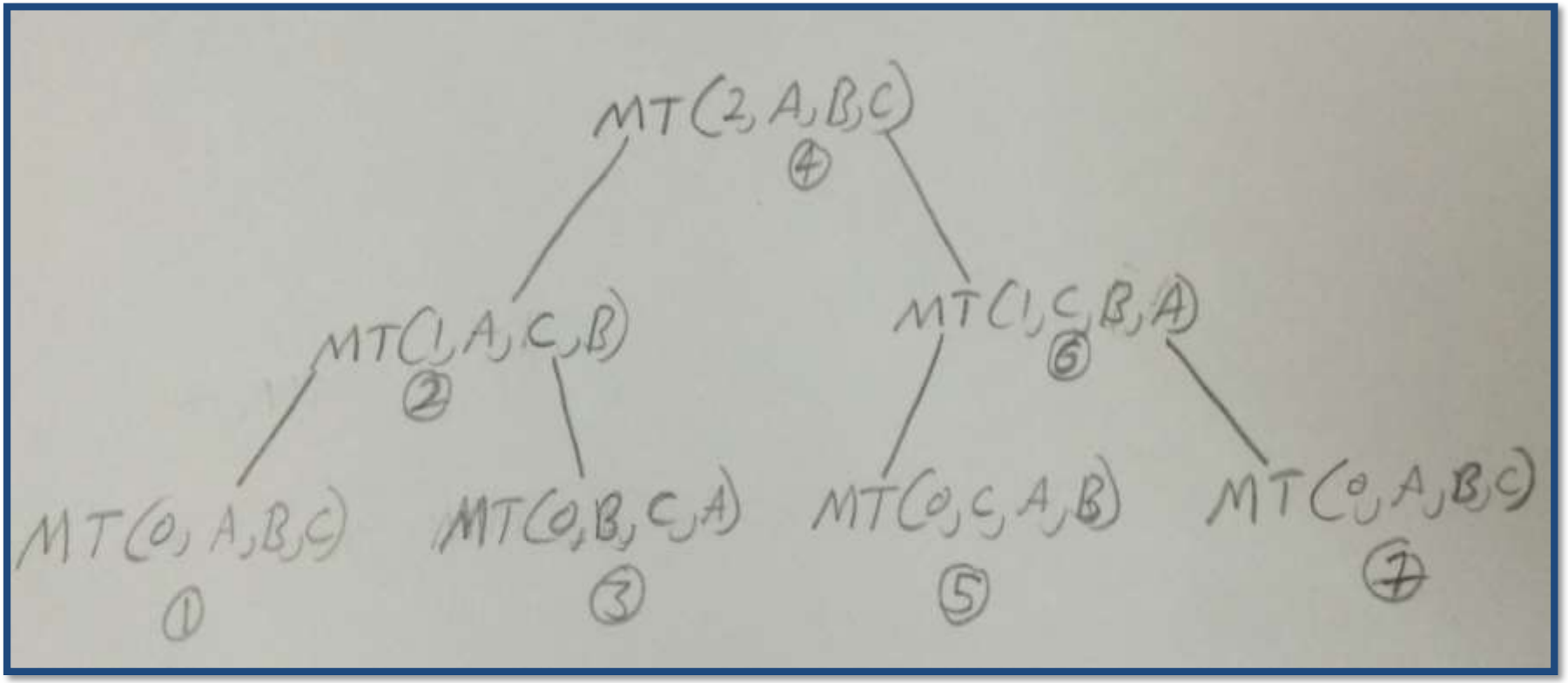
Step 2:- [Move top n-1 disks from A->B]
Tower(n-1,a,c,b)

Step 3:- print”disc-n from A->C”

Step 4:- [Move n-1 disks from B->C]
tower(n-1,b,a,c)

Step 5):- return

Tree For N=2



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

Postfix Evaluation Algorithm

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

Infix Notation and Prefix Notation

- 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 **prefix**ed 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 and Algorithm

- 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**.

Postfix Evaluation Algorithm

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

Evaluation of Postfix & prefix Expression

- After converting infix to postfix, we need postfix evaluation algorithm to find the correct answer.
- From the postfix expression, when some operands are found, pushed them in the stack. When some operator is found, two items are popped from the stack and the operation is performed in correct sequence. After that, the result is also pushed in the stack for future use. After completing the whole expression, the final result is also stored in the stack top.

Evaluation of expression Algorithm

Input: Postfix expression to evaluate.
Output: Answer after evaluating postfix form.

A postfix expression can be evaluated using the Stack data structure. To evaluate a postfix expression using Stack data structure we can use the following steps...

1. Read all the symbols one by one from left to right in the given Postfix Expression
2. If the reading symbol is operand, then push it on to the Stack.
3. If the reading symbol is operator (+ , - , * , / etc.,), then perform TWO pop operations and store the two popped oparands in two different variables (operand1 and operand2). Then perform reading symbol operation using operand1 and operand2 and push result back on to the Stack.
4. Finally! perform a pop operation and display the popped value as final result.

Parul®
University

DIGITAL LEARNING CONTENT

Links for conversion of Expressions

1. Infix to postfix
https://www.youtube.com/watch?v=vq-nUF0G4fI

2. Infix to Prefix coversion
https://www.youtube.com/watch?v=UK16ttNfGSk

3.Evaluation of postfix and prefix expressions
https://www.youtube.com/watch?v=MeRb_1bddWg

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in