Lecture 14-15

- The Stack

  Last-In-First-Out (LIFO)

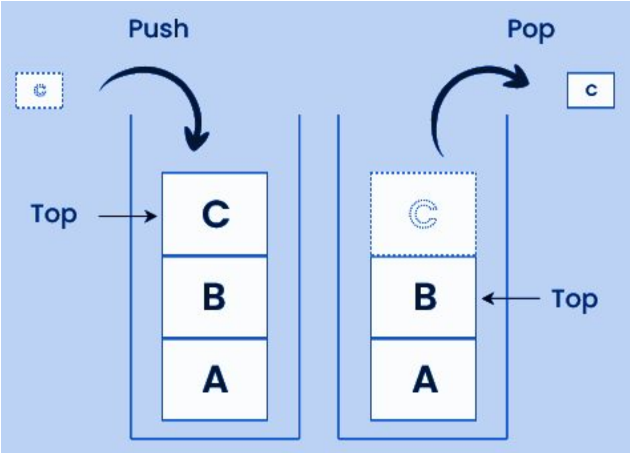
IT205: Data Structures (AY 2023/24 Sem II Sec B) — **Dr. Arpit Rana**

for stack read notebook also

Definition

A Stack is an

- **ordered collection** of
  It means stack is a indexed and a element come after another element
- **homogeneous** data elements
- where the insertion (a.k.a. **push**) and deletion (a.k.a. **pop**) operations take place **at one end** called the **top** of the stack.



The maximum number of elements that a stack can accommodate is termed as **SIZE** of the stack.

stack is linear data type because it is made from array and linklist

if in stack size is 100 but element fill out at 25 then top point at index 25

Representation of a Stack

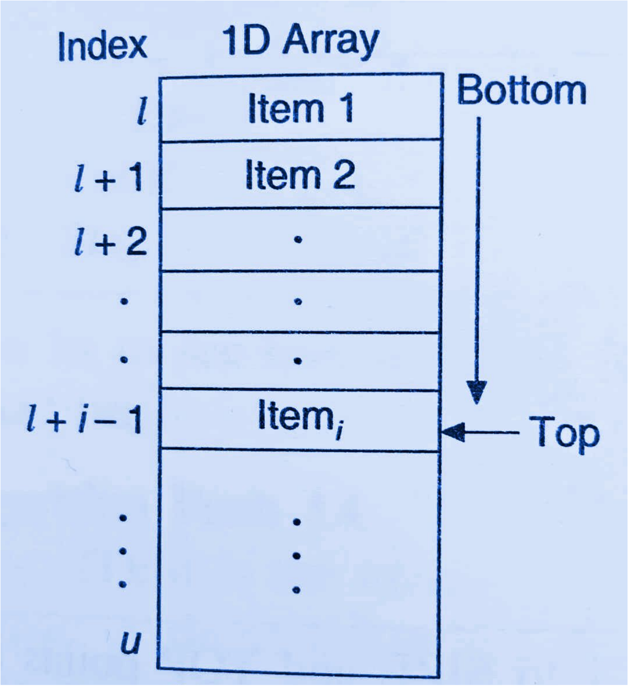
Most commonly, a Stack can be represented in the memory in two ways:

- Using a one-dimensional array
- Using a single linked list

Array Representation of a Stack

First, a block of sufficient size to accommodate the full capacity of the stack is allocated.

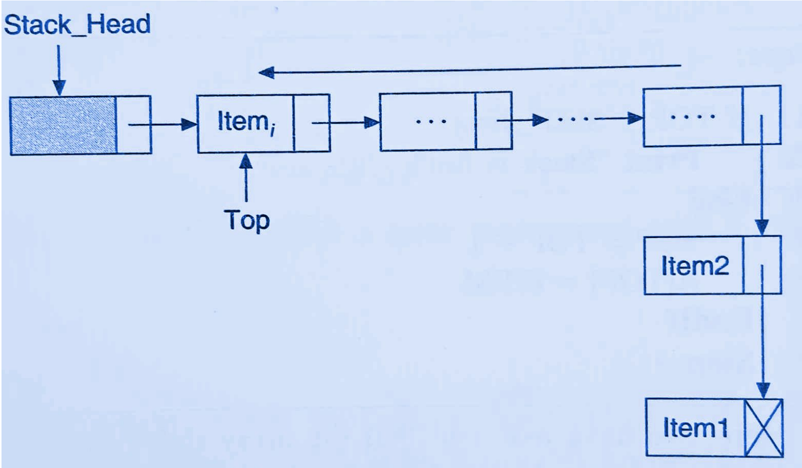
- Starting from the first location, i.e., l, items are stored up to the upper bound, i.e., u.
- Top is a pointer to point the position of the array up to which the stack is filled.
- Stack is *empty* if $Top < l$ and is *full* when $Top >= u$.



Linked Representation of a Stack

Array representation of stack is easy and convenient but **allows only fixed sized stacks**. The solution is linked representation.

- ***Top*** is a pointer to point the current node, i.e., the first node of the list.
- Thus, the ***push*** adds a node in the front and ***pop*** deletes from the front of the list.
- In this representation of stack, **size** is not important as it is dynamic in nature.



Operations on Stacks

The basic operations required to manipulate a stack are:

- ***Push:*** to insert an item onto a stack
- ***Pop:*** to remove an item from the stack
- ***Status***: to know the present state of a stack

Operations on Stacks

Input: A stack "A" and a new element ITEM to be pushed onto it
Output: A stack after inserting an element **on the top of the stack**

Steps to Push onto a Stack (Array):

```
If TOP >= SIZE then
   Print "Stack is full"
Else
   TOP = TOP + 1
   A[TOP] = ITEM
EndIf
Stop
```

Steps to Push onto a Stack (Linked List):

```
p = getnode()
/* Insert at front */
Data(p) = ITEM
Link(p) = TOP
TOP = p
Link(STACK_HEAD) = TOP
Stop
```

Operations on Stacks

Input: A Stack with elements
Output: Removes an item from the top of the stack if it is not empty

Steps to Pop from a Stack (Array):

```
If TOP < l then
    Print "Stack is empty"
Else
    ITEM = A[TOP]
    TOP = TOP - 1
EndIf
Stop
```

Steps to Pop from a Stack (Linked List):

```
If TOP == NULL
    Print "Stack is empty"
    Exit
Else
    p = Link(TOP)
    ITEM = Data(TOP)
    Link(STACK_HEAD) = p
    TOP = p
Endif
Stop
```

Operations on Stacks

Input: A Stack with elements
Output: Status whether it is empty or full, available free space, and the item on its Top

Steps to find the Status of the Stack (Array):

```
If TOP < l then
    Print “Stack is empty”
Else
    If TOP >= SIZE then
        Print “Stack is full”
    Else
        Print “The element at TOP is”, A[TOP]
        free = (SIZE - TOP)/SIZE * 100
        Print “Percentage of free stack is .”, free
    Endif
Endif
Stop
```

Operations on Stacks

Input: A Stack with elements
Output: Status whether it is empty or full, available free space, and the item on its Top

Steps to find the Status of the Stack (Linked List):

```
p = Link(STACK_HEAD)
If p == NULL then
    Print "Stack is empty"
Else
    nodeCount = 0
    While (p != NULL) do
        nodeCount = nodeCount + 1
        p = Link(p)
    EndWhile
    Print "Front Item: ", Data(TOP), "Stack has ", nodeCount, "items"
Endif
Stop
```

Applications of Stack: Balanced Parenthesis Problem

Consider a mathematical expression that includes several sets of nested parentheses. We want to ensure that the parentheses are nested correctly.

- There are equal number of right and left parenthesis
- Every right parenthesis is preceded by a matching left parenthesis

The following are the examples of invalid expressions:

```
((A+B) or A+B(
)A+B(-C or (A+B))-(C+D
```

here we use stack data structure because of his nature . for checking parenthesis are balanced or not we should check latest or recent parenthesis of given expression. in stack we know that we push first that come last out . so this nature of stack usefull for checking that the given expression is balanced or not.

parenthesis is used to give precedence of expression

Applications of Stack: Balanced Parenthesis Problem

At any point of expression -

- **Nesting depth** is the number of opening scopes that have not yet closed.
- **Parenthesis count** is the number of left parenthesis minus the number of right parenthesis (should be nonnegative).

Expression	`((A+B)  or`	`A+B(`
Parenthesis Count	`122221`	`0001`
	`)A+B(-C or`	`(A+B))-(C+D`
	`-1`	`11110-1`

Applications of Stack: Balanced Parenthesis Problem

Steps to find whether the Given String is Valid

```
valid = True
stk = an empty stack
symb = the first character of the string
While (symb != '\0') do
    If (symb == '(' || symb == '{' || symb == '[') then
        push(stk, symb)
    If (symb == ')' || symb == '}' || symb == ']') then
        If (Top(stk) < 0) then
            valid = False
            break
        contd…
```

write this algorithm in form of parenthesis count.

stack7.cpp

using push and pop
stack5.cpp and stcak4.cpp
with use of stl-1>stack11.cpp

if we use parenthesis count in this algorithm we should not use stack;

Applications of Stack: Balanced Parenthesis Problem

Steps to find whether the Given String is Valid

```
        Else
            i = pop(stk)
            If (i is not the matching opener of symb) then
                valid = False
                break
        EndIf
    EndIf
    symb = the next input character of the string
EndWhile
If (Top(stk) > 0) then
    valid = False
return valid
```

Applications of Stack: Postfix Expression

Consider the algebraic expression of sum of X and Y -

- `X + Y` — Infix notation
- `XY+` — Postfix/ Reverse Polish notation
- `+XY` — Prefix/ Polish notation

Postfix has a number of advantages over infix for expressing algebraic formulas.

- First, any formula can be expressed without parenthesis.
- Second, it is very convenient for evaluating formulas on computers with stacks.
- Third, infix operators have precedence, e.g., we know that a * b + c means (a * b) + c and not a * (b + c), because multiplication has been defined to have precedence over addition. Postfix eliminates this nuisance.

Applications of Stack: Postfix Expression

Consider the five binary operators: addition (+), subtraction (--), multiplication (*), division (/), and exponentiation (^) with the precedence:

() > ->^ > * / > + -

- Exponentiation > Multiplication/Division > Addition/Subtraction

and the associativity **for exponentiation is right to left** while **for others it is left to right**.

Infix	Postfix
A+B	AB+
A+B-C	AB+C-
(A+B)*(C-D)	AB+CD-*
A^B*C-D+E/F/(G+H)	AB^C*D-EF/GH+/+

Applications of Stack: Evaluating Postfix Expression

Steps to Evaluate Postfix Expression

```
opndstk = an empty stack
symb = the first character of the string
While (symb != '\0') do
     If (symb is an operand) then
          push(opndstk, symb)
     Else     //symb is an operator
          opnd2 = pop(opndstk)
          opnd1 = pop(opndstk)
          value = result of applying symb to opnd1 and opnd2
          push(opndstk, value)
     EndIf
EndWhile
return pop(opndstk)
```

If it is operand

stack9.cpp

Applications of Stack: Evaluating Postfix Expression

Postfix Expression: `6 2 3 + - 3 8 2 / + * 2 ^ 3 +`

symb	opnd1	opnd2	value	opndstk
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
−	6	5	1	1
3	6	5	1	1,3
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2
/	8	2	4	1,3,4
+	3	4	7	1,7
*	1	7	7	7
2	1	7	7	7,2
$	7	2	49	49
3	7	2	49	49,3
+	49	3	52	52

Exercises

Convert the following infix expressions into postfix

Infix	Postfix
A+B*C	ABC*+
(A+B)*C	AB+C*
((A+B)*C-(D-E))^(F+G)	AB+C*DE--FG+^
A-B/(C*D^E)	ABCDE^*/-

Applications of Stack: Converting Infix Expression to Postfix

code written in vs code stack6.cpp

Consider the following infix expression:

Step	Infix	Postfix	tempstk
	A + B * C		
1	↑ (A)	A	
2	↑ (+)	A	+
3	↑ (B)	AB	+
4	↑ (*)	AB	+ *
5	↑ (C)	ABC	+ *
6		ABC*	+
7		ABC*+	

Precedence plays an important role in converting infix expressions into postfix!

only check that if precendce of operator > top stack top then push into stack other wise pop

Applications of Stack: Converting Infix Expression to Postfix

We define a precedence function as follows:

Function Header: `boolean prcd(op1, op2)`

Type of arguments: `op1` and `op2` are characters representing algebraic operators

Return: `True` if precedence of `op1 > op2` when `op1` appears to the left of `op2`, `False` otherwise.

Examples: `prcd('*', '+')` and `prcd('+', '+')` are `True`
`prcd('+', '*')` and `prcd('^', '^')` are `False`

use precendece function that we use in code stack6.cpp

Applications of Stack: Converting Infix Expression to Postfix

Steps to Convert Infix Expression into Postfix

```
opstk = an empty stack
symb = the first character of the string
While (symb != '\0') do
    If (symb is an operand) then
        add symb to the postfix string
    Else    //symb is an operator
        While(!empty(opstk) && prcd(stacktop(opstk), symb))do
            topsymb = pop(opstk)
            add topsymb to the postfix string
        EndWhile
        push(opstk, symb)
    EndIf
EndWhile
```

contd...

stack6.cpp

Applications of Stack: Converting Infix Expression to Postfix

```
//output any remaining operators
While(!empty(opstk))do
    topsymb = pop(opstk)
    add topsymb to the postfix string
EndWhile
Stop
```

In this algorithm, we assumed that the infix expression has no parentheses.

What changes do we need to make to accommodate the parentheses?

Applications of Stack: Converting Infix Expression to Postfix

We define a precedence function as follows:

Function Header: `boolean prcd(op1, op2)`

Type of arguments: `op1` and `op2` are characters representing algebraic operators

Return: `True` if precedence of `op1 > op2` when `op1` appears to the left of `op2`, `False` otherwise.

We also add the following rules in our definition:

```
prcd('(', op) = False        //for any operator op
prcd(op, '(') = False        //for any operator op other than ')'
prcd(op, ')') = True         //for any operator op other than '('
prcd(')', op) = Undefined    //for any operator op
```

code:stack8.cpp

Applications of Stack: Converting Infix Expression to Postfix

Symbol	Out stack pre	in Stack pre
+, −	1	2
*, /	3	4
^	6	5
(	7	0
)	0	?

Steps to Convert Infix Expression into Postfix

```
opstk = an empty stack
symb = the first character of the string
While (symb != '\0') do
    If (symb is an operand) then
        add symb to the postfix string
    Else    //symb is an operator
        While(!empty(opstk) && prcd(stacktop(opstk), symb))do
            topsymb = pop(opstk)
            add topsymb to the postfix string
        EndWhile
        push(opstk, symb)
    EndIf
EndWhile
```

```
If(empty(opstk) || symb != ')') then
    push(opstk, symb)
Else
    topsymb = pop(opstk)
```

contd...

Applications of Stack: Evaluating Postfix Expression

infix

Postfix Expression: `((A - (B + C)) * D) $ (E + F)` // $ → ^

symb	*postfix string*	*opstk*
(		(
(		((
A	A	((
−	A	((−
(	A	((−(
B	AB	((−(
+	AB	((−(+
C	ABC	((−(+
)	ABC+	((−
)	ABC+−	(
*	ABC+−	(*
D	ABC+−D	(*
)	ABC+−D*	
$	ABC+−D*	$
(	ABC+−D*	$(
E	ABC+−D*E	$(
+	ABC+−D*E	$(+
F	ABC+−D*EF	$(+
)	ABC+−D*EF+	$
	ABC+−D*EF+$	

Other Applications of Stack

- Implementation of Recursion
- Implementation of Quicksort (we will study it later)
- Solving Tower of Hanoi Problem
- Activation Record Management (scope rules)

Next Lecture

- Queues