Module 4

Stack

Linear Data Structures

- **Linear Data Structures:** Data elements form a sequence or a linear list. The data is arranged in a linear fashion although the way they are stored in the memory need not to be sequential
  - Array
  - Linked List
  - Stack
  - Queue

Stack

- A linear data structure used for storing data
- Items are inserted and removed at one end
    - LIFO (Last-In-First-Out) principle.

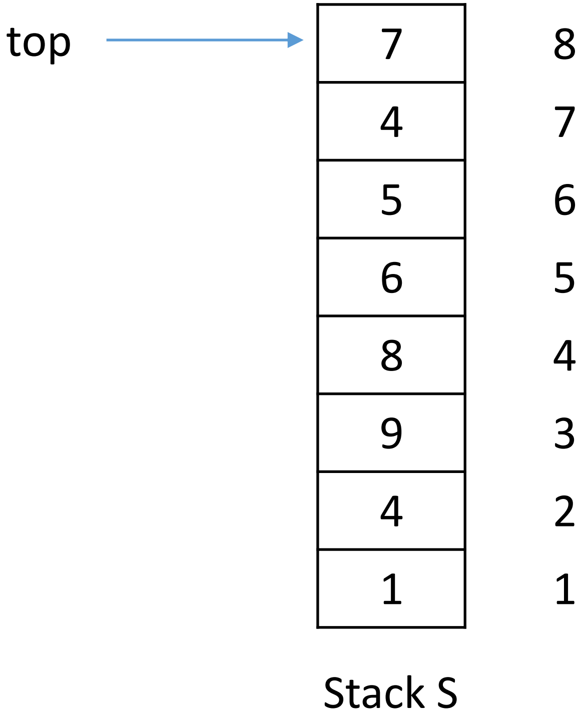
Restriction

- The last element inserted is the first one to be removed
- Example:
    - Pile (stack) of plates
    - Pile (stack) of books

Stack Representation and Operations

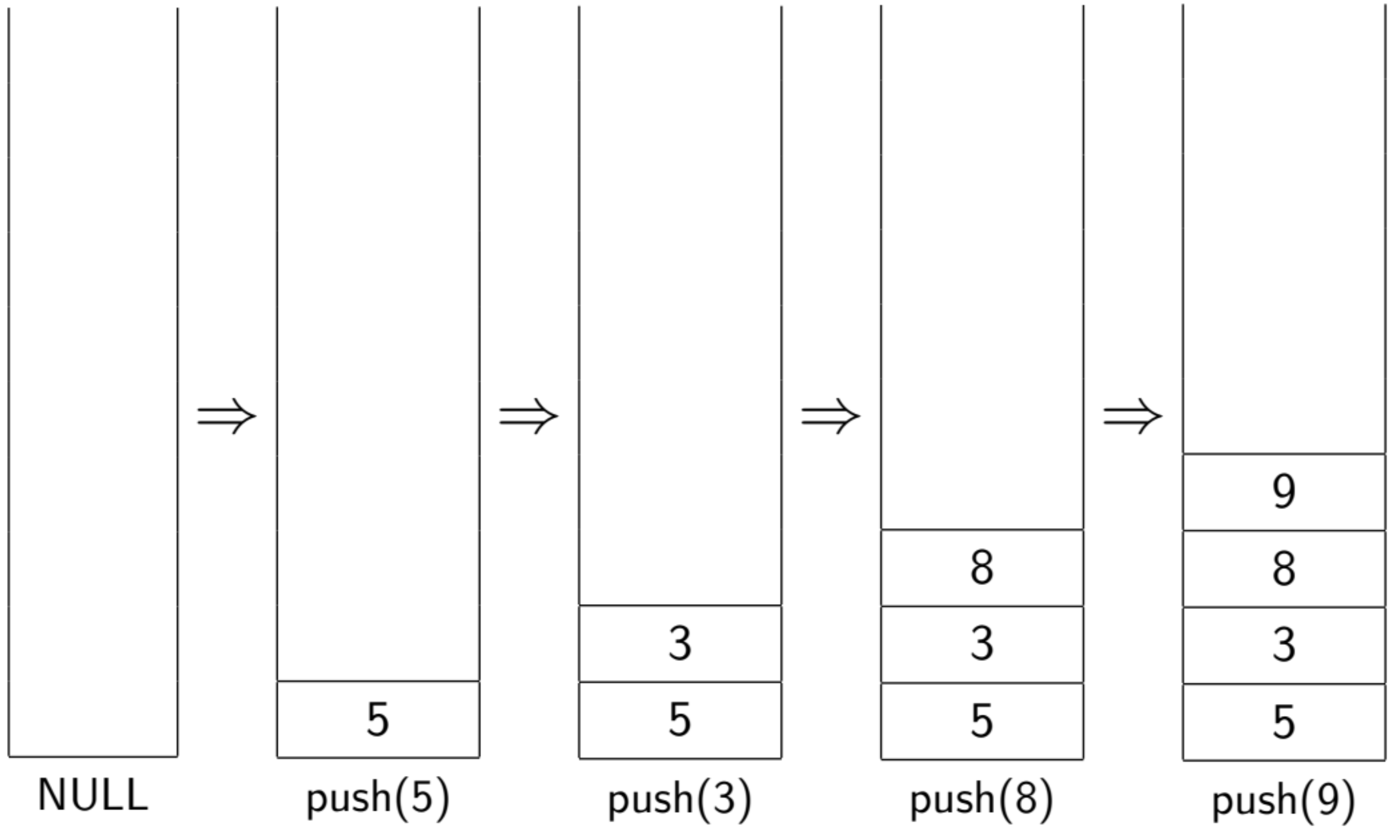
- Abstract Data Type
  - push(x): Insert element x on top of the stack
  - pop: Remove and return top element from the stack

- Additional Operations:
  - SIZE(): Returns the number of elements in Stack (stack size)
  - STACK-EMPTY(): Returns a Boolean indicating if the Stack is empty
  - TOP-ELEMENT(): Returns the top element on the stack (without removing it)



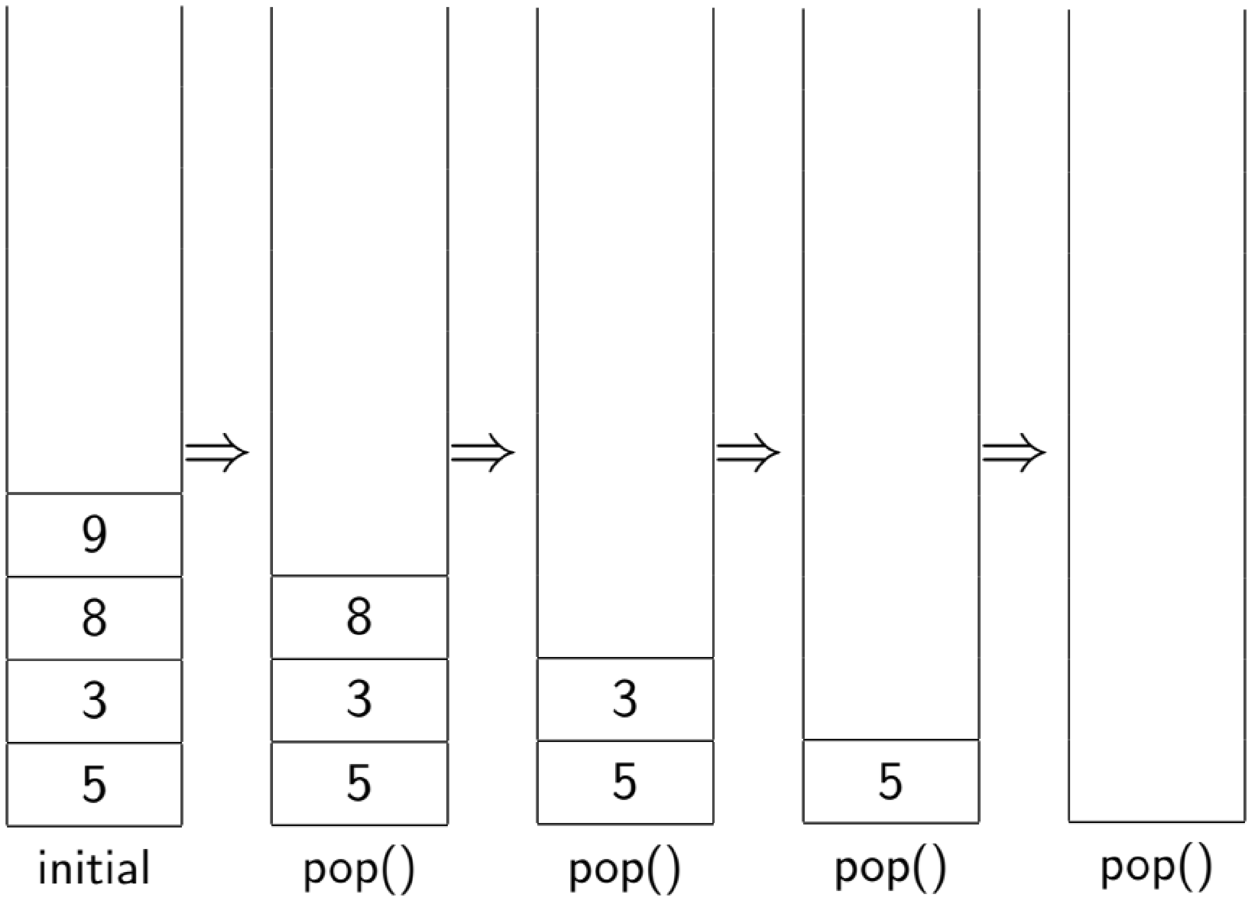
Stack Working Example: push(x)

- Increment top variable
- Insert element at top position
- Once the stack is full, the push(x) operation will throw OVERFLOW error



Stack Working Example: pop()

- Remove element from top position
- Decrement top variable
- Once the Stack is empty, the pop() operation will throw UNDERFLOW error



Stack Applications

- Function calls in a program (or recursion)
- Implement undo/redo operations
- Balanced parentheses in source code
- Expression conversion and evaluation
- String reversal

Implementation of Stack

- Using Arrays
- Using Linked Lists

- **Constraints to keep in mind**
  - Insertion and removal operations to be performed from top only
  - Complexity of above operations is $O(1)$

Stack Implementation using Arrays

- Use array to store stack elements
- *top* variable stores an array index
- If stack is empty → *top* = 0
- If stack is full → *top* = MAX-SIZE
- Push only till array is not full (*top* < MAX-SIZE)
- Pop only when top > 0

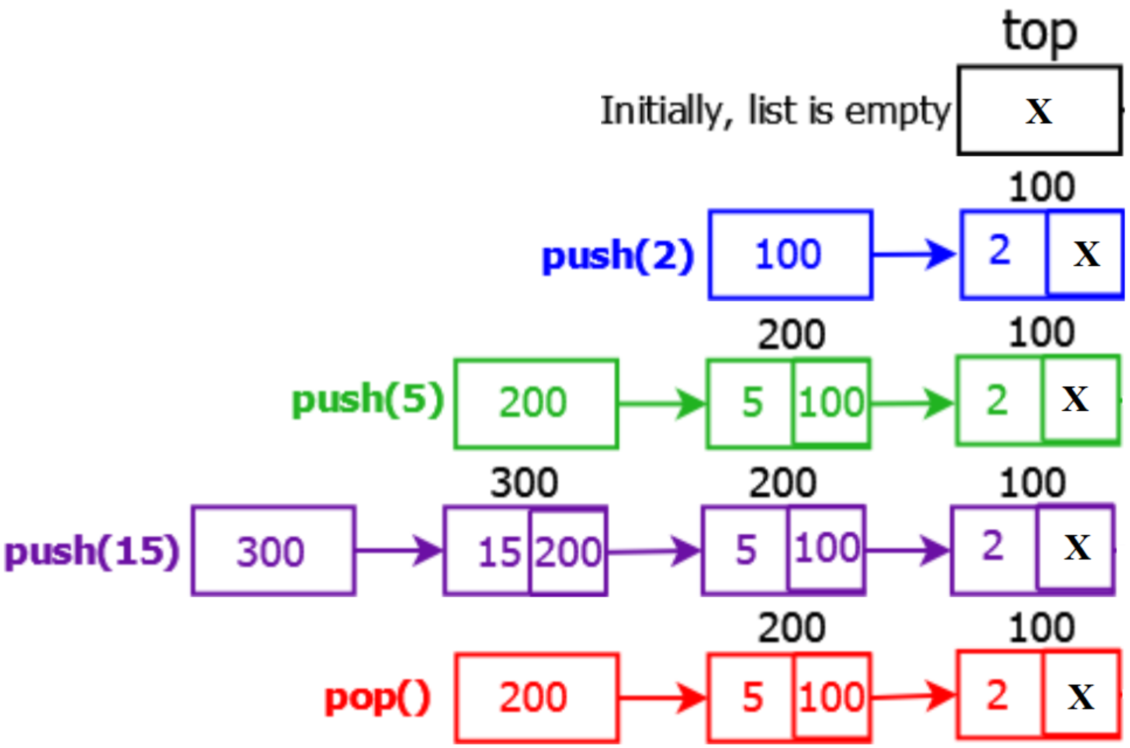
```
PUSH (S,x)
if top == MAX-SIZE
        error OVERFLOW
else
        top = top + 1
        S[top] = x
```

```
POP()
if top == 0
        error UNDERFLOW
else
        top = top - 1
        return S[top + 1]
```

- Time Complexity: $\Theta(1)$

Stack Implementation using Linked Lists

- Contiguous block of memory is not required
- Insertion and deletion operations at the end of the list costs $O(n)$ time
  - Perform both *push*(*x*) and *pop*() operations at the beginning of the list $O(1)$



Polish Notations

- Arithmetic Expressions: A + B, A∗B, A=B, A + B ∗C - D∗E

Infix Notations: <Operand> <Operator> <Operand>
Prefix Notations: <Operator> <Operand> <Operand>
Postfix Notations: <Operand> <Operand> <Operator>

- **Order of Operations:**
  Parentheses: (), {}, []
  Exponents: ↑ right to left
  Multiplication and Division: left to right
  Addition and Subtraction: left to right

Polish Notations

Infix	**Prefix**	**Postfix**
$A + B$	$+AB$	$AB+$
$A * B$	$*AB$	$AB*$
$A + (B * C)$	$+A * BC$	$ABC * +$
$(A + B) * (C - D)$	$* + AB - CD$	$AB + CD - *$
$A + (B * C) \uparrow D$	$+A \uparrow *BCD$	$ABC * D \uparrow +$

Example- Infix to Postfix without ()

$a + b * c - d / e * h$

Input	Stack	Postfix Expression
a		a
$+$	$+$	a
b	$+$	ab
$*$	$+*$	ab
c	$+*$	abc
$-$	$-$	$abc*+$
d	$-$	$abc*+d$
$/$	$-/$	$abc*+d$
e	$-/$	$abc*+de$
$*$	$-*$	$abc*+de/$
h	$-*$	$abc*+de/h$
		$abc*+de/h*-$

Converting Notations with ()

- **Additional Rules**
  - Push an opening parenthesis to the Stack
  - Pop all elements including opening parenthesis as soon as you get a closing parenthesis in the expression
  - Append the operators in the postfix expression string (excluding the parenthesis)
  - Perform *Infix to Postfix* operations

Example- Infix to Postfix with ()

$(a + b \uparrow c \uparrow d) * (e + (f/g))$

Input	Stack	Postfix
$($	$($	
a	$($	a
$+$	$(+$	a
b	$(+$	ab
$\uparrow$	$(+\uparrow$	ab
c	$(+\uparrow$	abc
$\uparrow$	$(+\uparrow\uparrow$	abc
d	$(+\uparrow\uparrow$	$abcd$
$)$		$abcd\uparrow\uparrow+$
$*$	$*$	$abcd\uparrow\uparrow+$

Input	Stack	Postfix
$($	$*($	$abcd\uparrow\uparrow+$
e	$*($	$abcd\uparrow\uparrow+e$
$+$	$*(+$	$abcd\uparrow\uparrow+e$
$($	$*(+($	$abcd\uparrow\uparrow+e$
f	$*(+($	$abcd\uparrow\uparrow+ef$
$/$	$*(+(/$	$abcd\uparrow\uparrow+ef$
g	$*(+(/$	$abcd\uparrow\uparrow+efg$
$)$	$*(+$	$abcd\uparrow\uparrow+efg/$
$)$	$*$	$abcd\uparrow\uparrow+efg/+$
		$abcd\uparrow\uparrow+efg/+*$

Infix to Prefix Conversion

- Reverse the expression
- Use postfix conversion algorithm
- Reverse the output
  - Push all characters to stack one by one
  - Pop all characters back once the expression is empty

Example- Infix to Prefix

$(a + b \uparrow c) * d + e \uparrow f$

Input	Stack	Prefix Expression
f		f
$\uparrow$	$\uparrow$	f
e	$\uparrow$	fe
$+$	$+$	$fe \uparrow$
d	$+$	$fe \uparrow d$
$*$	$+*$	$fe \uparrow d$
$)$	$+*)$	$fe \uparrow d$
c	$+*)$	$fe \uparrow dc$
$\uparrow$	$+*) \uparrow$	$fe \uparrow dc$
b	$+*) \uparrow$	$fe \uparrow dcb$
$+$	$+*)+$	$fe \uparrow dcb \uparrow$
a	$+*)+$	$fe \uparrow dcb \uparrow a$
$($	$+*$	$fe \uparrow dcb \uparrow a+$
		$fe \uparrow dcb \uparrow a + *+$

reverse the expression: $+ * +a \uparrow bcd \uparrow ef$

Postfix to Infix Conversion

- Push the operand to the stack
- If the next element is an operator then pop two operands from the stack and place the operator between them
- Push the resultant string back to the stack
- Repeat

Example: postfix to infix conversion

Postfix:	abcd↑↑+efg/+*
Input	**Stack**
a	a
b	a, b
c	a, b, c
d	a, b, c, d
↑	a, b, (c ↑ d)
↑	a, (b ↑ (c ↑ d))
+	(a + (b ↑ (c ↑ d)))
e	(a + (b ↑ (c ↑ d))), e
f	(a + (b ↑ (c ↑ d))), e, f
g	(a + (b ↑ (c ↑ d))), e,f, g
/	(a + (b ↑ (c ↑ d))), e, (f/g)
+	(a + (b ↑ (c ↑ d))), (e + (f/g))
*	(a + (b ↑ (c ↑ d))) * (e + (f/g))
Minimal Backets	(a + b ↑ c ↑ d) * (e + f/g)

Example : postfix to infix conversion

$1\ 2\ 3\ 2 \uparrow \uparrow + 5\ 15\ 3 / + *$

Input	Stack
1	1
2	1, 2
3	1, 2, 3
2	1, 2, 3, 2
$\uparrow$	1, 2, 9
$\uparrow$	1, 512
+	513
5	513, 5
15	513, 5, 15
3	513, 5, 15, 3
/	513, 5, 5
+	513, 10
*	5130

Prefix to Infix Conversion

- Reverse the input (prefix) expression
- Push the operand to the stack
- If the next element is an operator then pop two operands from the stack and place the operator between them
- Push the resultant string back to the stack
- Repeat

Example: prefix to infix conversion

Prefix:	**+*+a↑bcd↑ef**
Reverse the expression	**fe↑dcb↑a+*+**
Input	**Stack**
f	f
e	f, e
↑	(f ↑ e)
d	(f ↑ e), d
c	(f ↑ e), d, c
b	(f ↑ e), d, c, b
↑	(f ↑ e), d, (c ↑ b)
a	(f ↑ e), d, (c ↑ b), a
+	(f ↑ e), d, ((c ↑ b) + a)
*	(f ↑ e), (d * ((c ↑ b) + a))
+	(f ↑ e)+(d * ((c ↑ b) + a))
Reverse the expression	((a+(b↑c))*d)+(e↑f)
Minimal Backets	(a+b↑c)*d+e↑f

References

- Saymour L., **“Data Structures”,** Schaum’s Outline Series, McGraw Hill, Revised First Edition
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, **“Introduction to Algorithms”**, The MIT Press
- Sahni, S., **“Data Structures, Algorithms, and Applications in C++”,** WCB/McGraw-Hill