Data Structures & Algorithms

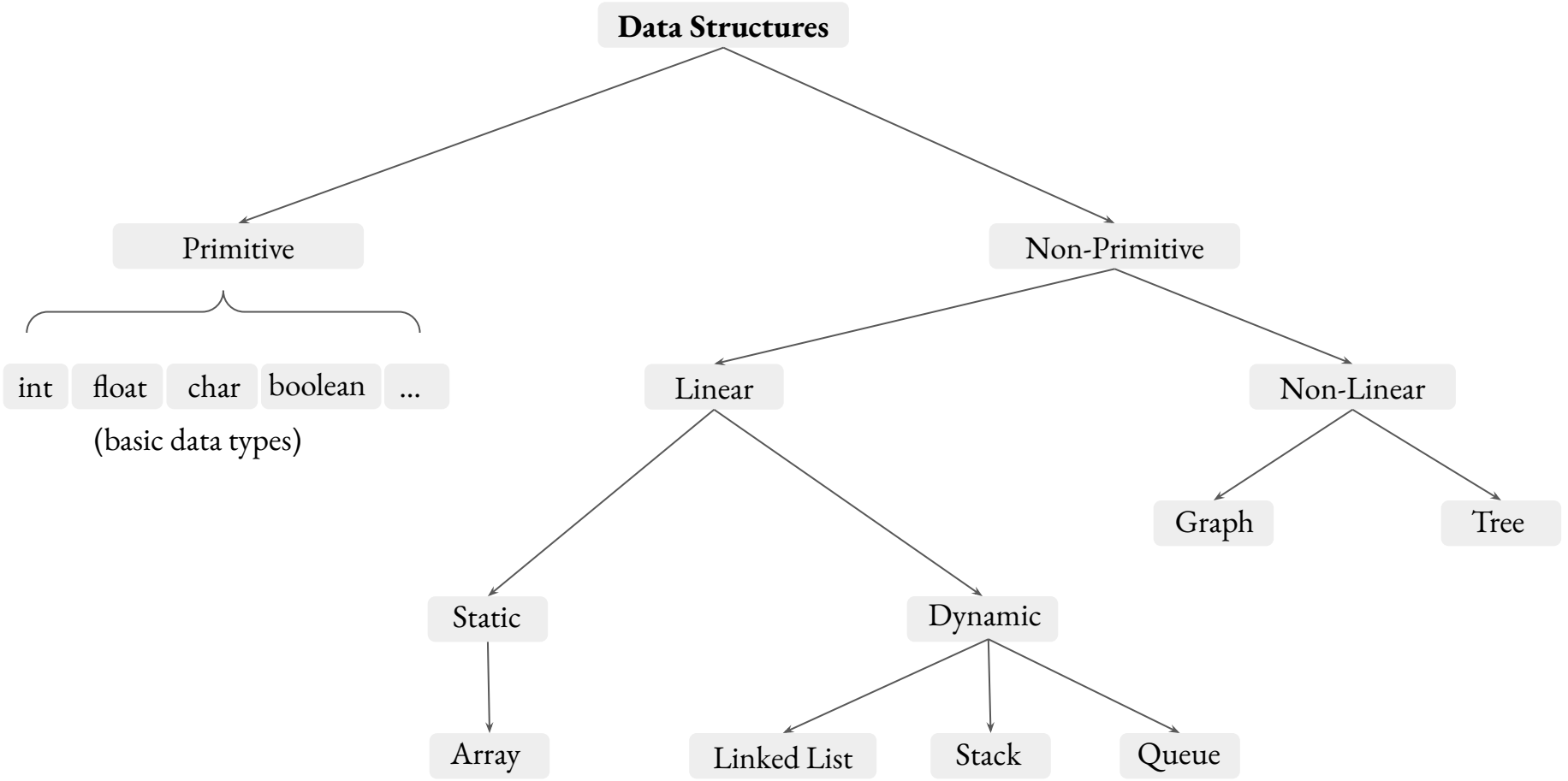
02: Stacks

Dr Ram Prasad Krishnamoorthy

Associate Professor
School of Computing and Data Science

ram.krish@saiuniversity.edu.in

Stacks
(LIFO: Last In First Out)

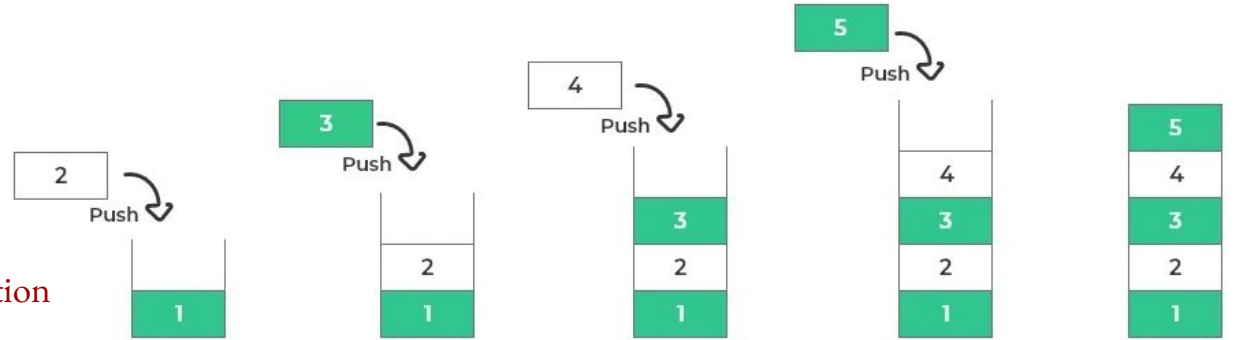

Data Structures
Primitive
Non-Primitive
int
float
char
boolean
...
(basic data types)
Linear
Non-Linear
Graph
Tree
Static
Dynamic
Array
Linked List
Stack
Queue

Stacks

Stack: A dynamic dataset in which the element deleted is the one that was most recently inserted.

- Stacks use a last-in, first-out, or **LIFO**, policy

Push → Insert Operation

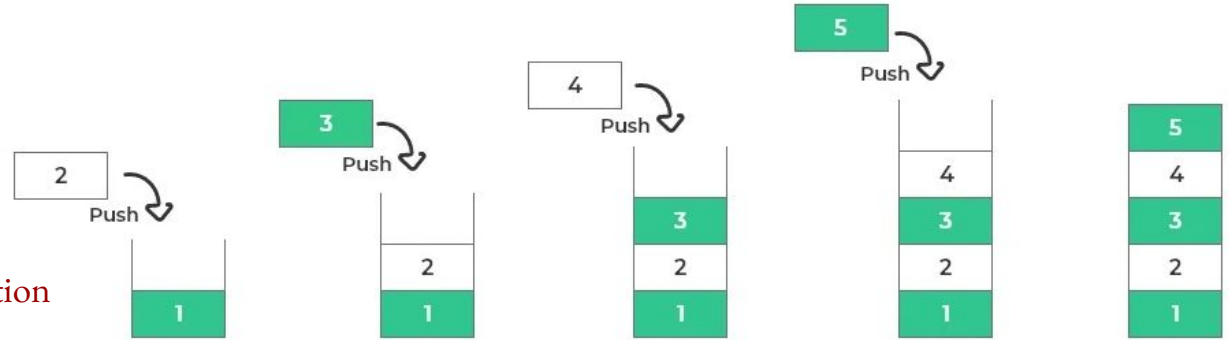


Stacks

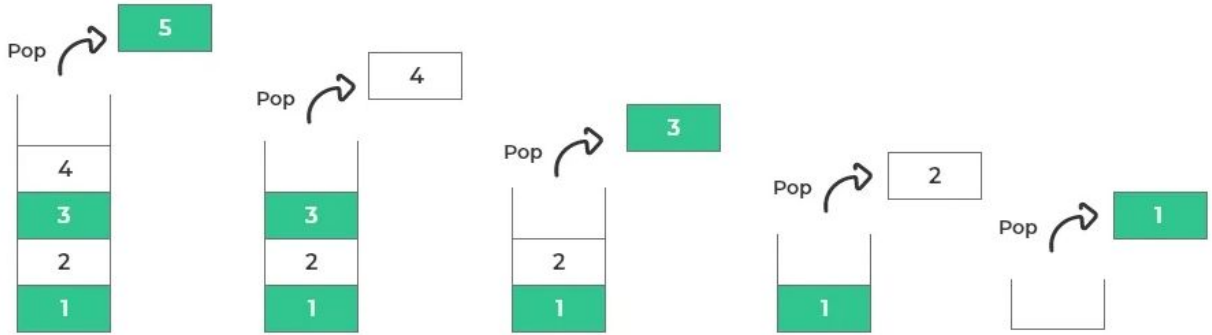
Stack: A dynamic dataset in which the element deleted is the one that was most recently inserted.

- Stacks use a last-in, first-out, or **LIFO**, policy

Push → Insert Operation



Pop → Delete Operation



Stacks

Here are some key points about **stacks**:

- **Linear data structure**: Elements are arranged in a single line, like a chain.
- **Limited access point**: You can only access and modify elements at the **top** of the stack.
- **Basic operations**:
  - **Push**: Adds an element to the top of the stack

    *Pushing onto a full stack causes an **overflow**.*
  - **Pop**: Removes and returns the top element.

    *Popping an empty stack causes **underflow**.*
  - **Peek**: Returns the top element without removing it.
  - **IsEmpty**: Checks if the stack is empty.

Real-World Examples of Stacks

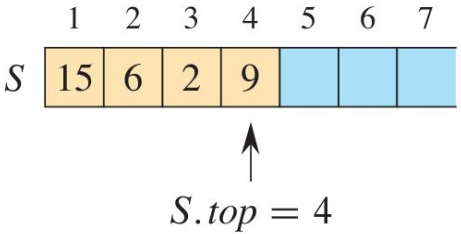
Stacks have various applications:

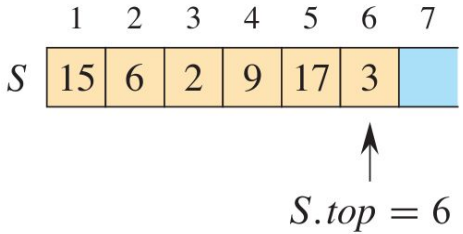
- **Undo/Redo functionality:** In text editors, a stack keeps track of changes, allowing undo and redo actions.
- **Function calls in programming:** When a function is called, its parameters and local variables are pushed onto a stack. When the function returns, these elements are popped off.
- **Browsing history:** The back and forward buttons in your web browser use a stack to manage visited pages.
- **Expression evaluation:** Evaluating expressions in calculators or compilers often involves using a stack to handle operands and operators.

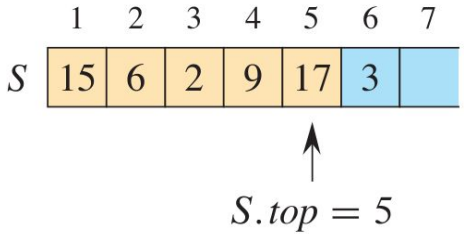
Pseudocode for Stack operations

Stacks

Stack

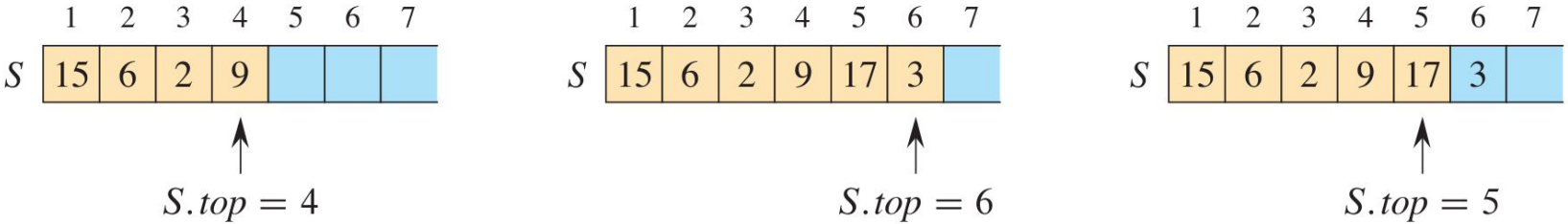






Stacks

Stack

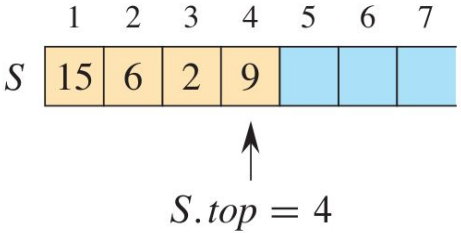


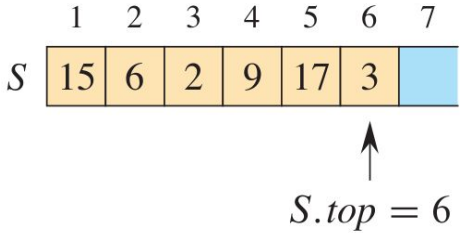
Stack operations: PUSH, POP, STACK-EMPTY.

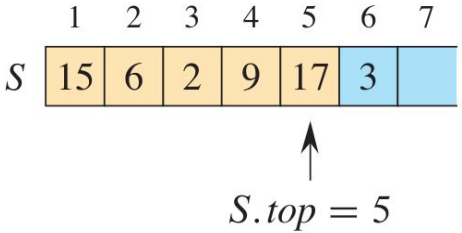
```
PUSH(S, x)
  if S.top == S.size
       error "overflow"
  else S.top = S.top + 1
       S[S.top] = x
```

Stacks

Stack







Stack operations: PUSH, POP, STACK-EMPTY.

PUSH(S, x)

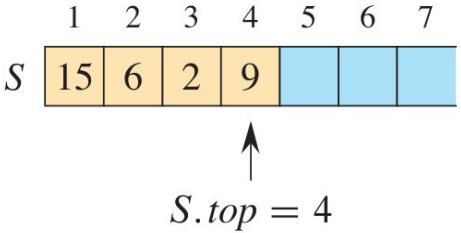
if $S.top$ == $S.size$
  error "overflow"
else $S.top = S.top + 1$
  $S[S.top] = x$

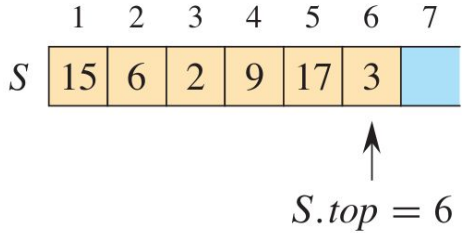
POP(S)

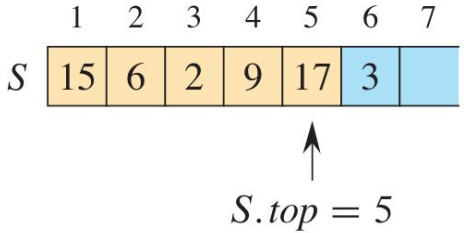
if STACK-EMPTY(S)
  error "underflow"
else $S.top = S.top - 1$
  return $S[S.top + 1]$

Stacks

Stack







Stack operations: PUSH, POP, STACK-EMPTY.

PUSH(S, x)
  if $S.top$ == $S.size$
    error "overflow"
  else $S.top = S.top + 1$
    $S[S.top] = x$

POP(S)
  if STACK-EMPTY(S)
    error "underflow"
  else $S.top = S.top - 1$
    return $S[S.top + 1]$

STACK-EMPTY(S)
  if $S.top$ == 0
    return TRUE
  else return FALSE

C Program
for
Stack operations
(Array implementation)