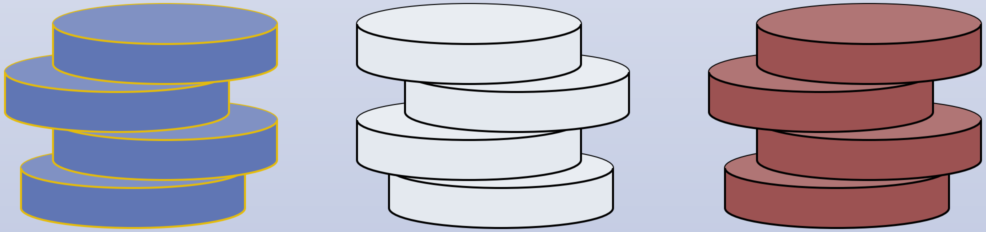
CS14: Summer 2019

Stacks



The Stack

- The concept of **stack** is derived from the metaphor of a stack of plates in a spring-loaded cafeteria dispenser.
- If you want to **remove** a plate, you **pop** the a plate off the **top** of stack.
- If you want to replace a plate or insert more plates, you **push** onto the **top** of the stack.
- To check if ALL plates were clean, you would need to check the top plate, remove that plate, and repeat the process until the entire stack was inspected.

Stack ADT

- **Definition:** A **stack** is a collection of objects that are inserted and removed according to the **last-in-first-out (LIFO)** principle.
    - Objects are **inserted** (as long as stack not full) onto the **top** of the stack.
    - Objects can **ONLY** be **removed** from the **top** of the stack.
- All stack operations are O(1).

Stack ADT

- Main **stack** operations:
  - ***push***(***o***) : inserts object ***o*** on top of stack
    - STL operation - ***push(o)***
  - ***pop***() : removes element from the top of the stack
    - STL operation - ***pop( )***
    - An error occurs if the queue is empty. (*exception*)
  - ***top()*** : examines the top object on the stack **without** removing it.
    - Used in combination with pop()
    - top() to inspect element, pop() to remove top element

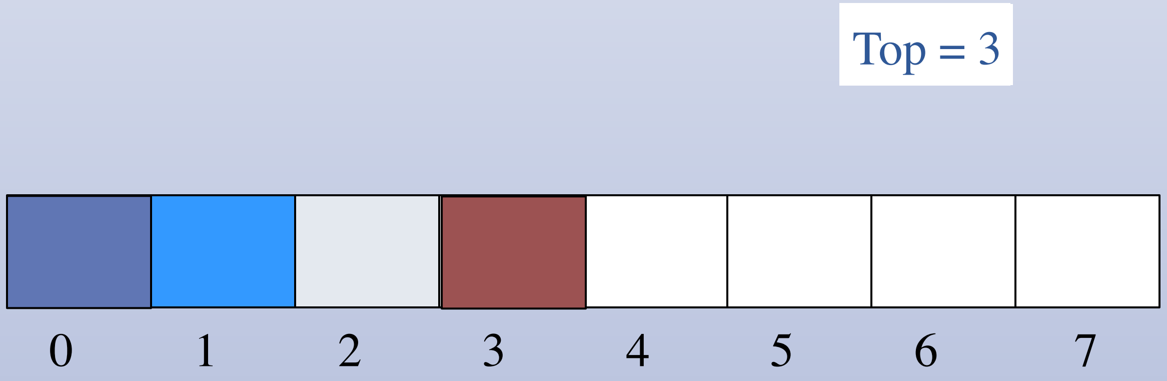
- Auxiliary **stack** operations:
  - ***size*( )** : returns the number of objects in a stack. Either store as a variable counter or calculate it.
  - ***isEmpty*( )** : returns **true** if the stack is empty, else **false**

Array-based Stack

- Store the elements in an N-element array ***S***
- Have an integer variable ***t*** that gives the index of the top element in the array ***S***
- The top element in the array ***S*** is stored in the cell ***S[t]***
- ***See an example…***

Array-based Stack

- A simple way of implementing the Stack ADT uses an array
- We push (add) elements from left to right
- A variable keeps track of the index of the last item pushed

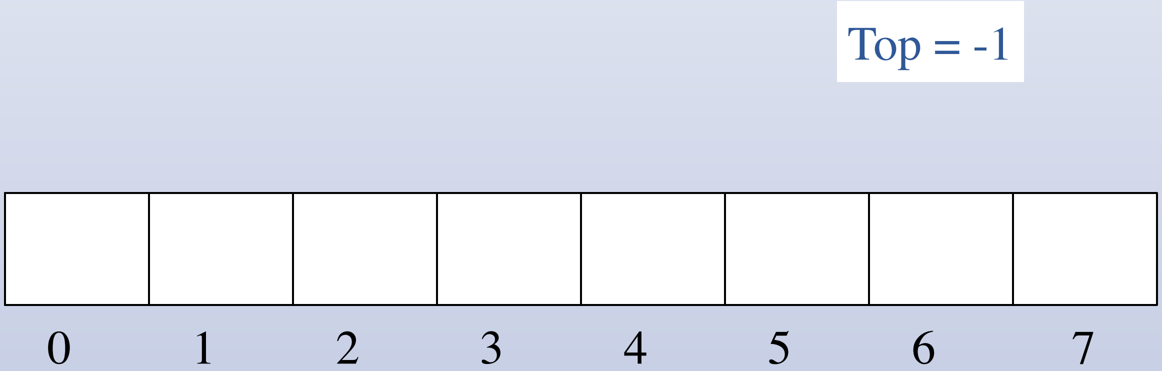


Top = 3

0 1 2 3 4 5 6 7

Array-based Stack

- We pop (remove) elements from right to left



Top = -1

0 1 2 3 4 5 6 7

Stack ADT Pseudocode

```
Algorithm size():
    return t+1
Algorithm isEmpty():
    return (t<0)
Algorithm top():
    if isEmpty() then
        throw a StackEmptyException
    return S[t]
```

Stack ADT Pseudocode

```
Algorithm push(o):
    if size()==N then
        throw a StackFullException
    t ← t+1
    S[t] ← o

Algorithm pop():
    if isEmpty() then
        throw a StackEmptyException
    t ← t-1
```

Evaluation

- Performance
  - Let n be the number of elements in the stack
  - The space used is $O(n)$
    - Space complexity = memory required - new!
  - Each operation runs in time $O(1)$
    - Time complexity = running time
- Limitations
  - The maximum size of the stack must be defined *a priori* , and cannot be changed
  - Trying to push a new element into a full stack causes an implementation-specific exception

Stack Data Structure

```
class Stack
{
private:
        objectType stack[MAX_STACK_SIZE];
        int top;
public:
        functions for stack manipulation
        constructor sets top to -1
};
```

Stack Implementation - Push

- The array storing the stack elements may become full
    - Limitation of the array-based implementation

```
void push ( const objectType & o )
{
        if ( top + 1 == MAX_STACK_SIZE )
                throw FullStackException
        else
                S[++top] = o;
}
```

Stack Implementation- Pop

- (Stop here) Quick exercise- write pop and getTop functions
    - Array may be empty when pop
    - getTop will return top item/object

```
void pop ( ) {
        if ( isEmpty ( ) )
                throw EmptyStackException
        else
                --top;
```

Stack Implementation- Top

- (Stop here) Quick exercise- write pop and getTop functions
  - Array may be empty when pop
  - getTop will return top item/object

```
objectType getTop ( ) {
        if ( isEmpty ( ) )
                throw EmptyStackException
        else
                return S[top];
```

Stack Applications

- Checking for balanced symbols in a program.

```
{
      {
      //}
}
```

- Evaluating postfix (**R**everse **P**olish **N**otation) expressions.
- Infix to Postfix expression conversion.
- Managing function calls in a program.

Reverse Polish Notation (RPN): Postfix

- Operators `*,/,+,-` follow their operands:
  - `3 + 8    (in infix)`
  - `3 8 +    (in postfix)`
- For expressions with multiple operands, operator occurs immediately after its second operand.
  - `40 4 5 * -      (in postfix)`
  - `40 (4*5) - , 40 20 - ,  40 - 20 ,  20`
- Eliminates need for parentheses to force operator precedence.
- Used widely for computation in early desktop calculators.

Algorithm PostfixEvaluation

```
Process infix expression one item (p) at a time, left-to-right
    if (p == operand) // examples: 5, 7, 77, 2
        push(p)
    if (p == operator) // examples: *, /, +, -
        top/pop and write to b
        top/pop and write to a
        push(a operator b)
```

Algorithm PostfixEvaluation

`3 * (5 + ((2 + 3) * 8) + 5) => 3 5 2 3 + 8 * + 5 + *`

Current Symbol	Stack
3	3
5	3 5
2	3 5 2
3	3 5 2 3
+	3 5 5

Algorithm PostfixEvaluation

```
3 * (5 + ((2 + 3) * 8) + 5) => 3 5 2 3 + 8 * + 5 + *
```

Current Symbol	**Stack**
8	3 5 5 8
*	3 5 40
+	3 45
5	3 45 5
+	3 50
*	150

Algorithm Infix2Postfix

```
Process infix expression one item (p) at a time, left-to-right
        if (p == operand) // examples: 5, 7, 77, 2
            write to output
        if (p == operator) // examples: *, /, +, -
            top/pop and write to output
            until top of stack is ( or item of lower precedence than p
            push(p)
        if (p == ‘( ‘)
            push(p)
        if (p == ‘) ’)
            top/pop and write to output
            until ( ,  pop but do NOT write (
        if (p == NULL) // p is empty
           top/pop and write to output
           until stack is empty
```

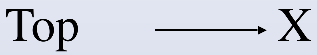
Linked List-based Stack

```
bool isEmpty ( ) {
   if ( top == NULL )
      return true;
   else
      return false;
}
```

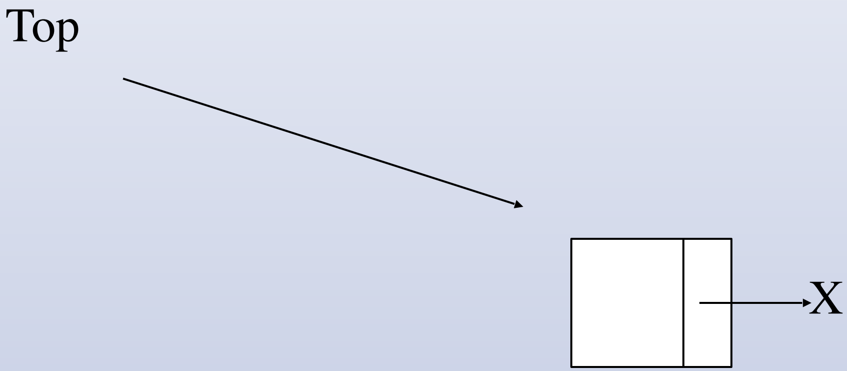
```
objectType getTop ( ) {
  if ( top )
     return top->obj;
  else
    throw stack_empty exception;
}
```

```
void push ( const objectType & obj ) {
     Node * newNode = new Node;
     newNode->obj = obj;
     newNode->next = top;
     top = newNode;
}
```

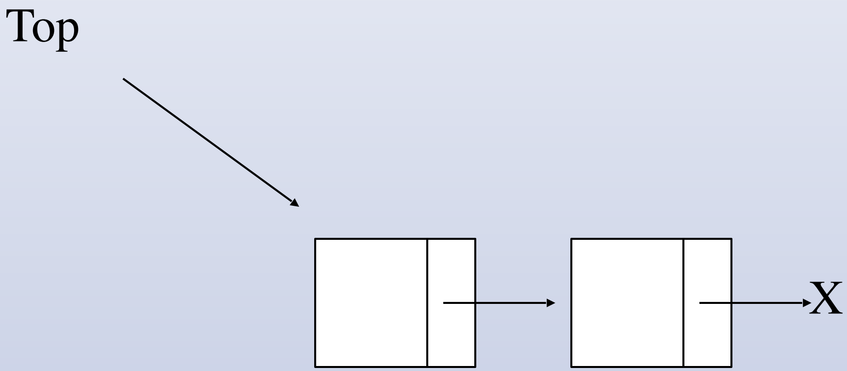
Linked List-based Stack



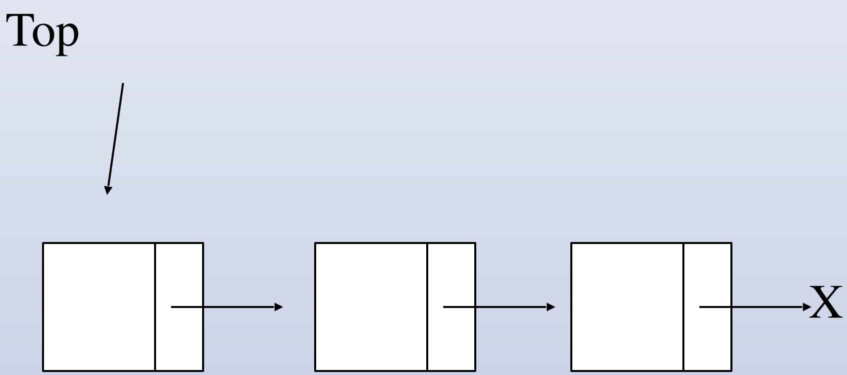
Linked List-based Stack



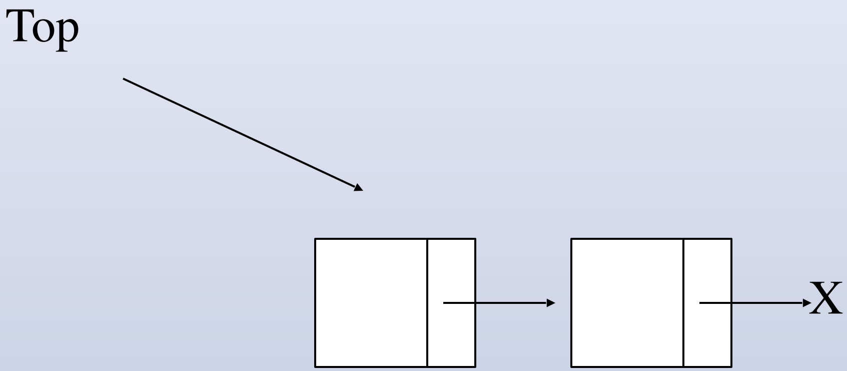
Linked List-based Stack



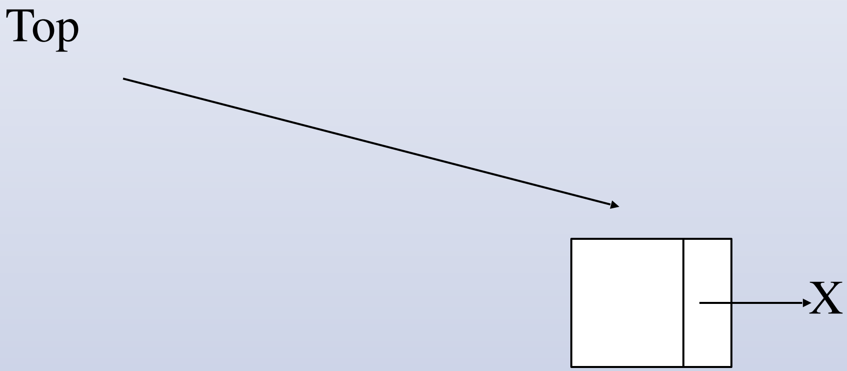
Linked List-based Stack



Linked List-based Stack



Linked List-based Stack



Linked List-based Stack