Data Structures

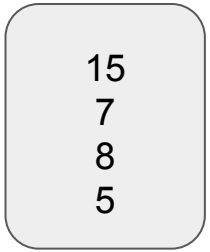
Array-based Stack

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Using an array

- Both the array and stack are great match!
- On right, we pushed in stack in order: 5, 8, 7, 15
- Simply, A corresponding array is {5, 8, 7, 15}
  - Then the last element of the array now 15 = its **peek**
  - To add a new element, 4, we **add** to end ⇒ {5, 8, 7, 15, 4}
  - If we want to **pop 2** elements ⇒ {5, 8, 7}



Index	0	1	2	3
Value	5	8	7	15

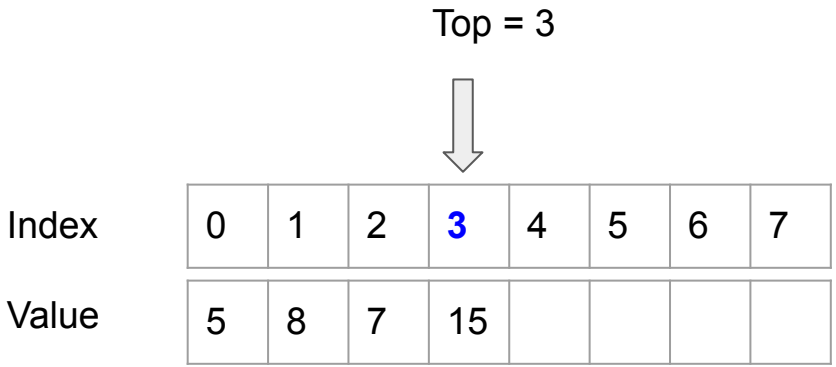
Design

- Internally, we need an array.
  - Let's make it **dynamic**
  - So we need an initial array with some initial size
- The last thing, we need to how many elements so far, e.g. length variable
  - Or let's call it top
    - It represents the index of last element
    - Then logically -1 if no elements

```
class Stack {
private:
    int size { };
    int top { };
    int* array { };
public:
    Stack(int size) :
            size(size), top(-1) {
        array = new int[size];
    }
```

Design

- Example: Let’s create a stack of 8 elements
- Now we pushed 4 elements, so top at 3
- We can push up to other 4 elements before stack is full!



Push, Pop and Peek

- These 3 methods are very direct
  - Recall: initially top = -1
- Push add, but we need to shift top first
  - ++top
- Pop: Top points to current element
  - So we need array[top]. Top-- to move step back
- Peek: Just return top
- Validation:
  - Our approach here to check and assert
  - More proper: **Throw an exception** with error msg

```
void push(int x) {
    assert(!isFull());
    array[++top] = x;
}

int pop() {
    assert(!isEmpty());
    return array[top--];
}

int peek() {
    assert(!isEmpty());
    return array[top];
}
```

Is Full & Is Empty

- Recall, top = -1 represents empty
- Full = just no more elements!
  - Top at last position
- To display, remember top elements represents our last stack element
  - So print reversely!

```
int isFull() {
    return top == size - 1;
}

int isEmpty() {
    return top == -1;
}

void display() {
    for (int i = top; i >= 0; i--)
        cout << array[i] << " ";
    cout << "\n";
}
```

Validation Choices

```
void push(int x) {
    assert(!isFull());
    array[++top] = x;
}
```

```
bool push(int x) {
    if(isFull())
        return false;
    array[++top] = x;
    return true;
}
```

```
void push(int x) {
    if(isFull())
        cout<<"Full Stack";
    else
        array[++top] = x;
}
```

- We can write our code in several ways!
- The first one assumes user MUST check, our we throw error (**acceptable**)
- The second way, guarantee the code will check (**acceptable** in industry)
- The 3rd print to console. This is ok only in educational context, not industry
  - Don’t print on console for user unless it is a console project
  - In reality, people see outputs on web or mobile screens!

Complexity

- Direct!
- Constructor, Destructor and Display are O(n) time
- All other operations are O(1) time
- Constructor is O(n) memory

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."