Simple Introduction to Data Structures and Stacks

in C/C++

Introduction - ADT versus Data Structure

Abstract Data Type (ADT): Any programmer defined data type. In other words, not a primitive data type (e.g. bool, int, char, float, double). Examples: string, struct, stack.

Data Structure: An ADT with strict rules (i.e. algorithm) for how data can be stored, retrieved, and manipulated. All data structures are ADTs, not all ADTs are data structures.

Introduction - ADT versus Data Structure

```
// this is an ADT,
// also a Data Structure
class IntArray{
    public:
        IntArray();
        ~IntArray();
        int find(int);
        int insert(int);
        int sort(int);
    private:
        int size;
        int *array;
};
```

```
// this is an ADT
struct Data {
    int id;
    char *information;
};
```

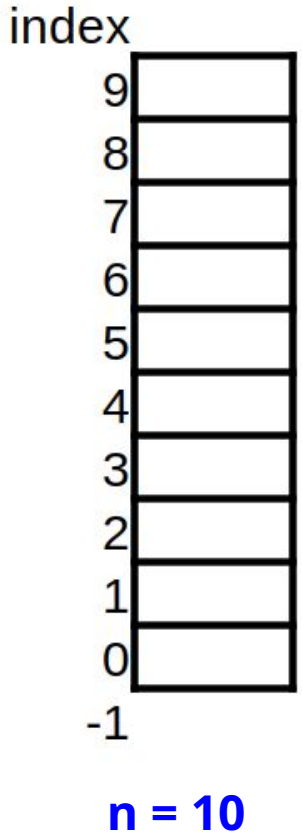
Introduction - Stack

Stack: A simple data structure; basically, “an array with rules.” Also, known as a last-in-first-out (LIFO) queue. The rules are push, pop, peek, isEmpty.

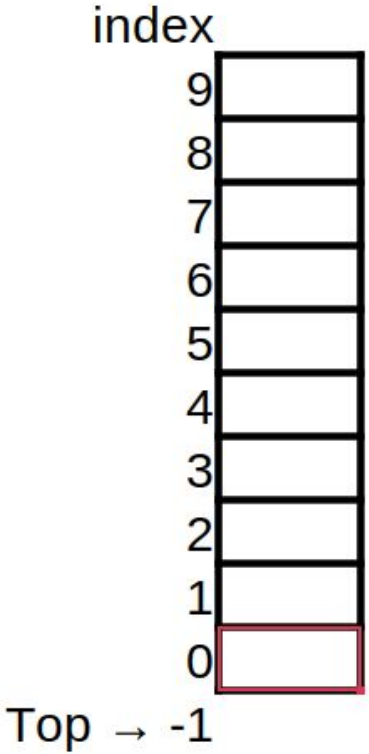
Concept

Imagine an array vertically.

The index goes from 0 to **n-1**, where **n** is the array size (e.g. 10). Take note of the fact that index value -1 is out of bounds, and not possible. We will use this fact.



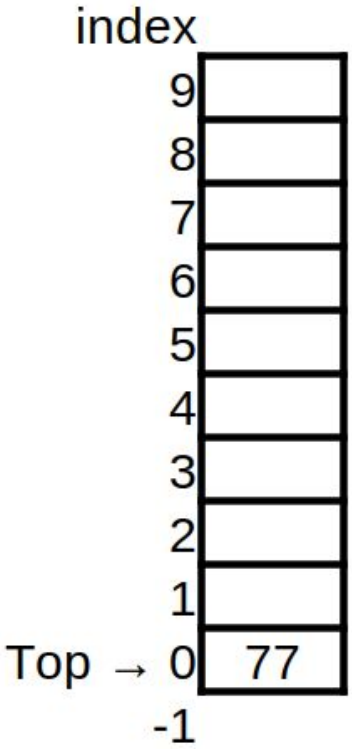
Rules - Introduction



When something is put “on the stack” it must go to the *lowest* open position. We call this spot the **top**. When the stack is first created, we set top = -1 to indicate the stack is “empty.”

When something is removed from the stack, it must come from the “top.”

Operations - Introduction

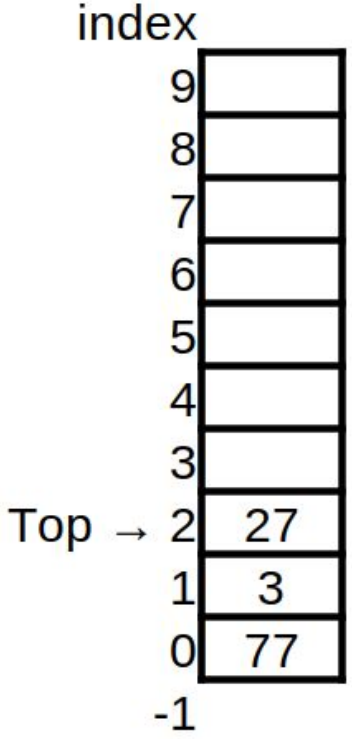


Imagine our stack holds integers.

```
#define SIZE 10;
int stack[SIZE];
```

If we put the number 77 on the stack, it goes into top+1 and that becomes the new top. i.e. **top = top + 1;**

Operations - Push

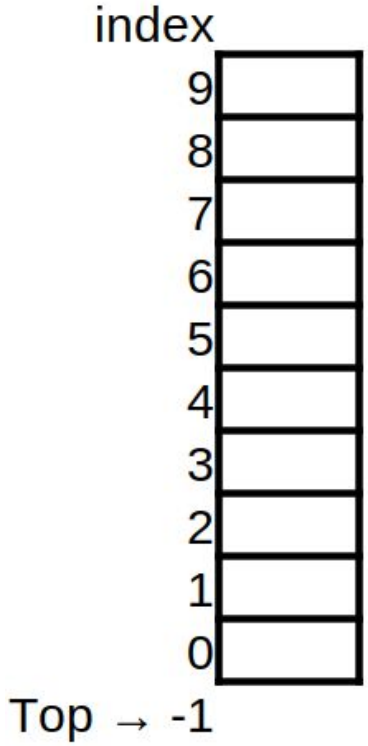


The act of putting something on the stack is called *push*.
The algorithm is:

- check that top is less than SIZE-1
- if true, add 1 to top, place the item in the stack
- if false, return false.
  - this is known as *overflow*

imagine we pushed 3 and 27 in that order, after 77

Operations - Pop

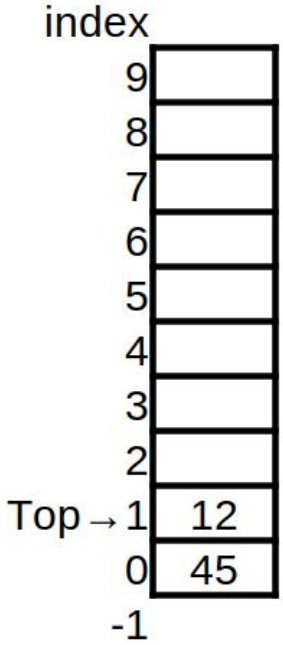


Pop is the reverse of push.
The algorithm is:

- check that top is greater than -1
- if true, remove the item from the stack *(i.e. return it somehow)*, decrement top by 1
- if false, indicate error somehow.
    - this is known as *underflow*

imagine we popped three times from the previous example

Operations - Peek

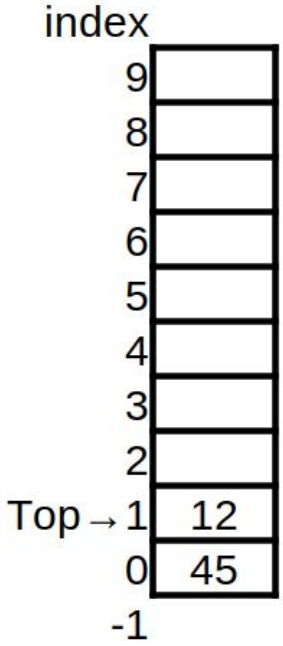


Peek is identical to pop, except the item is not removed, *but it **is** returned.* The algorithm is:

- check that top is greater than -1
- if true, return top item somehow
- if false, indicate error somehow.
    - this is known as *underflow*

if we peek the example stack to the left, it “returns” 12 and does nothing else

Operations - isEmpty



isEmpty checks if the stack is empty or not. The algorithm is:

- check that top < 0
- if true, return true
- if false, return false

the stack to the left would return false for isEmpty

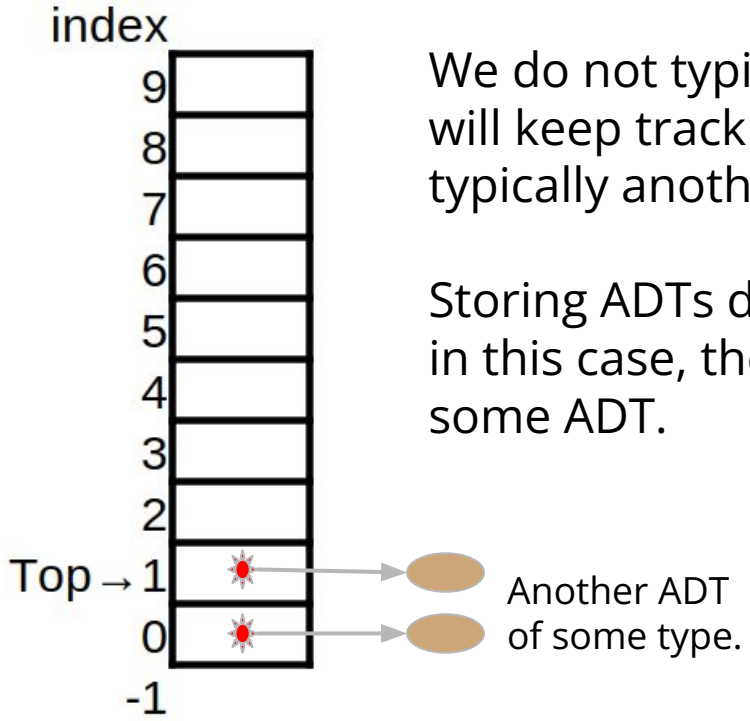
Abstract Data Type - The int Stack class

```
#define STACKSIZE 10

class Stack {
public:
    Stack();  // constructor
    ~Stack();  // destructor
    int pop();
    int peek();
    bool push(int);
    bool isEmpty();
private:
    int top;
    int stack[STACKSIZE];
};
```

```
// alternate versions
// pop and peek
bool pop(int*);
bool peek(int*);
```

The Real Stack ADT - Pointers to other ADTS



We do not typically make int stacks. A stack will keep track of something more complex, typically another ADT.

Storing ADTs directly is usually not efficient, so in this case, the stack becomes pointers to some ADT.