

1

STACKS

CS 250 – C++ Programming 2

DATA STRUCTURES

- **Data structure**

- A specific way to store and organize data in a computer so that it can be used efficiently.

- An **array** is a data structure.

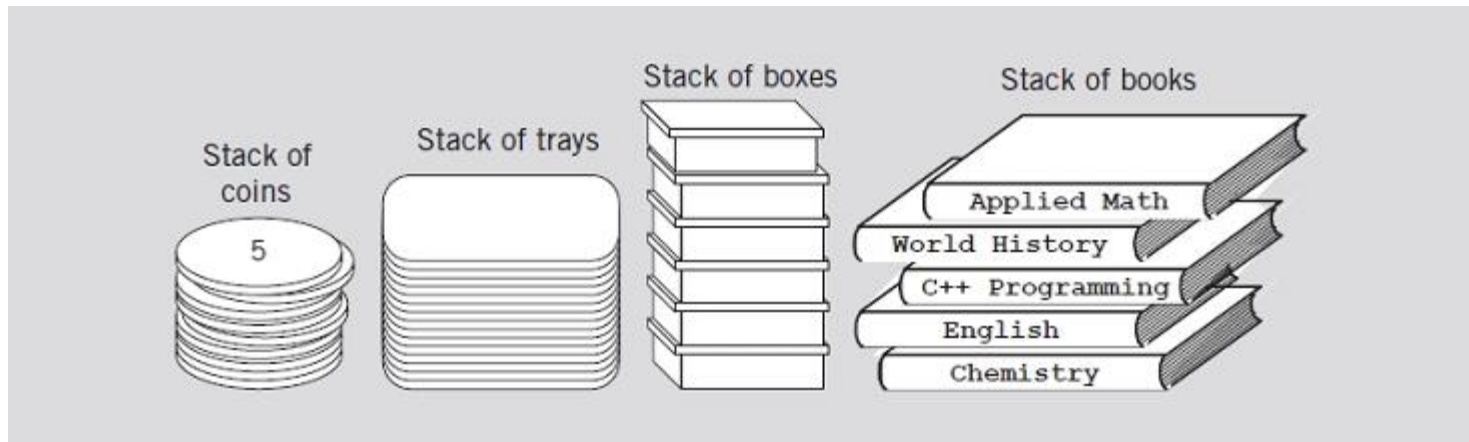
- We will look at two very common **data structures**:

- **Stacks**
- **Queues**

STACKS

◦ Stack data structure

- Elements are **added** and **removed** from **one end only**: the **top** of the **stack**
- Last In First Out (**LIFO**)



Various examples of stacks

STACK OPERATIONS

Operation	What it does
push(obj)	Inserts a new element at the top of the stack.

STACK OPERATIONS

Operation	What it does
push(obj)	Inserts a new element at the top of the stack.
pop()	Removes the element at the top of the stack.

STACK OPERATIONS

Operation	What it does
push(obj)	Inserts a new element at the top of the stack.
pop()	Removes the element at the top of the stack.
empty()	Returns true if the stack is empty , and returns false otherwise.

STACK OPERATIONS

Operation	What it does
push(obj)	Inserts a new element at the top of the stack.
pop()	Removes the element at the top of the stack.
empty()	Returns true if the stack is empty , and returns false otherwise.
top()	Retrieves (<u>without</u> removing) the element at the top of the stack.

STACK OPERATIONS

Operation	What it does
push(obj)	Inserts a new element at the top of the stack.
pop()	Removes the element at the top of the stack.
empty()	Returns true if the stack is empty , and returns false otherwise.
top()	Retrieves (<u>without</u> removing) the element at the top of the stack.
size()	Returns the number of elements in the stack.

STL STACK

- The **Standard Template Library (STL)** provides a **class** to implement a **stack**.
 - It is a **template** class

```
#include <stack>

...

stack<int> intStack;           // creates a stack of integers

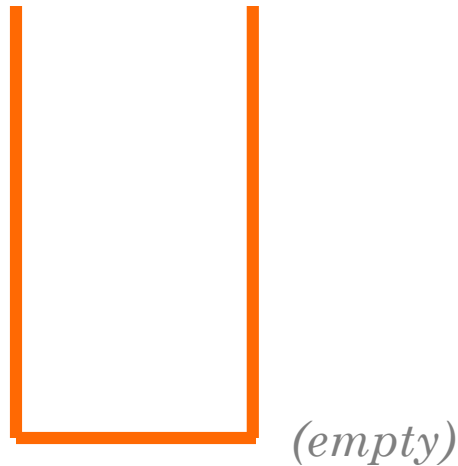
stack<string> stringStack;    // creates a stack of strings
                             // need to include <string>
```

TRACING CODE

We will create a **stack** of **integers**, **myStack**

```
stack<int> myStack;
myStack.push(1);
myStack.push(2);
myStack.push(3);
if (!myStack.empty())
    myStack.pop();
myStack.push(4);
while (!myStack.empty())
{
    cout << myStack.top() << " ";
    myStack.pop();
}
```

TRACING CODE



This is our **stack** of **integers** (now empty).

```
stack<int> myStack;
myStack.push(1);
myStack.push(2);
myStack.push(3);
if (!myStack.empty())
    myStack.pop();
myStack.push(4);
while (!myStack.empty())
{
    cout << myStack.top() << " ";
    myStack.pop();
}
```

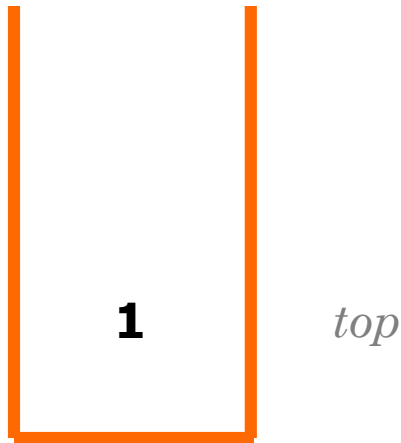
TRACING CODE



We **push** integer **1** into the **stack**.

```
stack<int> myStack;  
myStack.push(1);  
myStack.push(2);  
myStack.push(3);  
if (!myStack.empty())  
    myStack.pop();  
myStack.push(4);  
while (!myStack.empty())  
{  
    cout << myStack.top() << " ";  
    myStack.pop();  
}
```

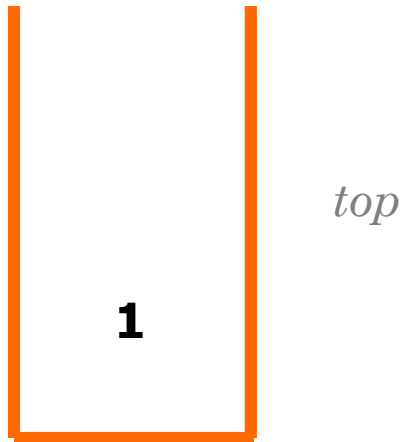
TRACING CODE



We **push** integer **1** into the **stack**.

```
stack<int> myStack;
myStack.push(1);
myStack.push(2);
myStack.push(3);
if (!myStack.empty())
    myStack.pop();
myStack.push(4);
while (!myStack.empty())
{
    cout << myStack.top() << " ";
    myStack.pop();
}
```

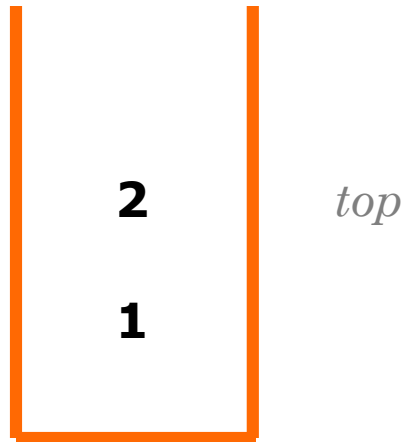
TRACING CODE



We **push** integer **2** into the **stack**.

```
stack<int> myStack;
myStack.push(1);
myStack.push(2);
myStack.push(3);
if (!myStack.empty())
    myStack.pop();
myStack.push(4);
while (!myStack.empty())
{
    cout << myStack.top() << " ";
    myStack.pop();
}
```

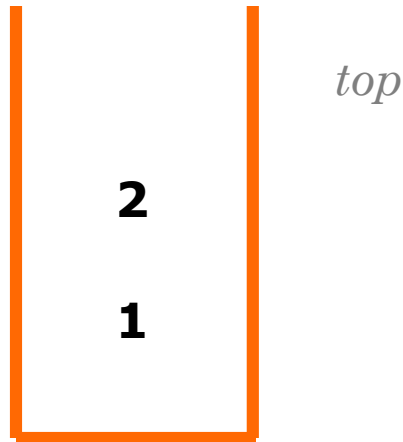
TRACING CODE



We **push** integer **2** into the **stack**.

```
stack<int> myStack;  
myStack.push(1);  
myStack.push(2);  
myStack.push(3);  
if (!myStack.empty())  
    myStack.pop();  
myStack.push(4);  
while (!myStack.empty())  
{  
    cout << myStack.top() << " ";  
    myStack.pop();  
}
```

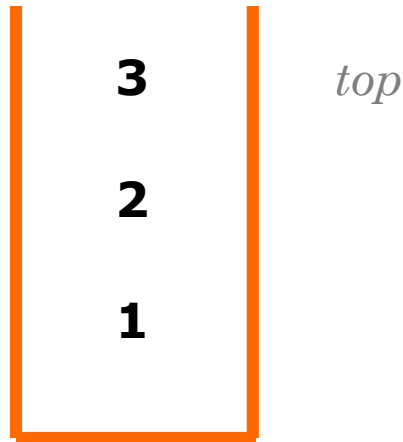
TRACING CODE



We **push** integer **3** into the **stack**.

```
stack<int> myStack;
myStack.push(1);
myStack.push(2);
myStack.push(3);
if (!myStack.empty())
    myStack.pop();
myStack.push(4);
while (!myStack.empty())
{
    cout << myStack.top() << " ";
    myStack.pop();
}
```

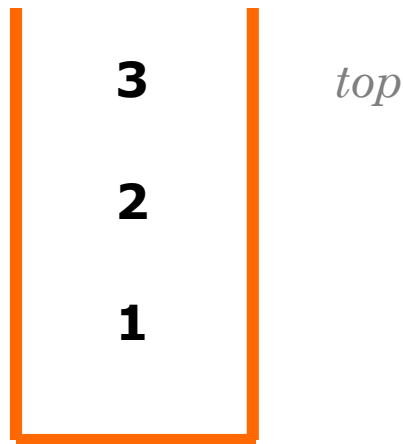

TRACING CODE



We **push** integer **3** into the **stack**.

```
stack<int> myStack;
myStack.push(1);
myStack.push(2);
myStack.push(3);
if (!myStack.empty())
    myStack.pop();
myStack.push(4);
while (!myStack.empty())
{
    cout << myStack.top() << " ";
    myStack.pop();
}
```

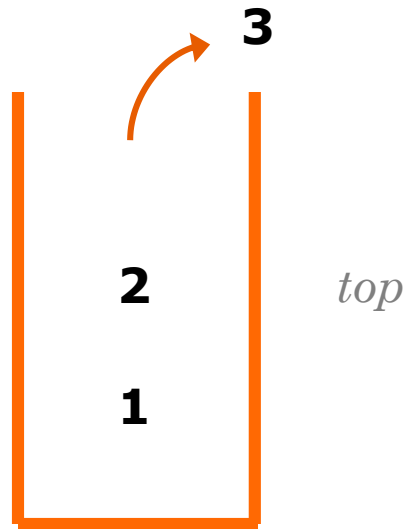
TRACING CODE



The **IF** statement is **TRUE** when the **stack** is **not** empty.

```
stack<int> myStack;  
myStack.push(1);  
myStack.push(2);  
myStack.push(3);  
if (!myStack.empty())  
    myStack.pop();  
myStack.push(4);  
while (!myStack.empty())  
{  
    cout << myStack.top() << " ";  
    myStack.pop();  
}
```

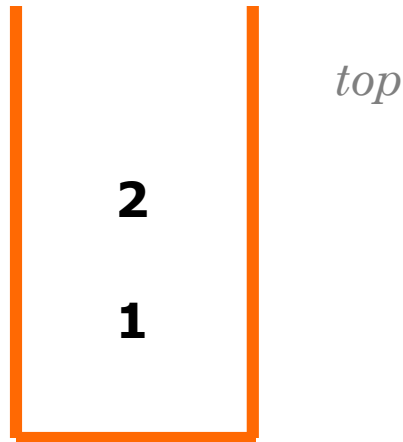
TRACING CODE



We **pop** the **top** element from the **stack** (**no** return value when popping).

```
stack<int> myStack;  
myStack.push(1);  
myStack.push(2);  
myStack.push(3);  
if (!myStack.empty())  
    myStack.pop();  
myStack.push(4);  
while (!myStack.empty())  
{  
    cout << myStack.top() << " ";  
    myStack.pop();  
}
```

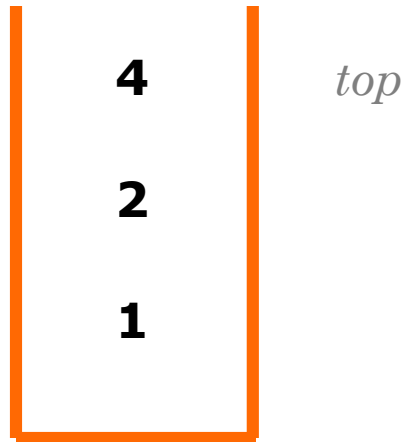
TRACING CODE



We **push** integer 4 into the **stack**.

```
stack<int> myStack;
myStack.push(1);
myStack.push(2);
myStack.push(3);
if (!myStack.empty())
    myStack.pop();
myStack.push(4);
while (!myStack.empty())
{
    cout << myStack.top() << " ";
    myStack.pop();
}
```

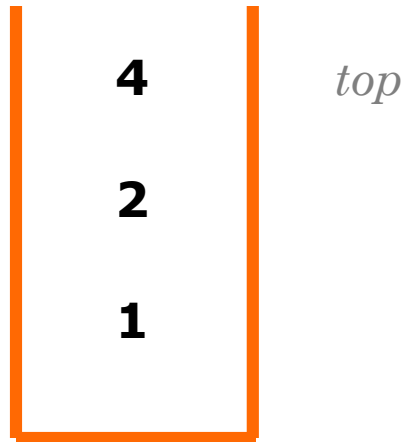
TRACING CODE



We **push** integer 4 into the **stack**.

```
stack<int> myStack;
myStack.push(1);
myStack.push(2);
myStack.push(3);
if (!myStack.empty())
    myStack.pop();
myStack.push(4);
while (!myStack.empty())
{
    cout << myStack.top() << " ";
    myStack.pop();
}
```

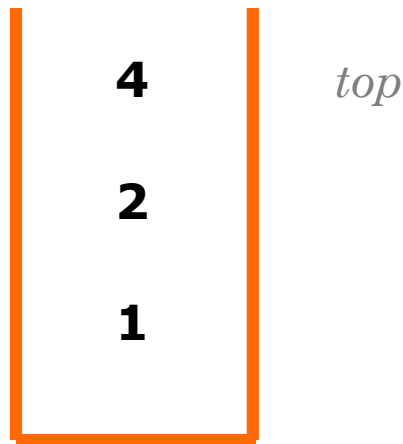
TRACING CODE



WHILE statement will execute as long as the **stack** is **not** empty.

```
stack<int> myStack;
myStack.push(1);
myStack.push(2);
myStack.push(3);
if (!myStack.empty())
    myStack.pop();
myStack.push(4);
while (!myStack.empty())
{
    cout << myStack.top() << " ";
    myStack.pop();
}
```

TRACING CODE



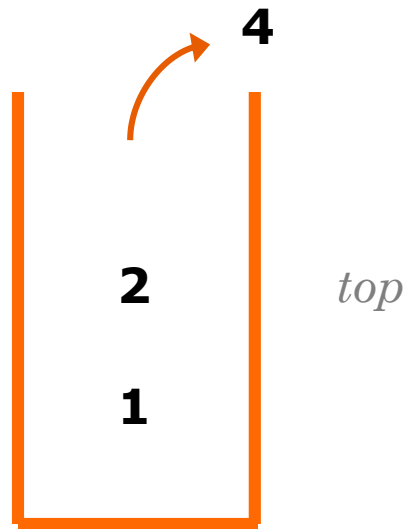
Retrieve (*without* removing) the **element** at the **top** of the **stack** and print it.

```
stack<int> myStack;
myStack.push(1);
myStack.push(2);
myStack.push(3);
if (!myStack.empty())
    myStack.pop();
myStack.push(4);
while (!myStack.empty())
{
    cout << myStack.top() << " ";
    myStack.pop();
}
```

Output:

4

TRACING CODE



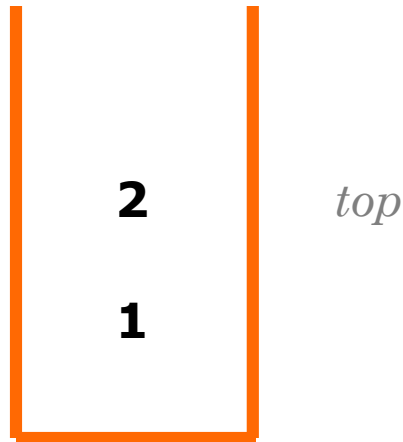
Pop the **element** at the **top** of the **stack**.

```
stack<int> myStack;
myStack.push(1);
myStack.push(2);
myStack.push(3);
if (!myStack.empty())
    myStack.pop();
myStack.push(4);
while (!myStack.empty())
{
    cout << myStack.top() << " ";
    myStack.pop();
}
```

Output:

4

TRACING CODE



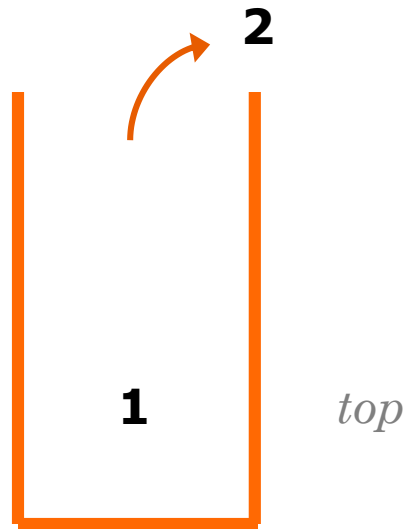
Retrieve (*without* removing) the **element** at the **top** of the **stack** and print it.

```
stack<int> myStack;
myStack.push(1);
myStack.push(2);
myStack.push(3);
if (!myStack.empty())
    myStack.pop();
myStack.push(4);
while (!myStack.empty())
{
    cout << myStack.top() << " ";
    myStack.pop();
}
```

Output:

4 2

TRACING CODE



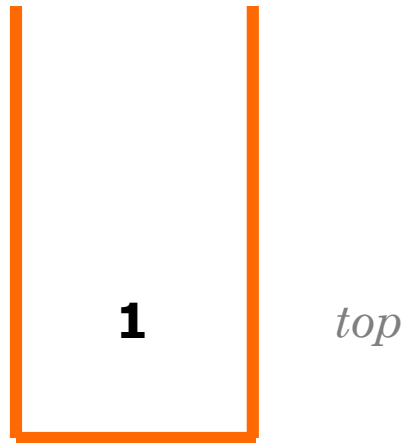
Pop the **element** at the **top** of the **stack**.

```
stack<int> myStack;
myStack.push(1);
myStack.push(2);
myStack.push(3);
if (!myStack.empty())
    myStack.pop();
myStack.push(4);
while (!myStack.empty())
{
    cout << myStack.top() << " ";
    myStack.pop();
}
```

Output:

4 2

TRACING CODE



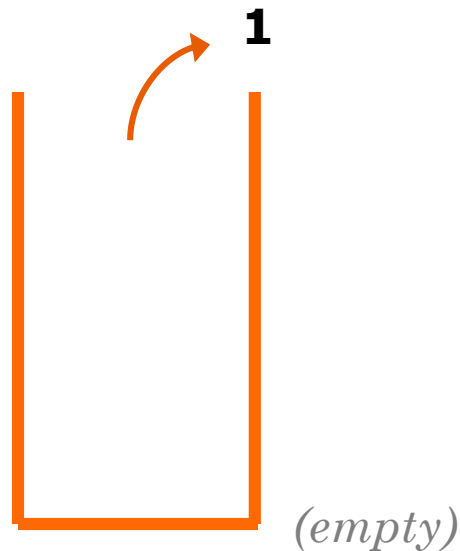
Retrieve (*without* removing) the **element** at the **top** of the **stack** and print it.

```
stack<int> myStack;
myStack.push(1);
myStack.push(2);
myStack.push(3);
if (!myStack.empty())
    myStack.pop();
myStack.push(4);
while (!myStack.empty())
{
    cout << myStack.top() << " ";
    myStack.pop();
}
```

Output:

4 2 1

TRACING CODE



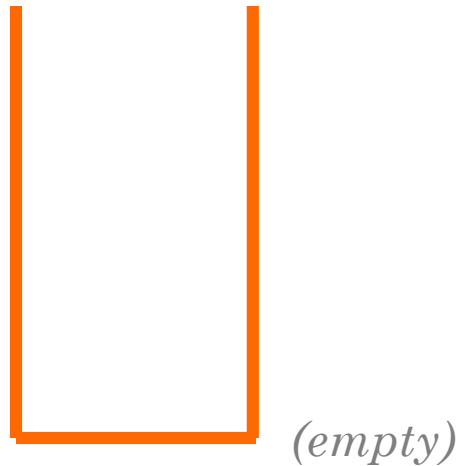
Pop the **element** at the **top** of the **stack**.

```
stack<int> myStack;  
myStack.push(1);  
myStack.push(2);  
myStack.push(3);  
if (!myStack.empty())  
    myStack.pop();  
myStack.push(4);  
while (!myStack.empty())  
{  
    cout << myStack.top() << " ";  
    myStack.pop();  
}
```

Output:

4

TRACING CODE



Stack is now **empty**;
WHILE statement
ends.

```
stack<int> myStack;  
myStack.push(1);  
myStack.push(2);  
myStack.push(3);  
if (!myStack.empty())  
    myStack.pop();  
myStack.push(4);  
while (!myStack.empty())  
{  
    cout << myStack.top() << " ";  
    myStack.pop();  
}
```

STACK ADT

- The **stack** is an **Abstract Data Type (ADT)**
 - What is an **ADT**?
 - It is an **abstract representation of data** that can be manipulated using a specific set of operations.
- Possible ways to implement a **stack**:
 - An **array**
 - Easier if inserting from left to right
 - **Top** is at **index[numOfElements – 1]**
 - A **linked list**
 - In a singly-linked list, the **top** is the **first** node

STACK ADT AS AN ARRAY

- Assume you are entering the following numbers, in this order, into the **stack**:

3 7 2 6 8

top



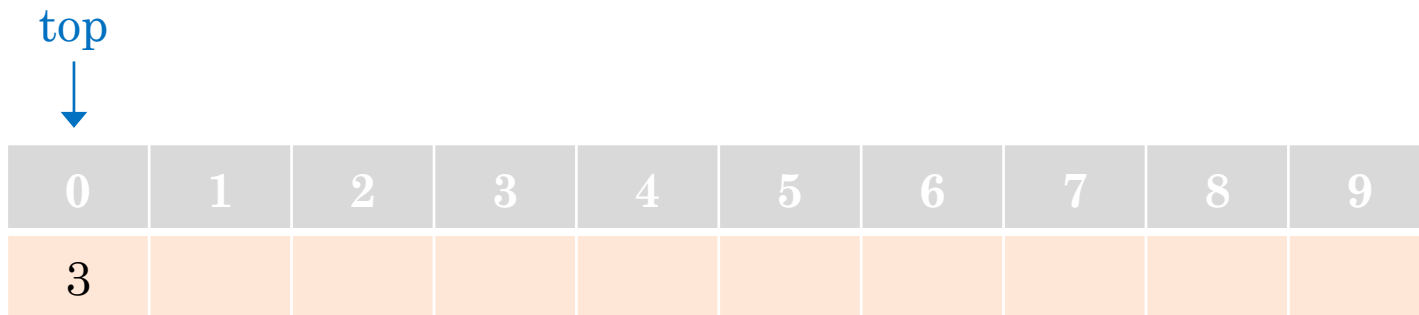
0	1	2	3	4	5	6	7	8	9

STACK ADT AS AN ARRAY

- Assume you are entering the following numbers, in this order, into the **stack**:

3 7 2 6 8

- Push 3 into the **stack**

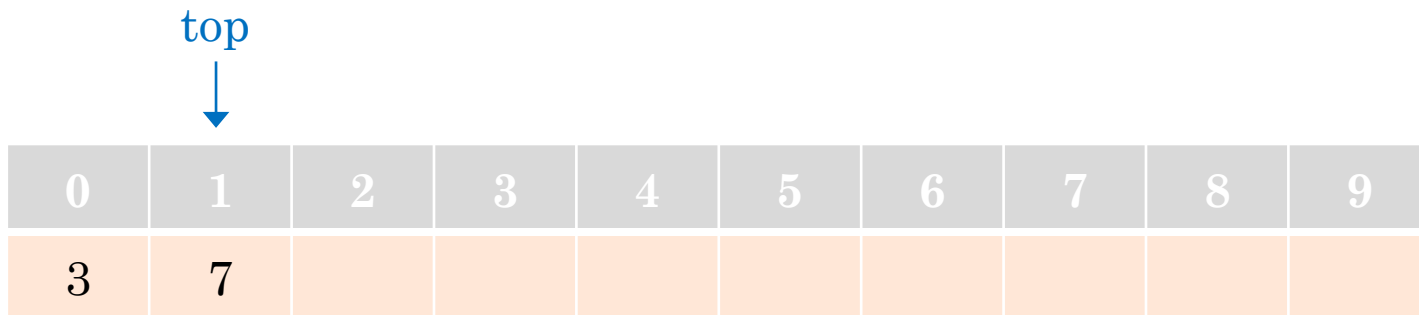


STACK ADT AS AN ARRAY

- Assume you are entering the following numbers, in this order, into the **stack**:

3 7 2 6 8

- Push 7 into the **stack**

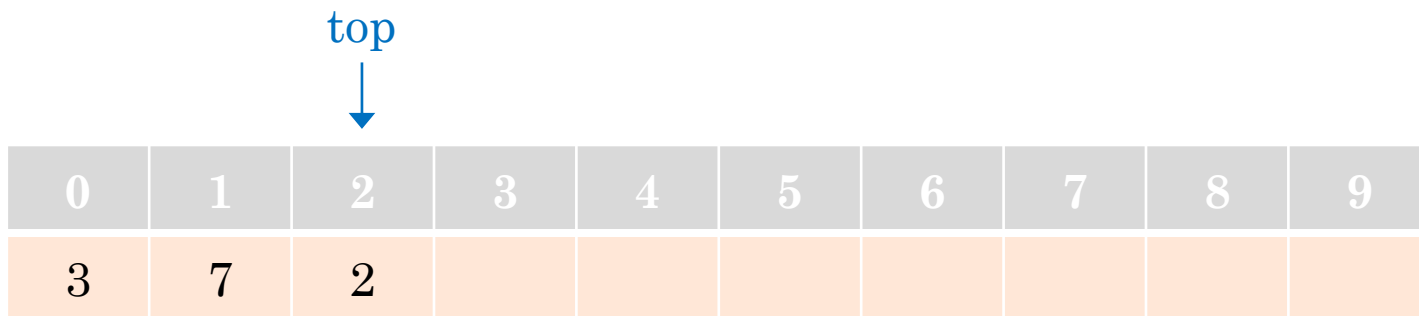


STACK ADT AS AN ARRAY

- Assume you are entering the following numbers, in this order, into the **stack**:

3 7 2 6 8

- Push 2 into the **stack**

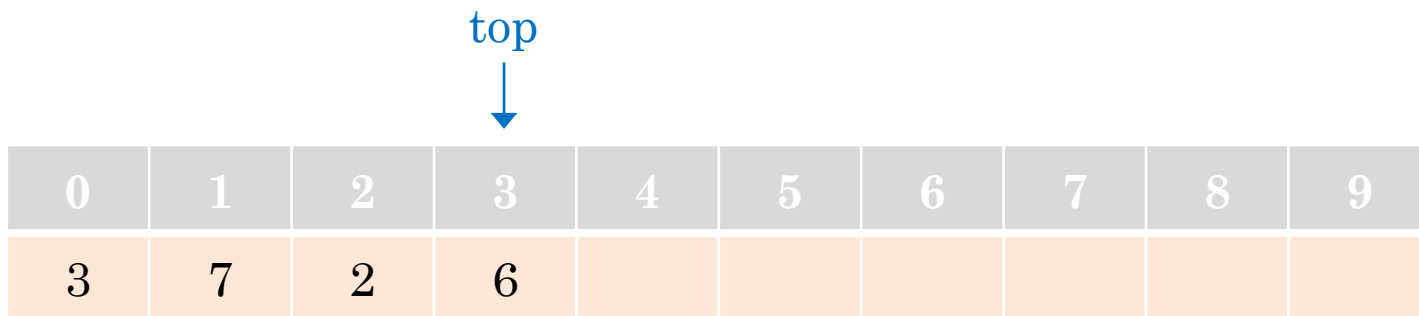


STACK ADT AS AN ARRAY

- Assume you are entering the following numbers, in this order, into the **stack**:

3 7 2 6 8

- Push 6 into the **stack**

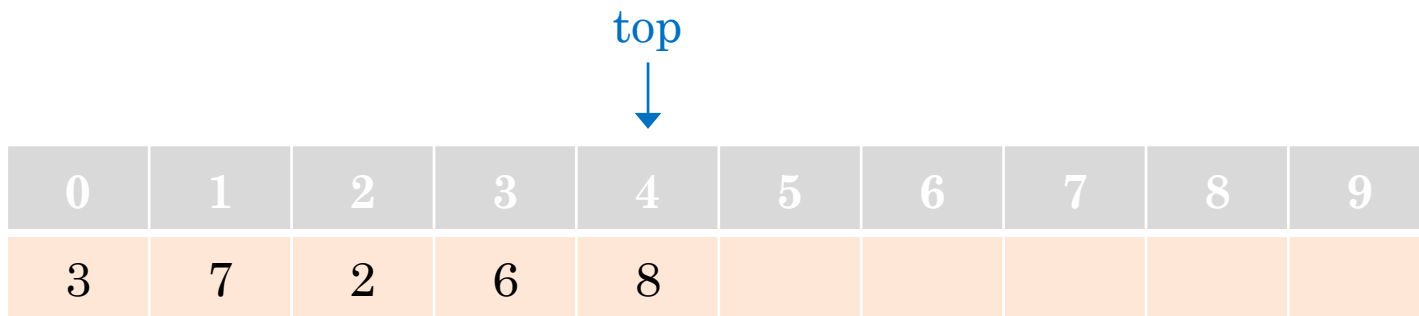


STACK ADT AS AN ARRAY

- Assume you are entering the following numbers, in this order, into the **stack**:

3 7 2 6 8

- Push 8 into the **stack**



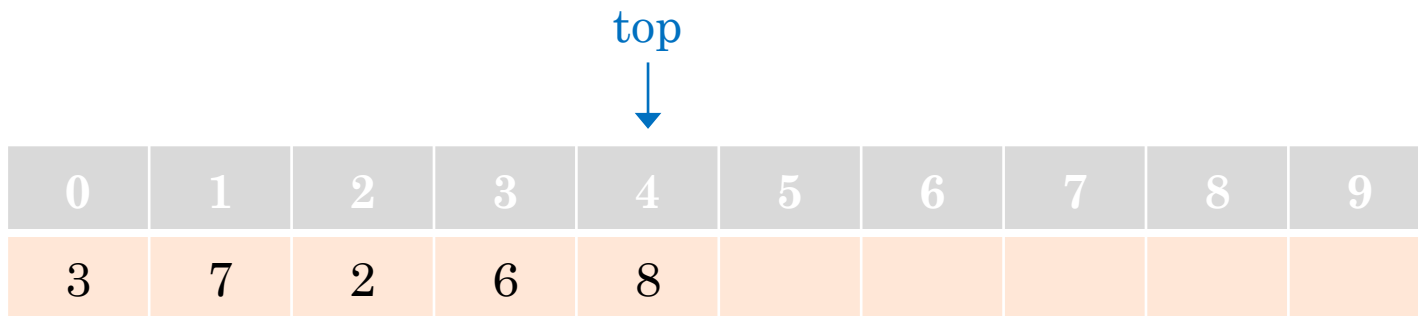
0	1	2	3	4	5	6	7	8	9
3	7	2	6	8					

STACK ADT AS AN ARRAY

- Assume you are entering the following numbers, in this order, into the **stack**:

3 7 2 6 8

- 3 will be at the **bottom** of the **stack**
- 8 will be at the **top** of the **stack**
 - Variable **top** will be at **index 4**



0	1	2	3	4	5	6	7	8	9
3	7	2	6	8					

STACK ADT AS A SINGLY-LINKED LIST

- Assume you are entering the following numbers, in this order, into the **stack**:

7 2 6 4

- You only need the pointer **top**
 - It is actually the pointer you have been naming **first** (or **head**)

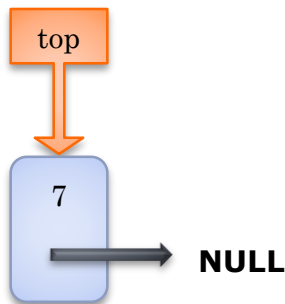


STACK ADT AS A SINGLY-LINKED LIST

- Assume you are entering the following numbers, in this order, into the **stack**:

7 2 6 4

- Insert 7 to the **front** of the list

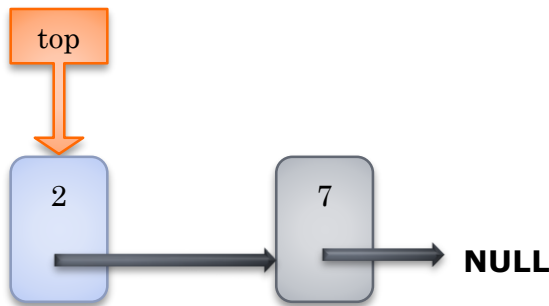


STACK ADT AS A SINGLY-LINKED LIST

- Assume you are entering the following numbers, in this order, into the **stack**:

7 2 6 4

- Insert 2 to the **front** of the list

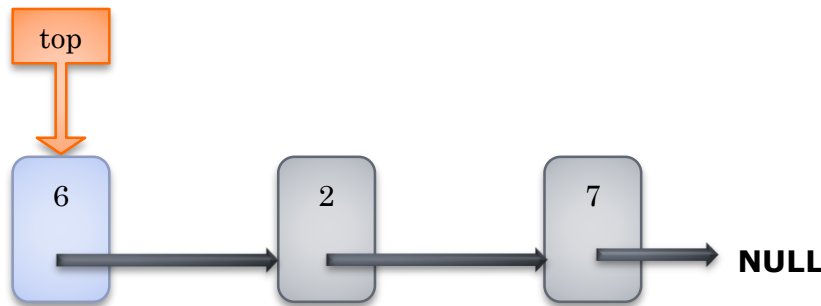


STACK ADT AS A SINGLY-LINKED LIST

- Assume you are entering the following numbers, in this order, into the **stack**:

7 2 6 4

- Insert 6 to the **front** of the list

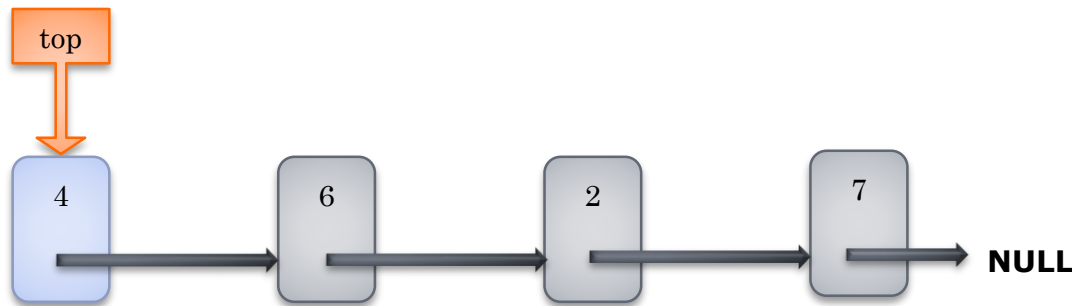


STACK ADT AS A SINGLY-LINKED LIST

- Assume you are entering the following numbers, in this order, into the **stack**:

7 2 6 4

- Insert 4 to the **front** of the list



COMMON OPERATION IDENTIFIERS

- Other **identifiers** used for common operations on the **stack**:
 - **empty()** = **isEmpty()**
 - **top()** = **peek()** = **retrieve()**
- **Note** that in some implementations the function **pop()** returns a value **and** removes the element as well.

STACK APPLICATIONS

- **Stacks** are used in many **applications**:
 - Track C++ **function calls**
 - Compilers perform **syntax analysis** (loops)
 - **Back button** in a browser
 - **Undo button** in a word processor (or other applications)
 - And more...

EXAMPLE

- Project: stacks



STACKS (END)

46