CS302 - Data Structures

using C++

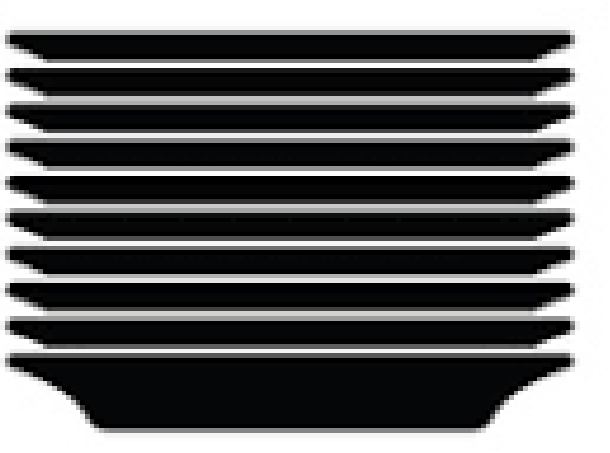
Topic: The ADT Stack

Kostas Alexis

AUTONOMOUS ROBOTS LAB

The ADT Stack

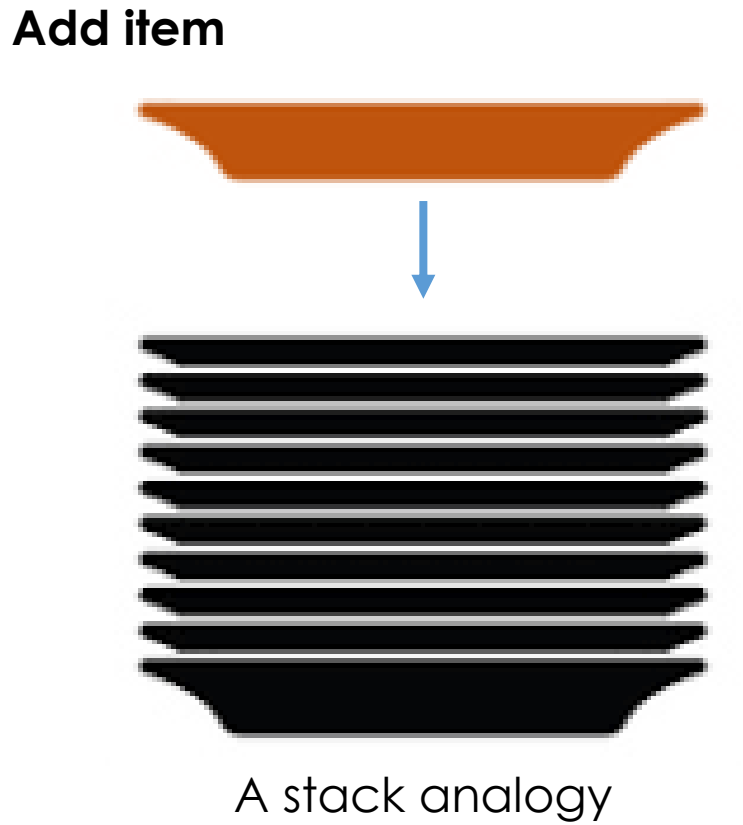
- Stack Concept
  - **Last-in, first-out (LIFO) property**
  - Last item placed on stack will be first item removed
  - Items placed and removed on top of stack
- Analogies
  - Books on a desk
  - Dishes in a Cafeteria
  - Boxes in an attic



A stack analogy

The ADT Stack

- Stack Concept
  - **Last-in, first-out (LIFO) property**
  - Last item placed on stack will be first item removed
  - Items placed and removed on top of stack
- Analogies
  - Books on a desk
  - Dishes in a Cafeteria
  - Boxes in an attic

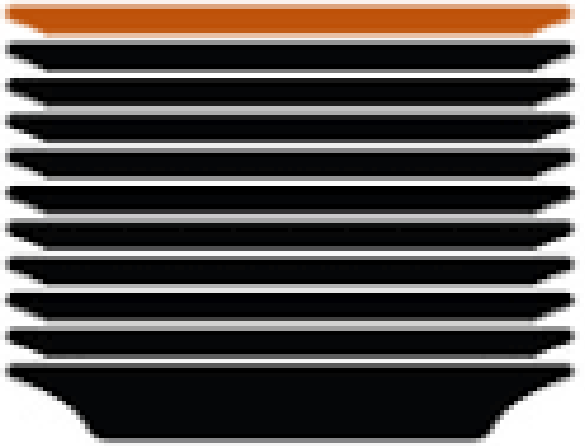


A stack analogy

The ADT Stack

- Stack Concept
  - **Last-in, first-out (LIFO) property**
  - Last item placed on stack will be first item removed
  - Items placed and removed on top of stack
- Analogies
  - Books on a desk
  - Dishes in a Cafeteria
  - Boxes in an attic

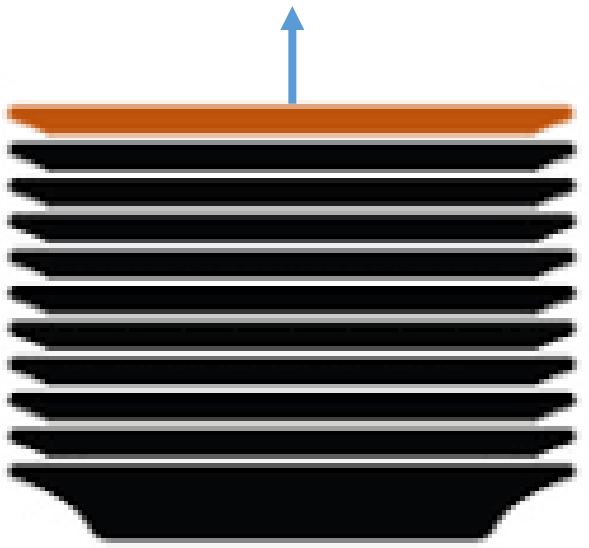
Add item



A stack analogy

The ADT Stack

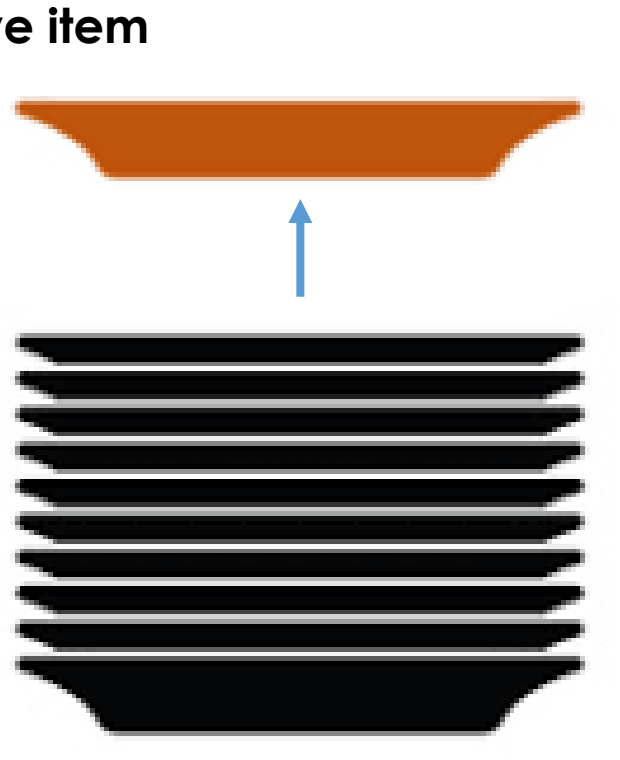
- Stack Concept
  - **Last-in, first-out (LIFO) property**
  - Last item placed on stack will be first item removed
  - Items placed and removed on top of stack
- Analogies
  - Books on a desk
  - Dishes in a Cafeteria
  - Boxes in an attic



A stack analogy

The ADT Stack

- Stack Concept
  - **Last-in, first-out (LIFO) property**
  - Last item placed on stack will be first item removed
  - Items placed and removed on top of stack
- Analogies
  - Books on a desk
  - Dishes in a Cafeteria
  - Boxes in an attic



A stack analogy

The ADT Stack

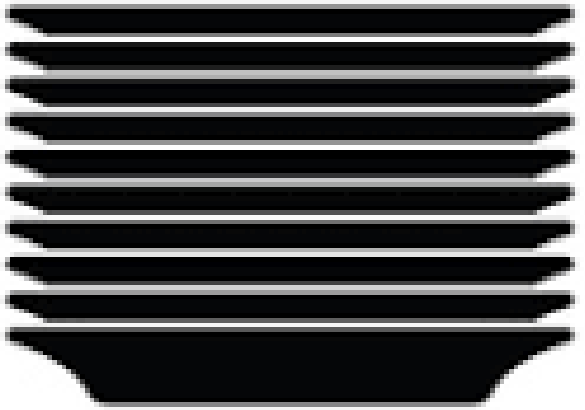
- Stacks are heavily used in CS
  - Examples?

The ADT Stack

- Stacks are heavily used in CS
  - Reverse a word
  - Undo mechanism in text editors
  - Backtracking – when you need to access the most recent data element in a serious of elements
  - Function Call
    - A stack is used to keep information about the active functions or subroutines
  - Language processing:
    - Space for parameters and local variables is created internally using a stack
    - Compilet's syntax check for matching braces is implemented by using a stack
    - Support for recusion

The ADT Stack

Restricts access “from top”

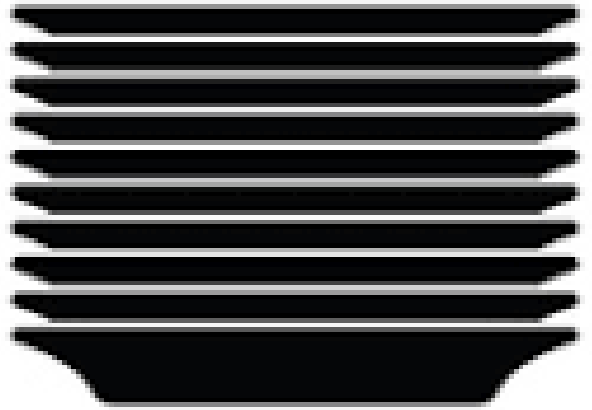


A stack analogy

The ADT Stack

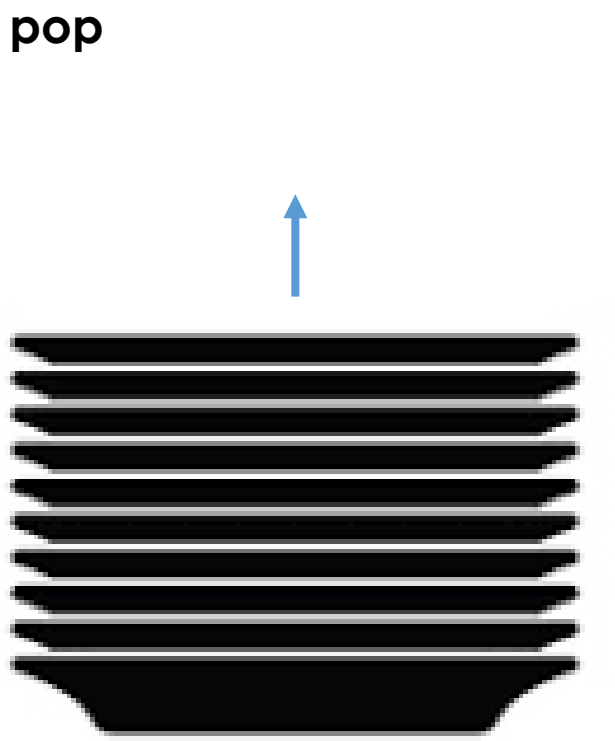
Restricts access "from top"

If one wants to access the 6th- from top entry?



A stack analogy

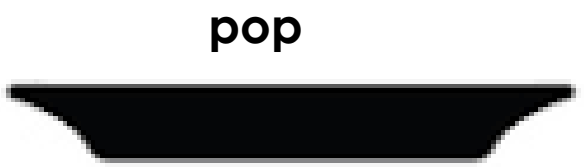
The ADT Stack

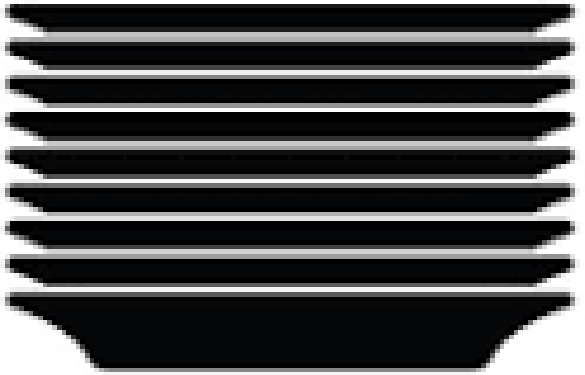


If one wants to access the 6th- from top entry?

A stack analogy

The ADT Stack



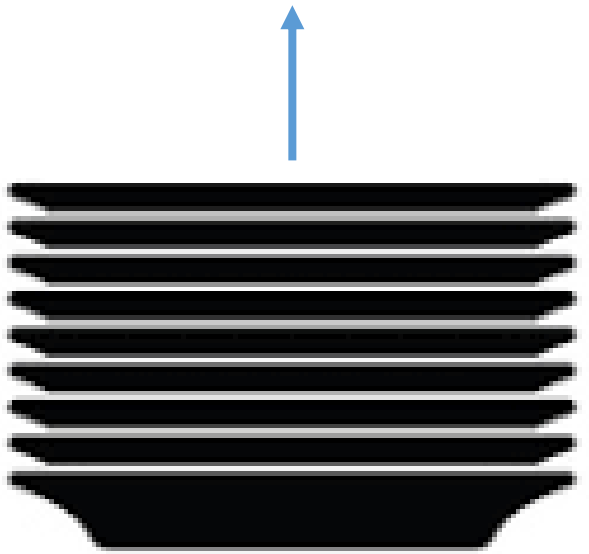


If one wants to access the 6th- from top entry?

A stack analogy

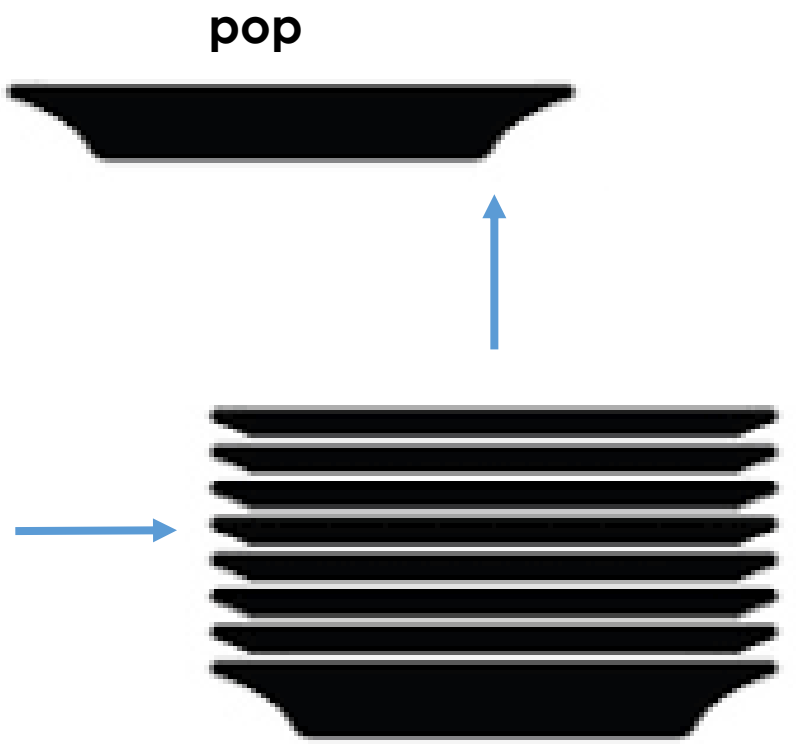
The ADT Stack

If one wants to access the 6th- from top entry?



A stack analogy

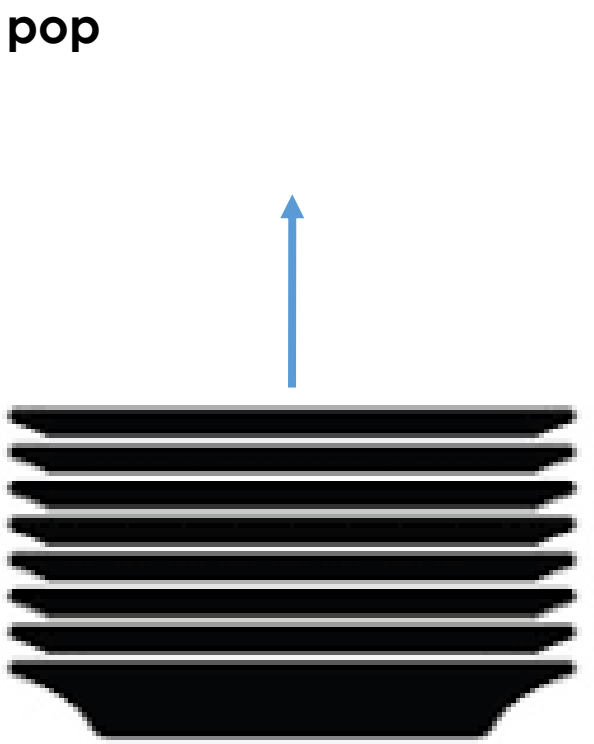
The ADT Stack



If one wants to access the 6th- from top entry?

A stack analogy

The ADT Stack

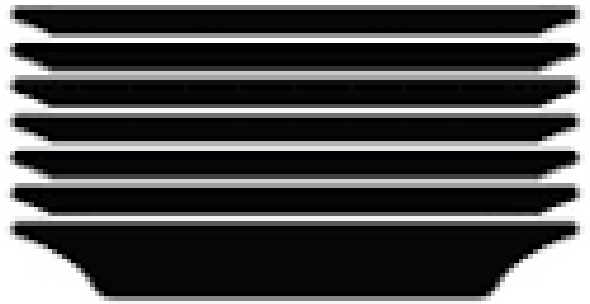


If one wants to access the 6th- from top entry?

A stack analogy

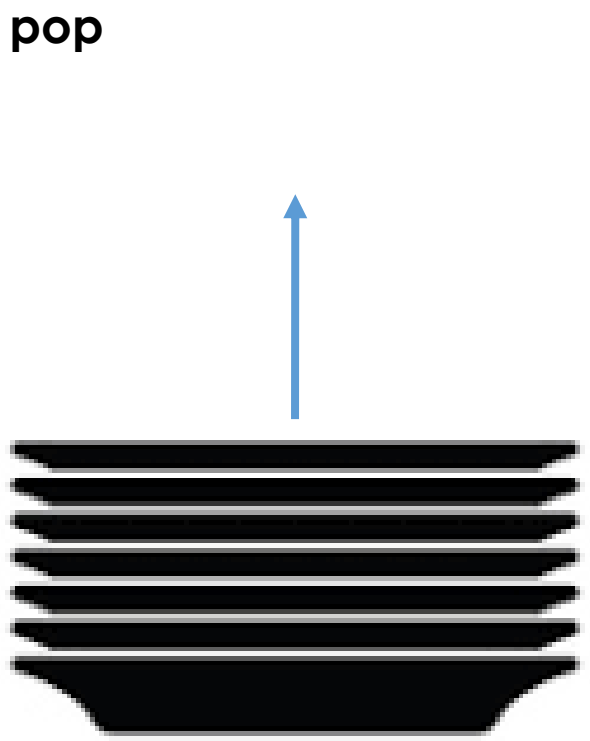
The ADT Stack

If one wants to access the 6th- from top entry?



A stack analogy

The ADT Stack

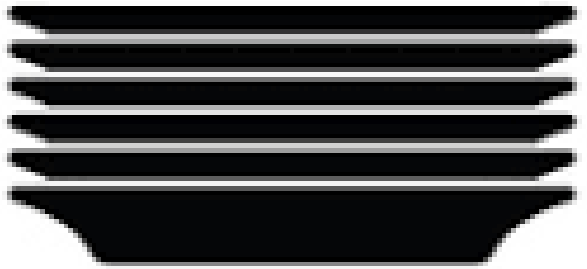


If one wants to access the 6th- from top entry?

A stack analogy

The ADT Stack

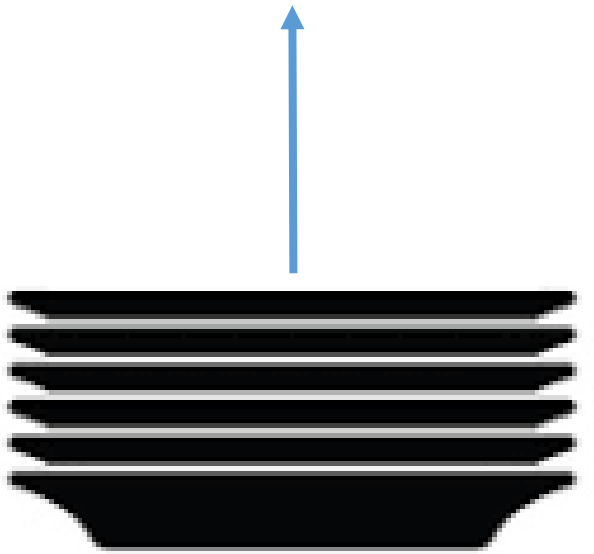
If one wants to access the 6th- from top entry?



A stack analogy

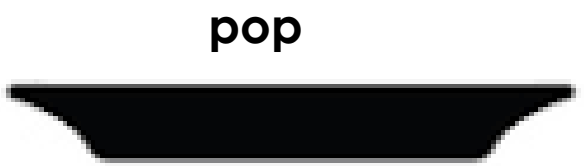
The ADT Stack

If one wants to access the 6th- from top entry?

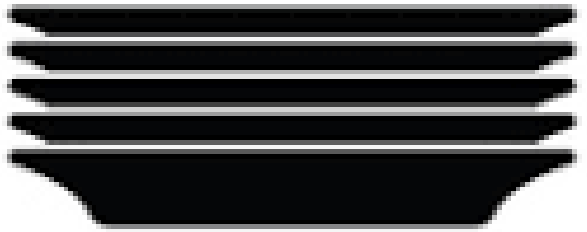


A stack analogy

The ADT Stack

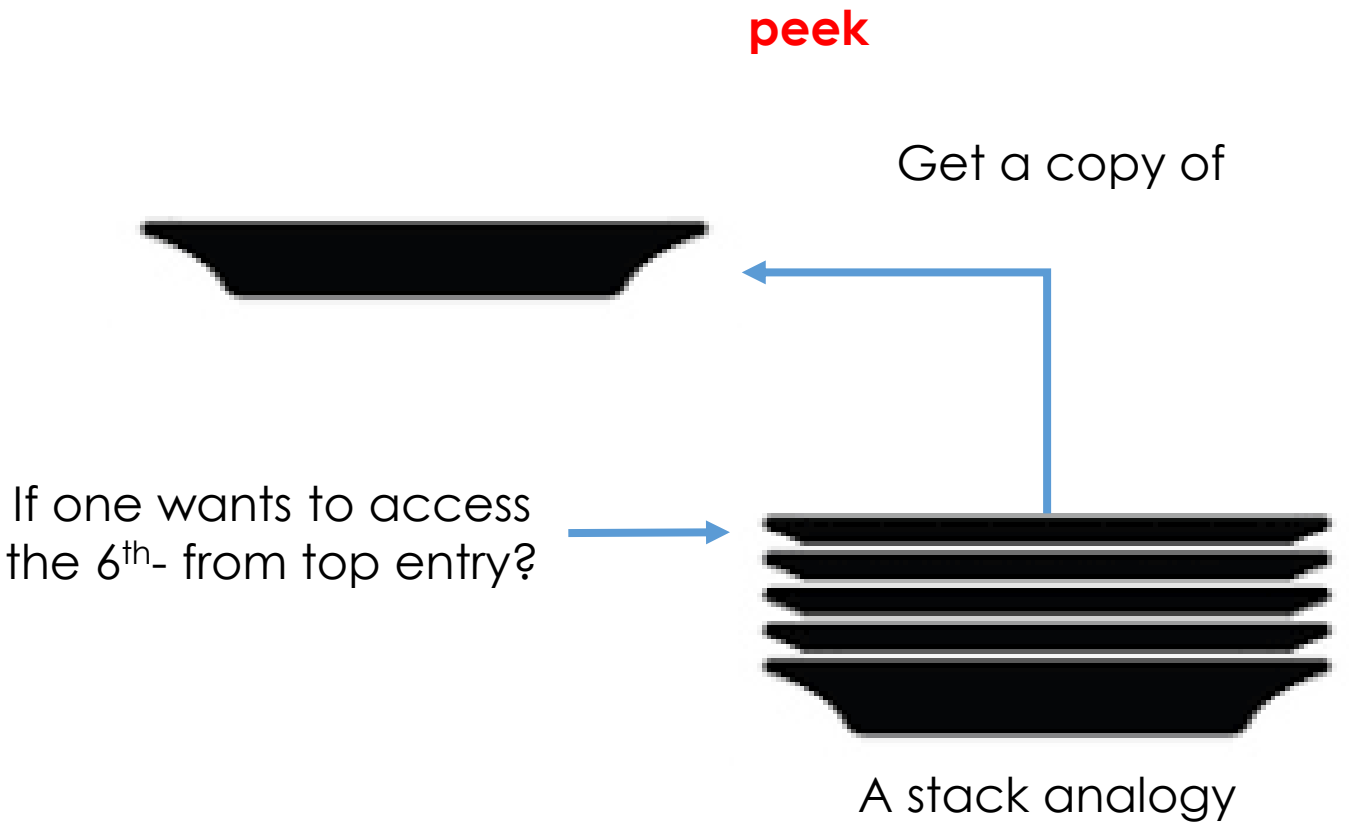


If one wants to access the 6th- from top entry?



A stack analogy

The ADT Stack



The ADT Stack

- Collection of items in reverse chronological order with the same data type
- ADT Stack operations
  - Add a new item to the stack: **`push(ItemType newEntry)`**
  - Remove item that was added most recently: **`pop()`**
  - Retrieve item that was added most recently: **`ItemType peek()`**
  - Determine whether a stack is empty: **`boolean isEmpty()`**

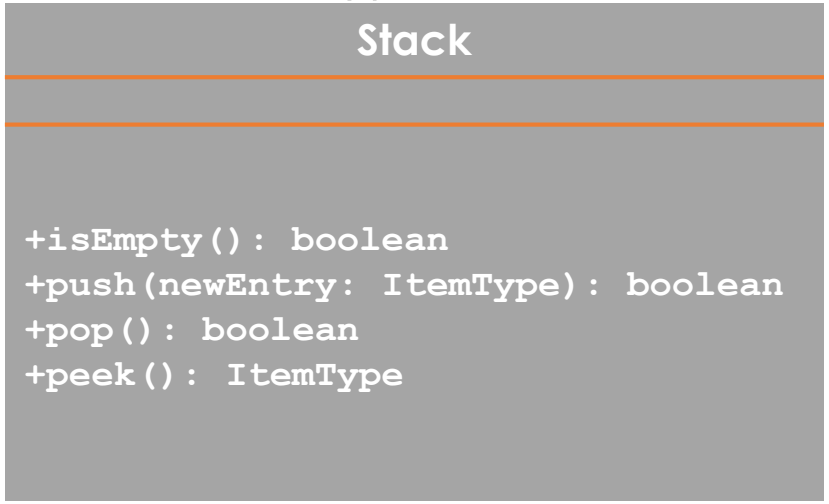
The ADT Stack

- Collection of items in reverse chronological order with the same data type
- ADT Stack operations
  - Add a new item to the stack: **`push(ItemType newEntry)`**
  - Remove item that was added most recently: **`pop()`**
  - Retrieve item that was added most recently: **`ItemType peek()`**
  - Determine whether a stack is empty: **`boolean isEmpty()`**

Only returns a copy – the item is not removed from the stack

The ADT Stack

- Collection of items in reverse chronological order with the same data type
- ADT Stack operations
  - Add a new item to the stack: `push(ItemType newEntry)`
  - Remove item that was added most recently: `pop()`
  - Retrieve item that was added most recently: `ItemType peek()`
  - Determine whether a stack is empty: `boolean isEmpty()`



The ADT Stack

- Collection of items in reverse chronological order with the same data type
- ADT Stack operations
  - Add a new item to the stack: **`push(ItemType newEntry)`**
  - Remove item that was added most recently: **`pop()`**
  - Retrieve item that was added most recently: **`ItemType peek()`**
  - Determine whether a stack is empty: **`boolean isEmpty()`**

```
/** @file StackInterface.h */

#ifndef STACK_INTERFACE_
#define STACK_INTERFACE_

template<class ItemType>
class StackInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual bool push(const ItemType& newEntry) = 0;
      virtual bool pop() = 0;
      virtual ItemType peek() const = 0;
      virtual ~StackInterface() { }
};    // end StackInterface
#endif
```

Using the ADT Stack

```
template<class ItemType>
class StackInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual bool push(const ItemType& newEntry) = 0;
      virtual bool pop() = 0;
      virtual ItemType peek() const = 0;
      virtual ~StackInterface() { }
};    // end StackInterface
```

```
stack<std::string>* stringStack = new Stack<std::string>();
stringStack->push("Jim");
stringStack->push("Jess");
stringStack->push("Jill");
stringStack->push("Jane");
stringStack->push("Joe");

std::string top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";

top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";
```

Using the ADT Stack

```
template<class ItemType>
class StackInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual bool push(const ItemType& newEntry) = 0;
      virtual bool pop() = 0;
      virtual ItemType peek() const = 0;
      virtual ~StackInterface() { }
};    // end StackInterface
```

```
stack<std::string>* stringStack = new Stack<std::string>();
stringStack->push("Jim");
stringStack->push("Jess");
stringStack->push("Jill");
stringStack->push("Jane");
stringStack->push("Joe");

std::string top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";

top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";
```

Using the ADT Stack

```
template<class ItemType>
class StackInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual bool push(const ItemType& newEntry) = 0;
      virtual bool pop() = 0;
      virtual ItemType peek() const = 0;
      virtual ~StackInterface() { }
};    // end StackInterface
```

Jim

```
stack<std::string>* stringStack = new Stack<std::string>();
stringStack->push("Jim");
stringStack->push("Jess");
stringStack->push("Jill");
stringStack->push("Jane");
stringStack->push("Joe");

std::string top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";

top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";
```

Using the ADT Stack

```
template<class ItemType>
class StackInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual bool push(const ItemType& newEntry) = 0;
      virtual bool pop() = 0;
      virtual ItemType peek() const = 0;
      virtual ~StackInterface() { }
};    // end StackInterface
```

Jess
Jim

```
stack<std::string>* stringStack = new Stack<std::string>();
stringStack->push("Jim");
stringStack->push("Jess");
stringStack->push("Jill");
stringStack->push("Jane");
stringStack->push("Joe");

std::string top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";

top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";
```

Using the ADT Stack

```
template<class ItemType>
class StackInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual bool push(const ItemType& newEntry) = 0;
      virtual bool pop() = 0;
      virtual ItemType peek() const = 0;
      virtual ~StackInterface() { }
};    // end StackInterface
```

```
stack<std::string>* stringStack = new Stack<std::string>();
stringStack->push("Jim");
stringStack->push("Jess");
stringStack->push("Jill");
stringStack->push("Jane");
stringStack->push("Joe");

std::string top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";

top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";
```

Jill
Jess
Jim

Using the ADT Stack

```
template<class ItemType>
class StackInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual bool push(const ItemType& newEntry) = 0;
      virtual bool pop() = 0;
      virtual ItemType peek() const = 0;
      virtual ~StackInterface() { }
};    // end StackInterface
```

```
stack<std::string>* stringStack = new Stack<std::string>();
stringStack->push("Jim");
stringStack->push("Jess");
stringStack->push("Jill");
stringStack->push("Jane");
stringStack->push("Joe");

std::string top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";

top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";
```

Jane
Jill
Jess
Jim

Using the ADT Stack

```
template<class ItemType>
class StackInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual bool push(const ItemType& newEntry) = 0;
      virtual bool pop() = 0;
      virtual ItemType peek() const = 0;
      virtual ~StackInterface() { }
};    // end StackInterface
```

```
stack<std::string>* stringStack = new Stack<std::string>();
stringStack->push("Jim");
stringStack->push("Jess");
stringStack->push("Jill");
stringStack->push("Jane");
stringStack->push("Joe");

std::string top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";

top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";
```

Joe
Jane
Jill
Jess
Jim

```
Joe is at the top of the stack
```

Using the ADT Stack

```
template<class ItemType>
class StackInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual bool push(const ItemType& newEntry) = 0;
      virtual bool pop() = 0;
      virtual ItemType peek() const = 0;
      virtual ~StackInterface() { }
};    // end StackInterface
```

```
stack<std::string>* stringStack = new Stack<std::string>();
stringStack->push("Jim");
stringStack->push("Jess");
stringStack->push("Jill");
stringStack->push("Jane");
stringStack->push("Joe");

std::string top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";

top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";
```

Jane
Jill
Jess
Jim

```
Joe is at the top of the stack
Joe is removed from the stack
```

Using the ADT Stack

```
template<class ItemType>
class StackInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual bool push(const ItemType& newEntry) = 0;
      virtual bool pop() = 0;
      virtual ItemType peek() const = 0;
      virtual ~StackInterface() { }
};    // end StackInterface
```

```
stack<std::string>* stringStack = new Stack<std::string>();
stringStack->push("Jim");
stringStack->push("Jess");
stringStack->push("Jill");
stringStack->push("Jane");
stringStack->push("Joe");

std::string top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";

top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";
```

Jane
Jill
Jess
Jim

```
Joe is at the top of the stack
Joe is removed from the stack
Jane is at the top of the stack
```

Using the ADT Stack

```
template<class ItemType>
class StackInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual bool push(const ItemType& newEntry) = 0;
      virtual bool pop() = 0;
      virtual ItemType peek() const = 0;
      virtual ~StackInterface() { }
};    // end StackInterface
```

```
stack<std::string>* stringStack = new Stack<std::string>();
stringStack->push("Jim");
stringStack->push("Jess");
stringStack->push("Jill");
stringStack->push("Jane");
stringStack->push("Joe");

std::string top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";

top = stringStack->peek();
std::cout << top << " is at the top of the stack \n";

if(stringStack->pop())
      std::cout << top << " is removed from the stack \n";
```

Jill
Jess
Jim

```
Joe is at the top of the stack
Joe is removed from the stack
Jane is at the top of the stack
Jane is removed from the stack
```

Using the ADT Stack

- Checking for balanced expressions

Using the ADT Stack

- Checking for balanced expressions

Using the ADT Stack

- Checking for balanced expressions
  - Scan expression:
    - Discard characters that are not delimiters

{ [( ) ( )] ( ) }

[( ])

a { b [c (d + e) / 2 – f] + 1 }

Using the ADT Stack

- Checking for balanced expressions
  - Scan expression:
    - Discard characters that are not delimiters
  - When open delimiter is encountered
    - **`push`** it on the stack

{ [( ) ( )] ( ) }

[( ])

a { b [c (d + e) / 2 – f] + 1 }

Using the ADT Stack

- Checking for balanced expressions
  - Scan expression:
    - Discard characters that are not delimiters
  - When open delimiter is encountered
    - **`push`** it on the stack
  - When close delimiter is encountered
    - Check to see if it matches top of stack
    - If yes, **`pop`** off top of stack
    - If not, expression is not balanced

{ [( ) ( )] ( ) }

[( ])

a { b [c (d + e) / 2 – f] + 1 }

Using the ADT Stack

- Checking for balanced expressions
  - Scan expression:
    - Discard characters that are not delimiters
  - When open delimiter is encountered
    - **push** it on the stack
  - When close delimiter is encountered
    - Check to see if it matches top of stack
    - If yes, **pop** off top of stack
    - If not, expression is not balanced
  - If braces are balanced
    - Stack is empty when expression is done

{ [( ) ( )] ( ) }

[( ])

a { b [c (d + e) / 2 – f] + 1 }

Using the ADT Stack

- Checking for balanced expressions
  - Scan expression:
    - Discard characters that are not delimiters
  - When open delimiter is encountered
    - **`push`** it on the stack
  - When close delimiter is encountered
    - Check to see if it matches top of stack
    - If yes, **`pop`** off top of stack
    - If not, expression is not balanced
  - If braces are balanced
    - Stack is empty when expression is done

{ [( ) ( )] ( ) }

[( ])

a { b [c (d + e) / 2 – f] + 1 }

Using the ADT Stack

- Checking for balanced expressions
  - Scan expression:
    - Discard characters that are not delimiters
  - When open delimiter is encountered
    - **push** it on the stack
  - When close delimiter is encountered
    - Check to see if it matches top of stack
    - If yes, **pop** off top of stack
    - If not, expression is not balanced
  - If braces are balanced
    - Stack is empty when expression is done

{ [( ) ( )] ( ) }

[( ])

a { b [c (d + e] / 2 – f) + 1 }

Using the ADT Stack

- Checking for balanced expressions
  - Scan expression:
    - Discard characters that are not delimiters
  - When open delimiter is encountered
    - **`push`** it on the stack
  - When close delimiter is encountered
    - Check to see if it matches top of stack
    - If yes, **`pop`** off top of stack
    - If not, expression is not balanced
  - If braces are balanced
    - Stack is empty when expression is done

{ [( ) ( )] ( ) }

[( ])

a { b [c (d + e **]** / 2 – f **)** + 1 }

Using the ADT Stack (of characters)

- Checking for balanced expressions

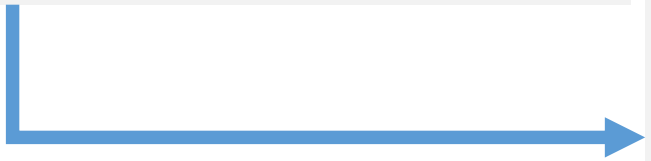
```
// Returns true if the given characters, open and close, form a
// pair of parentheses, brackets, or braces

bool isPaired(char open, char close)
{
     return (open == '(' && close == ')') ||
            (open == '[' && close == ']') ||
            (open == '{' && close == '}');
};         // end isPaired
```

Helper method

Using the ADT Stack (of characters)

```
bool checkBalance(string expression)
{
      Stack<char> openDelimiterStack = new Stack<char>();
      int characterCount = expression.length();
      bool isBalanced = true;
      int index = 0;
      char nextCharacter = ' ';
```



```
// Returns true if the given characters, open and close, form a
// pair of parentheses, brackets, or braces

bool isPaired(char open, char close)
{
      return (open == '(' && close == ')') ||
             (open == '[' && close == ']') ||
             (open == '{' && close == '}');
};          // end isPaired
```

```
while (isBalanced && (index < characterCount)) {
{
      nextCharacter = expression.charAt(index);
      switch (nextCharacter)
      {
            case '(': case '[': case '{':
                  openDelimiterStack->push(nextCharacter)
                  break;
            case ')': case ']': case '}':
                  if (openDilimiterStack->isEmpty())
                        isBalanced = false;
                  else
                  {
                        char openDilimeter = openDilimiterStack->peek();
                        openDelimiterStack->pop();
                        isBalanced =
                  isPaired(openDilimiter,nextCharacter);
                  } // end if
                  break;
            default:
                  break;
      } // end switch
      index++;
} // end while
if (!openDilimeterStack->isEmpty())
      isBalanced = false;
return isBalanced;
} // end checkBalance
```

Thank you