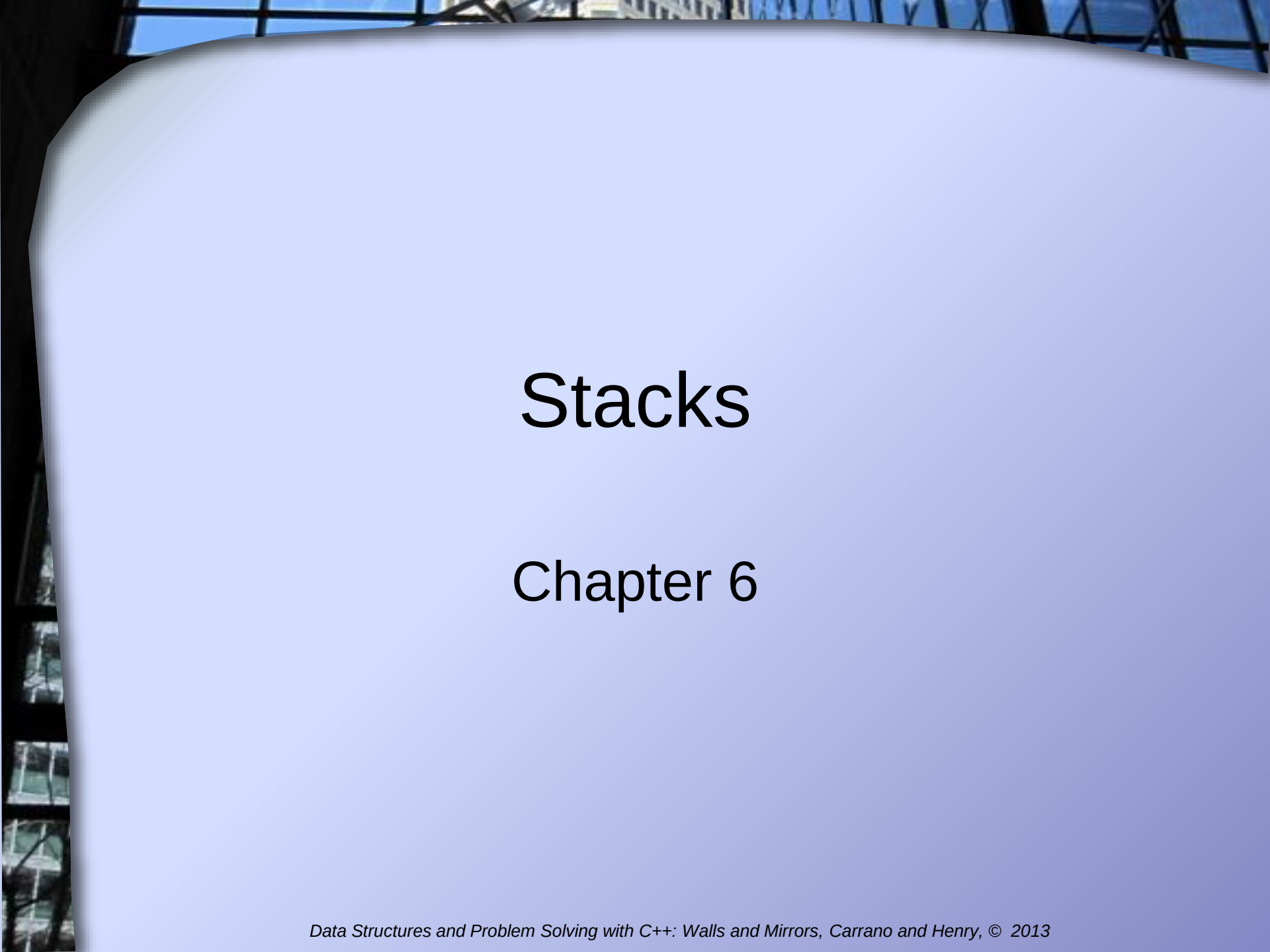


# Stacks

## Chapter 6

# Contents

- The Abstract Data Type Stack
- Simple Uses of a Stack
- Using Stacks with Algebraic Expressions
- Using a Stack to Search a Flight Map
- The Relationship Between Stacks and Recursion

# The Abstract Data Type Stack

- Developing an ADT during the design of a solution
- Consider entering keyboard text
  - Mistakes require use of backspace
    `abcdd←←efgg←`
- We seek a programming solution to read these keystrokes

# The Abstract Data Type Stack

- Pseudocode of first attempt

```
// Read the line, correcting mistakes along the way
while (not end of line)
{
    Read a new character ch
    if (ch is not a '←')
        Add ch to the ADT
    else
        Remove from the ADT (discard) the item that was added most recently
}
```

- Requires
  - Add new item to ADT
  - Remove most recently added item

# Specifications for the ADT `Stack`

- We have identified the following operations:
  - See whether stack is empty.
  - Add a new item to stack.
  - Remove from the stack item added most recently.
  - Get item that was added to stack most recently.
- Stack uses LIFO principle
  - Last In First Out

# Specifications for the ADT `Stack`

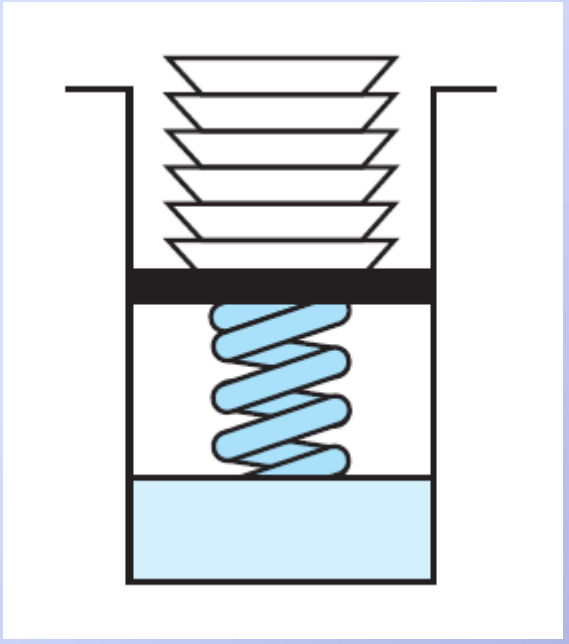


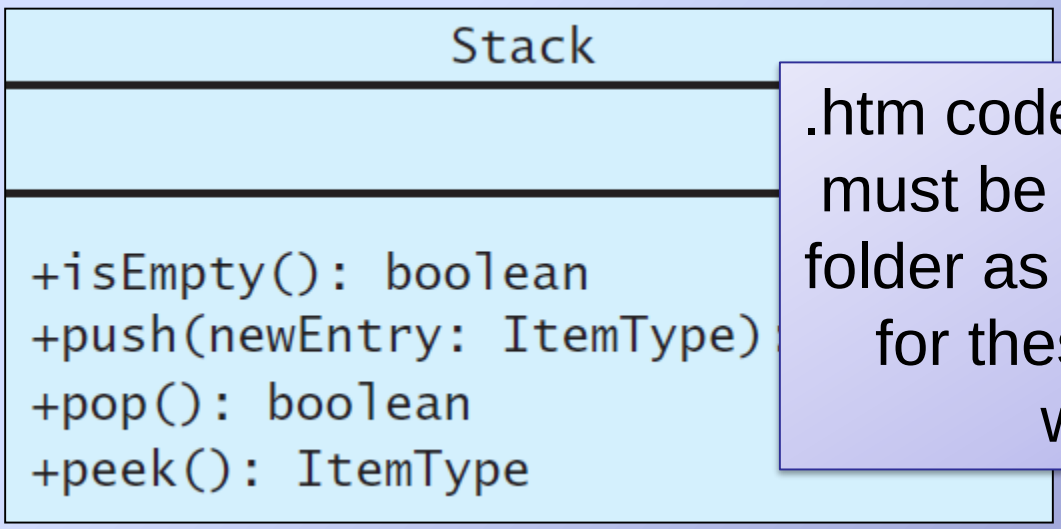
FIGURE 6-1 A stack of cafeteria plates

# Abstract Data Type: `Stack`

- A finite number of objects
  - Not necessarily distinct
  - Having the same data type
  - Ordered by when they were added
- Operations
  - **`isEmpty()`**
  - **`push(newEntry)`**
  - **`pop()`**
  - **`peek()`**

# Abstract Data Type: `Stack`

- View C++ Stack interface, Listing 6-1




.htm code listing files must be in the same folder as the .ppt files for these links to work

FIGURE 6-2 UML diagram for the class `Stack`

# Axioms for the ADT **`Stack`**

- **`new Stack()).isEmpty() = true`**
- **`new Stack()).pop() = false`**
- **`new Stack()).peek() = error`**
- **`aStack.push(item)).isEmpty() = false`**
- **`aStack.push(item)).peek() = item`**
- **`aStack.push(item)).pop() = true`**

# Simple Uses of a Stack

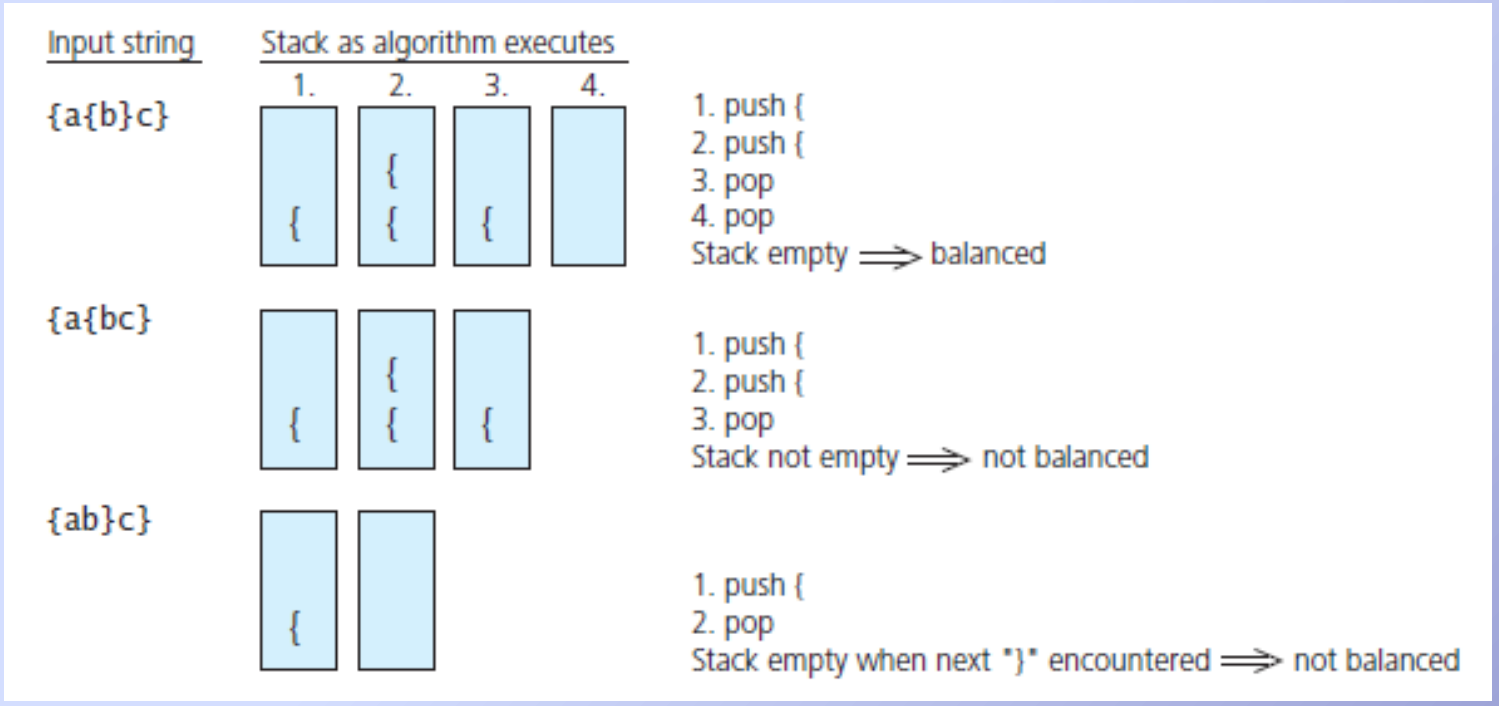



FIGURE 6-3 Traces of the algorithm that checks for balanced braces

# Simple Uses of a Stack

- Recognizing strings in a language
- Consider
  $L$ = { $s\$s'$ : $s$ is a possibly empty string of characters other than $\$$ , $s'$ = reverse( $s$ )}
- View algorithm to verify a string for a given language, Listing 6-A

# Using Stacks with Algebraic Expressions

- Evaluating postfix expressions

| Key entered | Calculator action | | Stack (bottom to top): |
|---|---|---|---|
| 2 | push 2 | | 2 |
| 3 | push 3 | | 2 3 |
| 4 | push 4 | | 2 3 4 |
| + | operand2 = peek | (4) | 2 3 4 |
| | pop | | 2 3 |
| | operand1 = peek | (3) | 2 3 |
| | pop | | 2 |
| | result = operand1 + operand2 | (7) | |
| | push result | | 2 7 |
| * | operand2 = peek | (7) | 2 7 |
| | pop | | 2 |
| | operand1 = peek | (2) | 2 |
| | pop | | |
| | result = operand1 * operand2 | (14) | |
| | push result | | 14 |

FIGURE 6-4 The effect of a postfix calculator on a stack when evaluating the expression 2 * (3 + 4)

# Using Stacks with Algebraic Expressions

- Converting infix expressions to equivalent postfix expressions
- Possible pseudocode solution

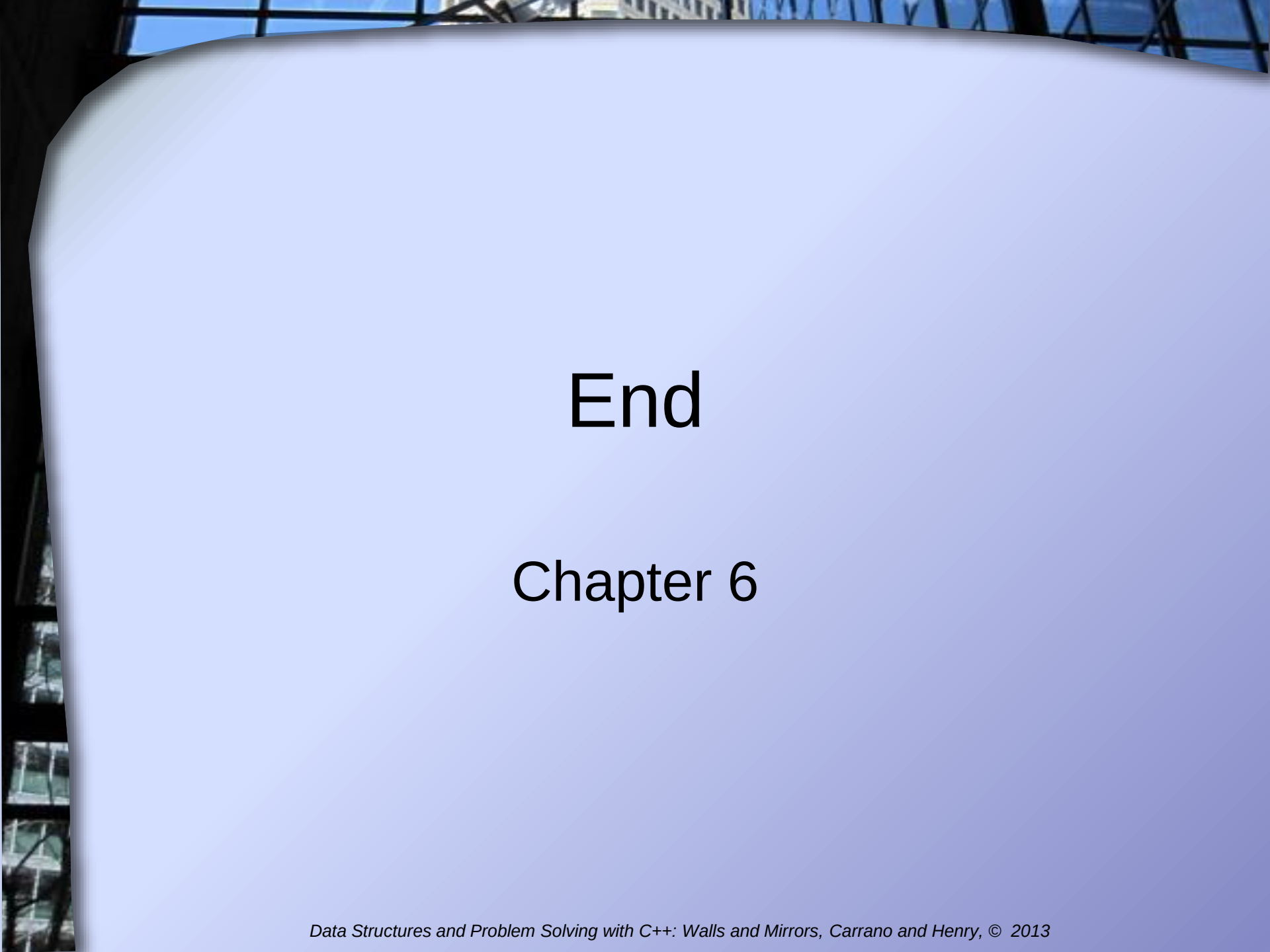
```
Initialize postfixExp to the empty string
for (each character ch in the infix expression)
{
    switch (ch)
    {
        case ch is an operand:
            Append ch to the end of postfixExp
            break
        case ch is an operator:
            Save ch until you know where to place it
            break
        case ch is a '(' or a ')':
            Discard ch
            break
    }
}
```

# Using Stacks with Algebraic Expressions

View pseudocode algorithm that converts infix to postfix
Listing 6-B

| ch | aStack (bottom to top) | postfixExp | |
|---|---|---|---|
| a | | a | |
| – | – | a | |
| ( | – ( | a | |
| b | – ( | ab | |
| + | – ( + | ab | |
| c | – ( + | abc | |
| * | – ( + * | abc | |
| d | – ( + * | abcd | |
| ) | – ( + | abcd* | Move operators from stack to postfixExp until "( " |
| | – ( | abcd*+ | |
| | – | abcd*+ | |
| / | – / | abcd*+ | Copy operators from stack to postfixExp |
| e | – / | abcd*+e | |
| | | abcd*+e/– | |

FIGURE 6-5 A trace of the algorithm that converts the infix expression a – ( b + c * d ) / e to postfix form



# End

## Chapter 6