Recursion as a Problem-Solving Technique

Chapter 5

Contents

Defining Languages

- A language is
  - A set of strings of symbols
  - From a finite alphabet.
- C++Programs = {string s : s is a syntactically correct C++ program}
- AlgebraicExpressions = {string s : s is an algebraic expression}

The Basics of Grammars

- Special symbols
  - x | y means x or y
  - xy (and sometimes x • y) means x followed by y
  - < word > means any instance of word, where word is a symbol that must be defined elsewhere in the grammar.
- C++Identifiers = {string s : s is a legal C++ identifier}

The Basics of Grammars

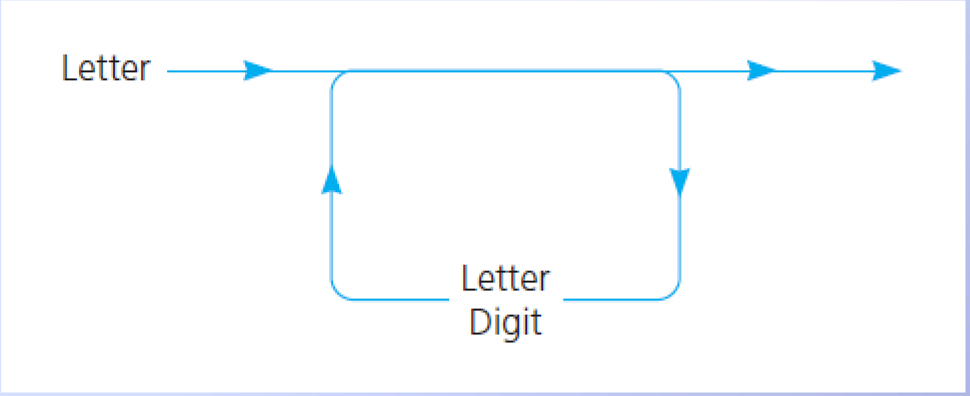
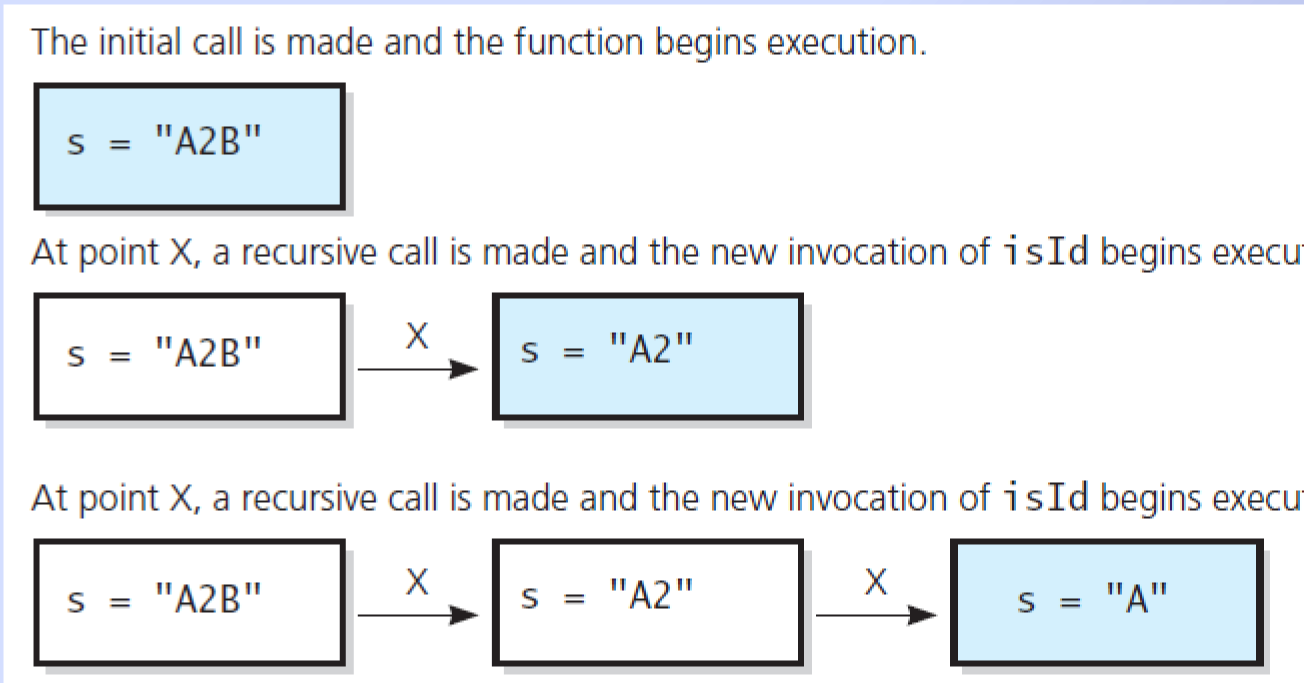


FIGURE 5-1 A syntax diagram for C++ identifiers

Recognition Algorithm for Identifiers



. . .

FIGURE 5-2 Trace of **`isId("A2B")`**

Recognition Algorithm for Identifiers

…

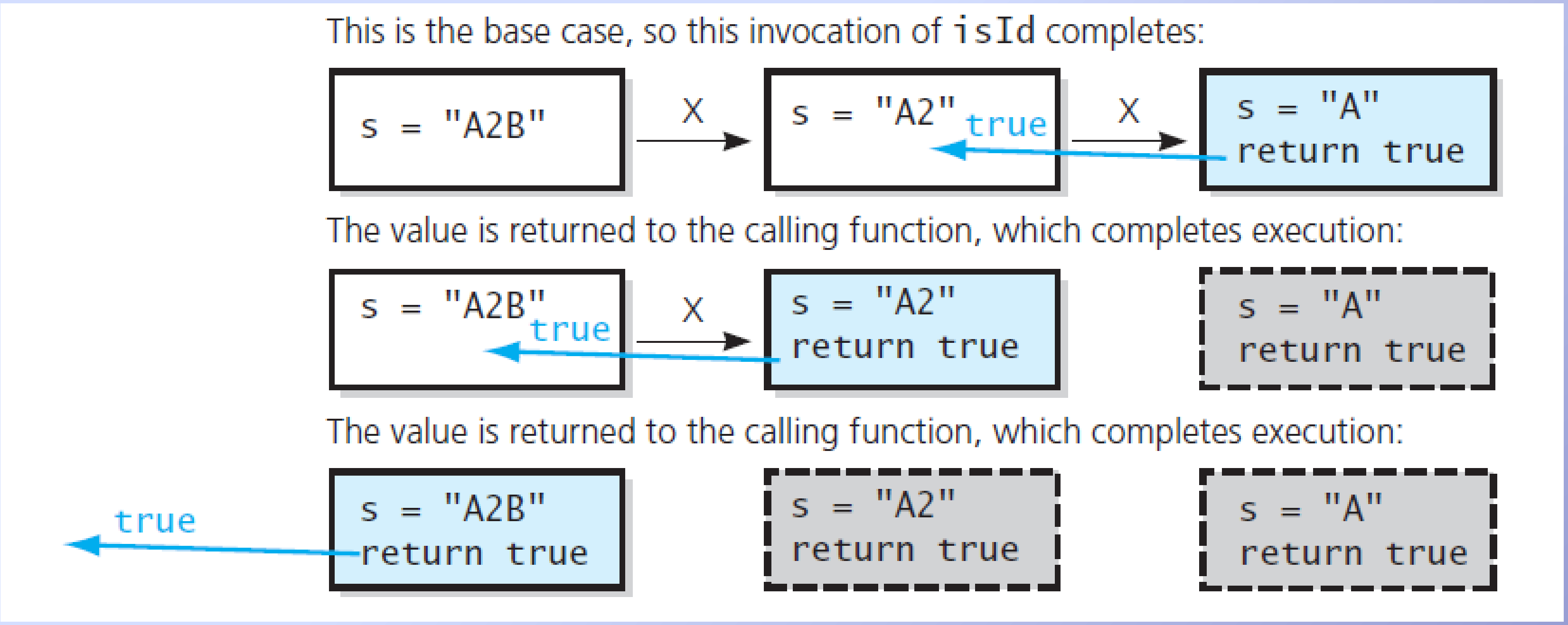


FIGURE 5-2 Trace of **`isId("A2B")`**

Two Simple Languages

- Palindromes = {string s : s reads the same left to right as right to left}
- Grammar for the language of palindromes:

$$
\begin{aligned}
<pal> \; &= \text{empty string} \mid <ch> \mid \text{a} <pal> \text{a} \mid \text{b} <pal> \text{b} \mid \ldots \mid \text{Z} <pal> \text{Z} \\
<ch> \; &= \text{a} \mid \text{b} \mid \ldots \mid \text{z} \mid \text{A} \mid \text{B} \mid \ldots \mid \text{Z}
\end{aligned}
$$

Two Simple Languages

- A recognition algorithm for palindromes

```
// Returns true if the string s of letters is a palindrome; otherwise returns false.
isPalindrome(s: string): boolean

   if (s is the empty string or s is of length 1)
      return true
   else if (s's first and last characters are the same letter)
      return isPalindrome(s minus its first and last characters)
   else
      return false
```

Algebraic Expressions

- Compiler must recognize and evaluate algebraic expressions
- Example
  `y = x + z * (w / k + z * (7 * 6));`
- Kinds of algebraic expressions
  - infix
  - prefix
  - postfix

Algebraic Expressions

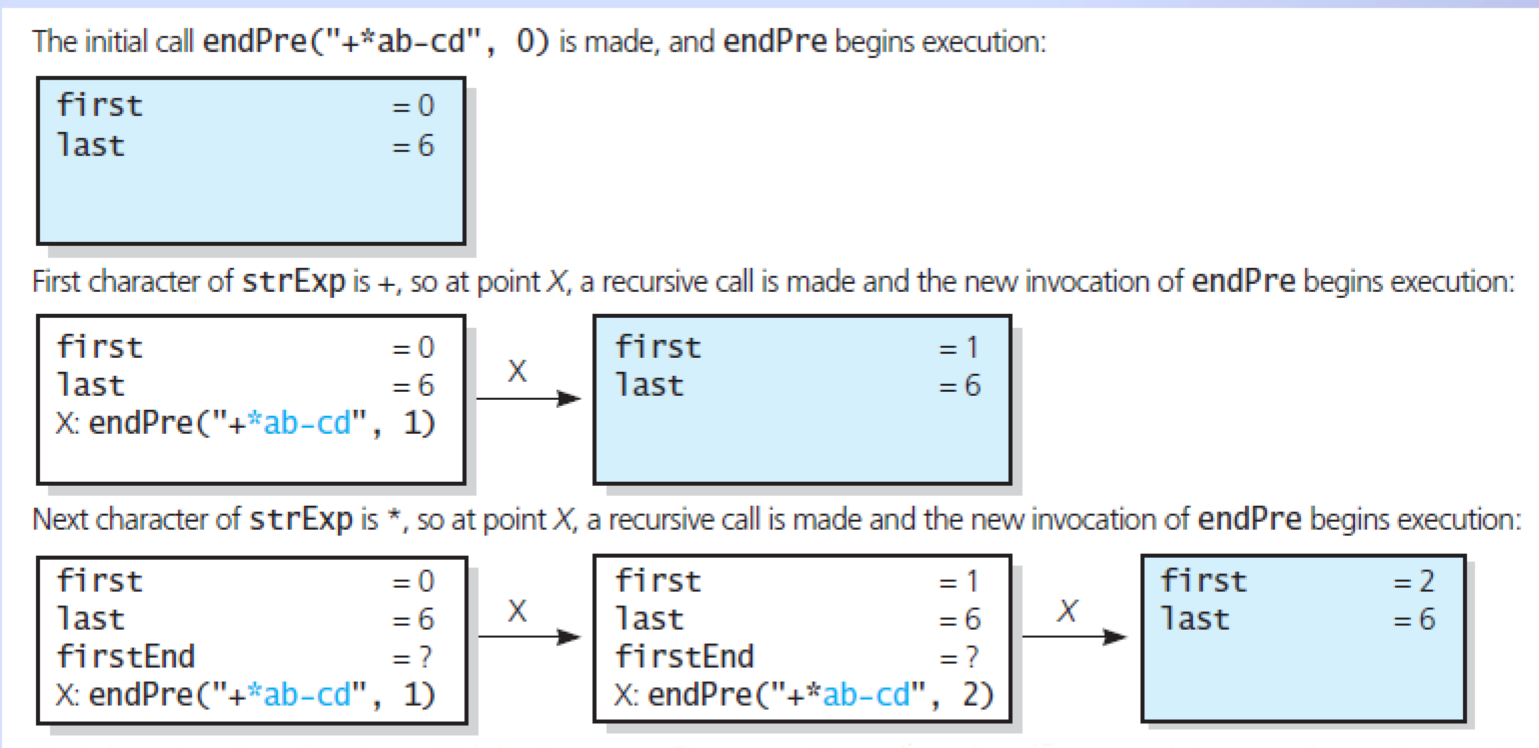
- infix
  - Binary operator appears between its operands
- prefix
  - Operator appears before its operands
- postfix
  - Operator appears after its operands

Prefix Expressions

- Grammar that defines language of all prefix expressions

$$\langle prefix \rangle = \langle identifier \rangle \mid \langle operator \rangle \langle prefix \rangle \langle prefix \rangle$$
$$\langle operator \rangle = + \mid - \mid * \mid /$$
$$\langle identifier \rangle = a \mid b \mid \ldots \mid z$$

Prefix Expressions

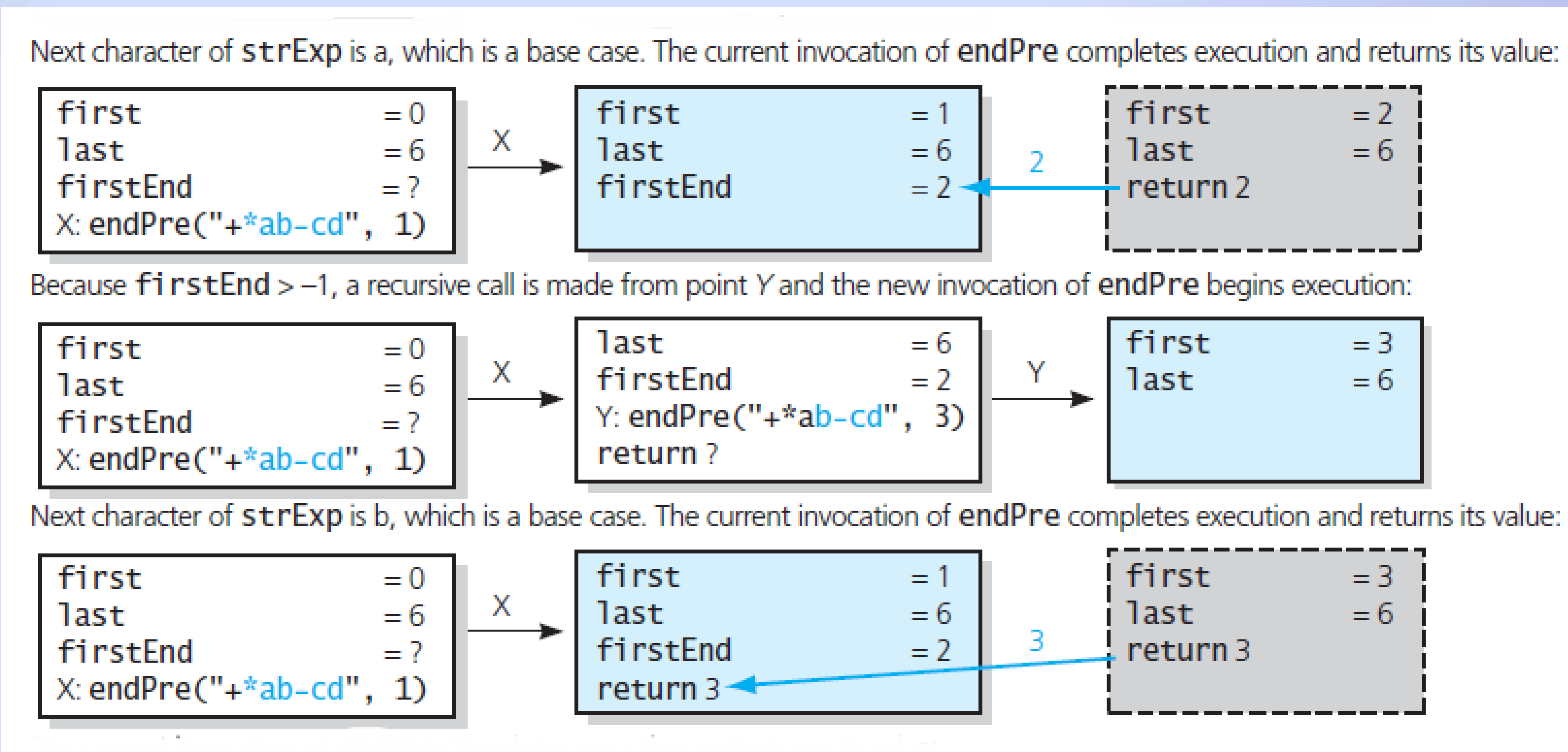


...

FIGURE 5-3 Trace of **`endPre( "+/ab-cd",0)`**

Prefix Expressions

...

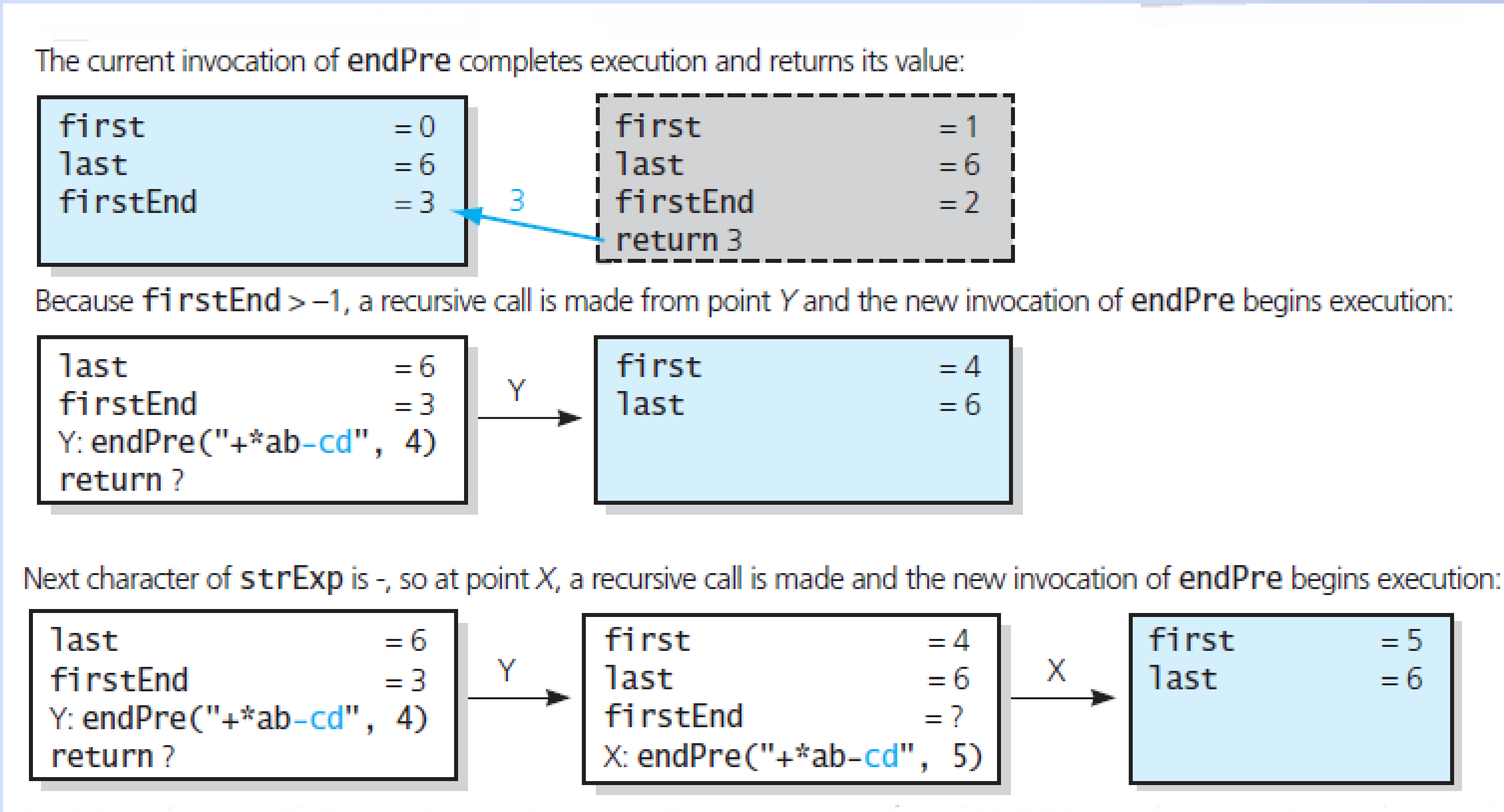


...

FIGURE 5-3 Trace of **`endPre( "+/ab-cd",0)`**

Prefix Expressions

…

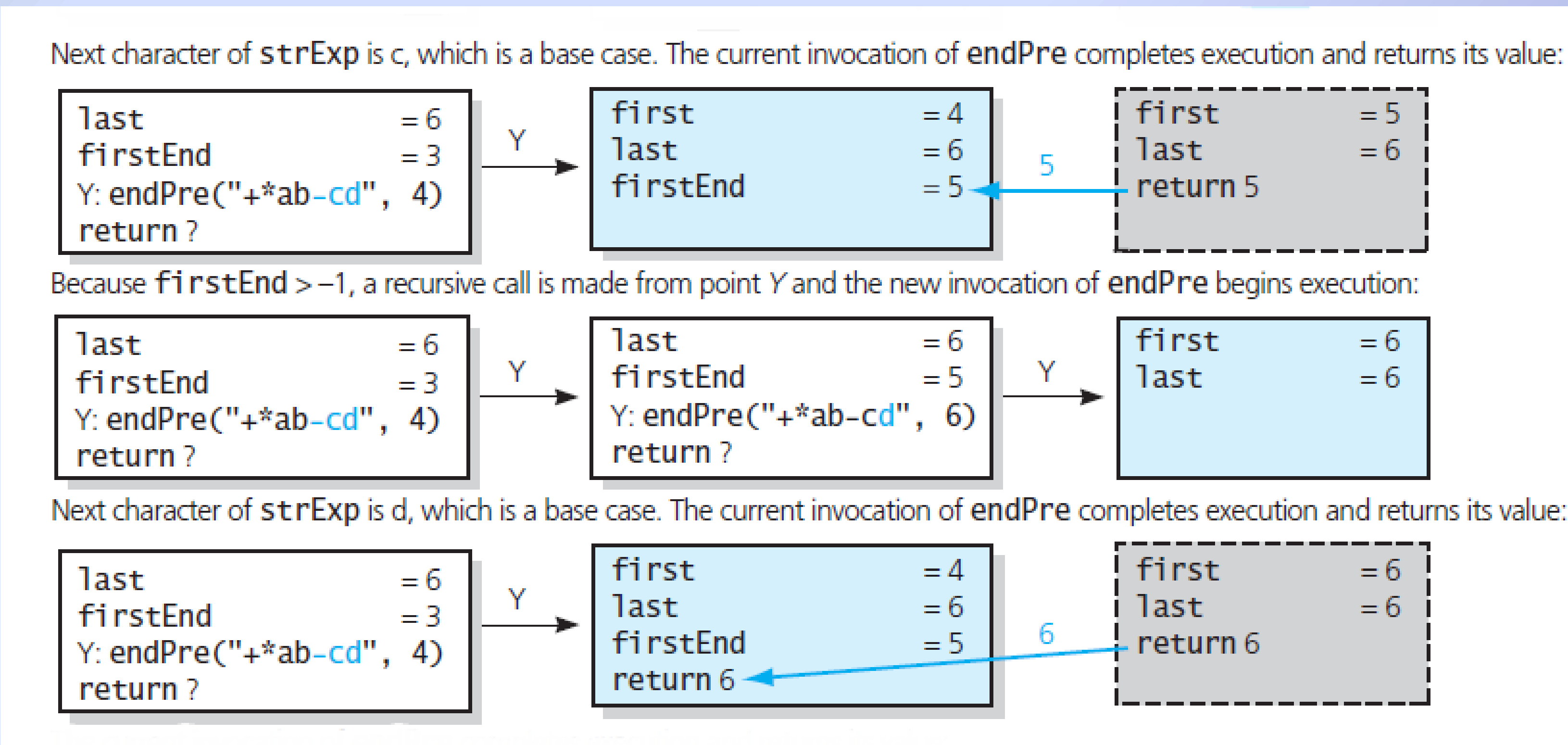


…

FIGURE 5-3 Trace of `endPre( "+/ab-cd",0)`

Prefix Expressions

. . .



. . .

FIGURE 5-3 Trace of **`endPre( "+/ab-cd",0)`**

Prefix Expressions

…

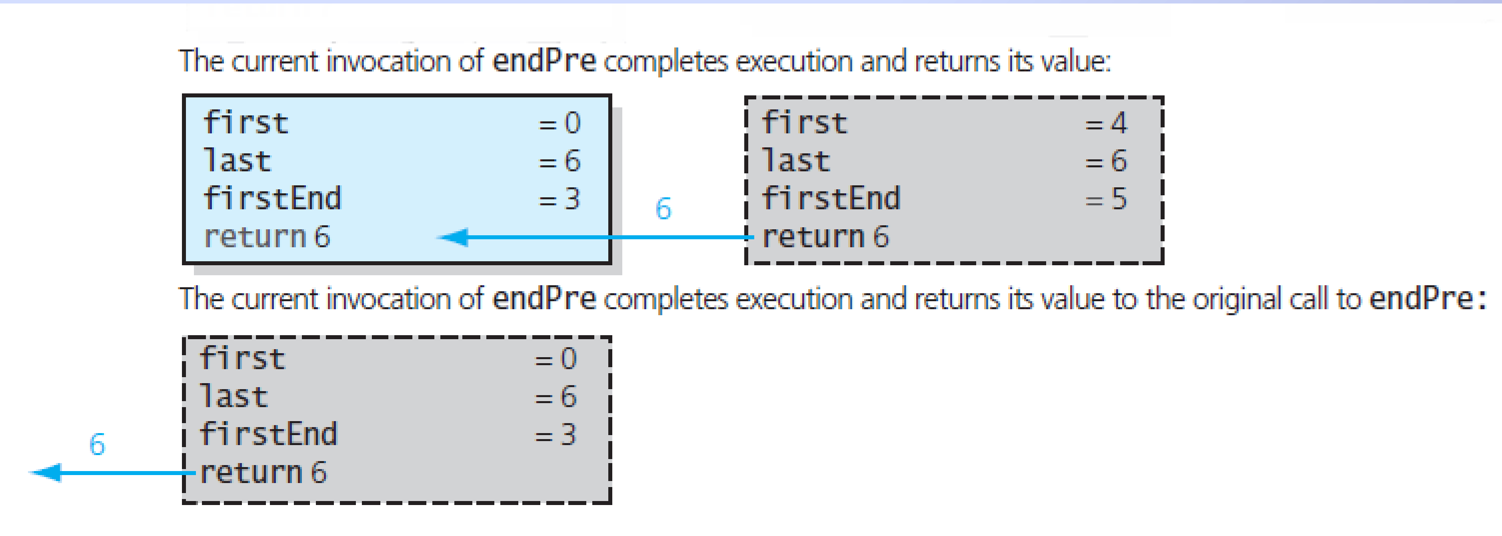


FIGURE 5-3 Trace of **`endPre( "+/ab-cd",0)`**

Postfix Expressions

- Grammar that defines language of postfix expressions

$$<postfix> \; = \; <identifier> \,|\, <postfix><postfix><operator>$$
$$<operator> \; = \; + \,|\, - \,|\, * \,|\, /$$
$$<identifier> \; = \; a \,|\, b \,|\, \ldots \,|\, z$$

Fully Parenthesized Expressions

- Grammar that defines language of fully parenthesized infix expression

$$
\begin{aligned}
<infix> &= <identifier> \mid (<infix> <operator> <infix>) \\
<operator> &= + \mid - \mid * \mid / \\
<identifier> &= a \mid b \mid \ldots \mid z
\end{aligned}
$$

Backtracking

- Consider searching for an airline route
- Input text files that specify all of the flight information for HPAir Company
  - Names of cities HPAir serves
  - Pairs of city names, each pair representing origin and destination of one of HPAir’s flights
  - Pairs of city names, each pair representing a request to fly from some origin to some destination

Backtracking

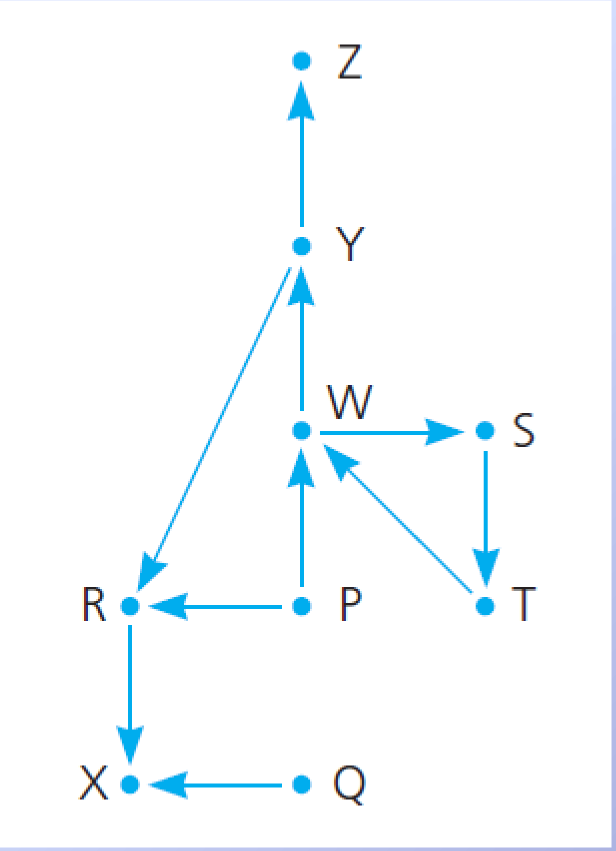


FIGURE 5-4 Flight map for HPAir

Backtracking

- A recursive strategy

To fly from the origin to the destination:

```
Select a city C adjacent to the origin
Fly from the origin to city C
if (C is the destination city)
    Terminate— the destination is reached
else
    Fly from city C to the destination
```

Backtracking

- Possible outcomes of applying the previous strategy
  1. Eventually reach destination city and can conclude that it is possible to fly from origin to destination.
  2. Reach a city C from which there are no departing flights.
  3. Go around in circles.

Backtracking

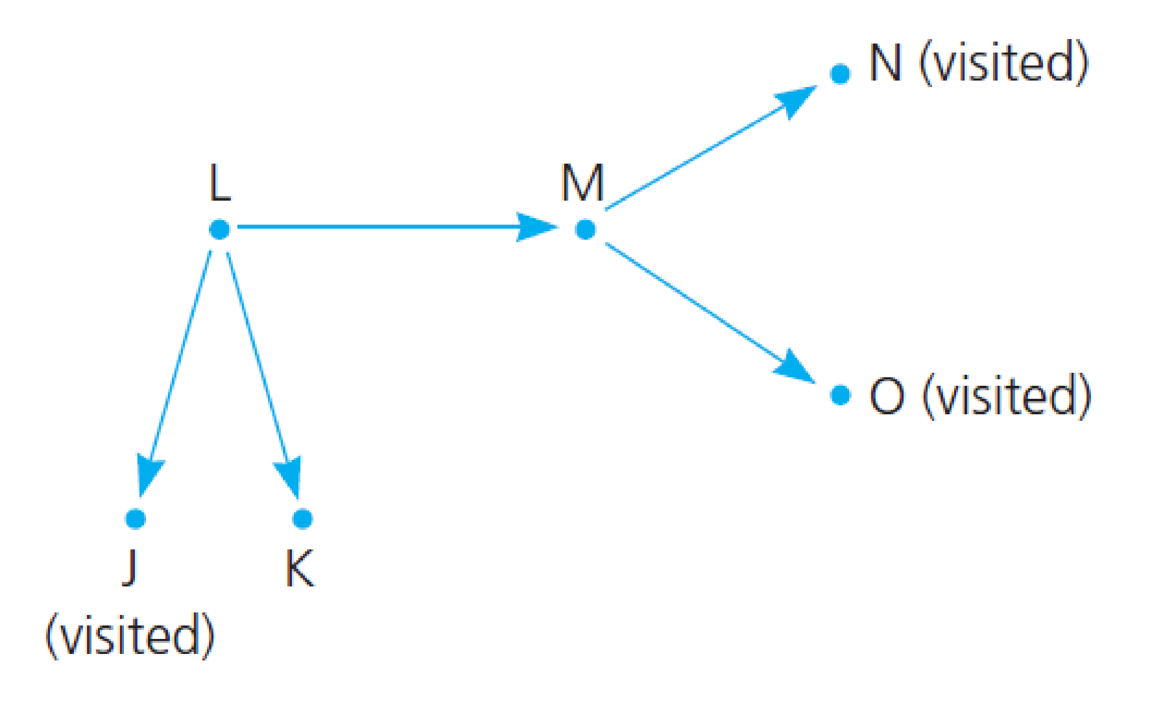


FIGURE 5-5 A piece of a flight map

Backtracking

- Note possible operations for ADT flight map, Listing 5-A
- View source code for C++ implementation of **`searchR`**, Listing 5-B

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

FIGURE 5-6 Flight map for Checkpoint Question 6

The Eight Queens Problem

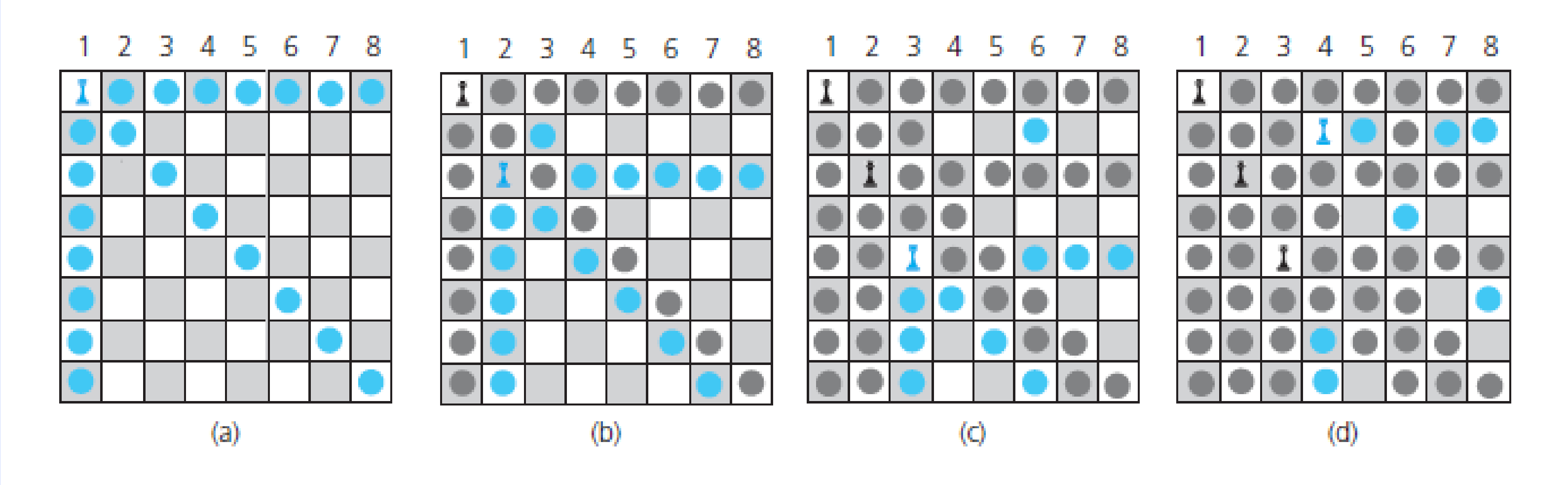


FIGURE 5-7 Placing one queen at a time in each column, and the placed queens' range of attack: (a) the first queen in column 1; (b) the second queen in column 2; (c) the third queen in column 3; (d) the fourth queen in column 4;

The Eight Queens Problem

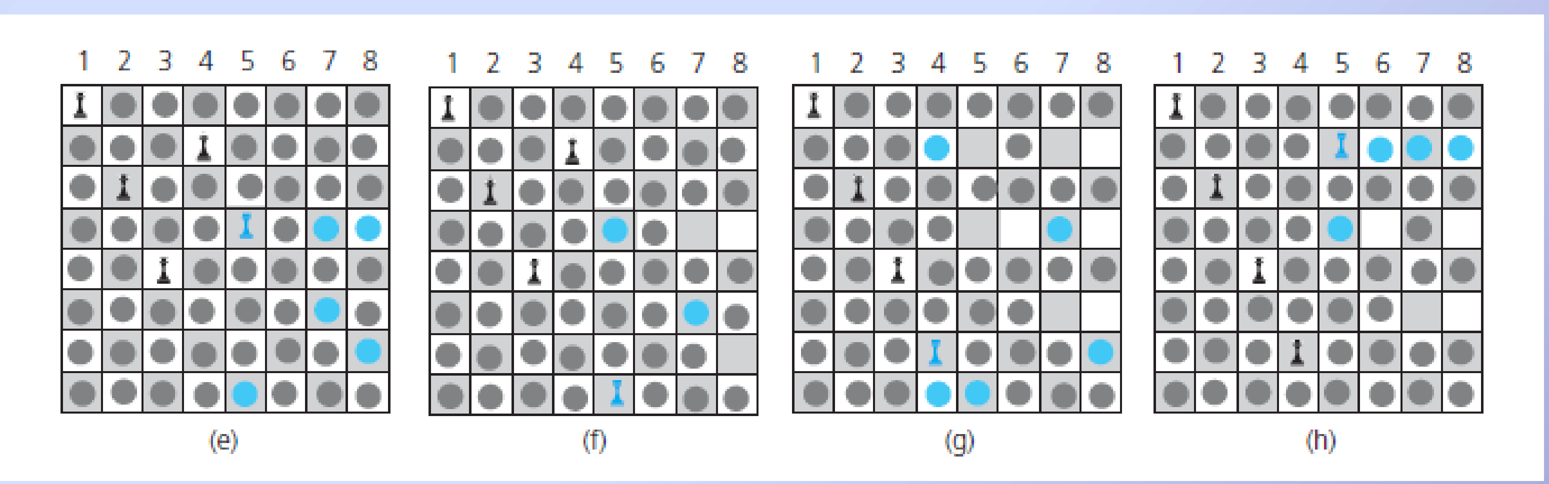


FIGURE 5-7 Placing one queen at a time in each column, and the placed queens' range of attack: (e) five queens can attack all of column 6; (f) backtracking to column 5 to try another square for queen; (g) backtracking to column 4 to try another square for the queen; (h) considering column 5 again

The Eight Queens Problem

- View pseudocode of algorithm for placing queens in columns, Listing 5-C

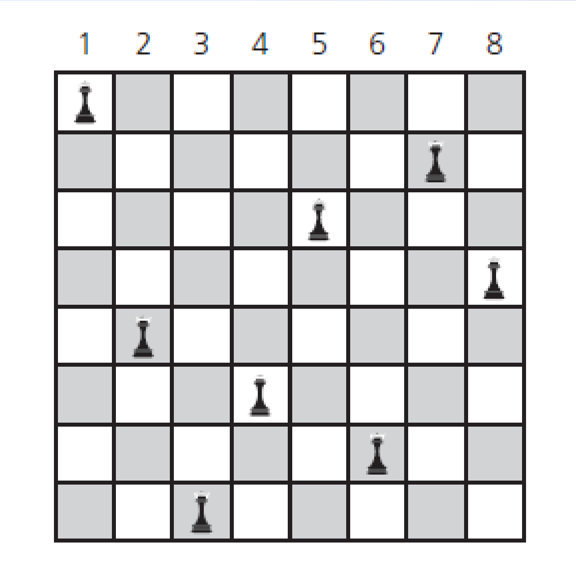


FIGURE 5-8 A solution to the Eight Queens problem

The Eight Queens Problem

- Note header file for the **`Board`** class, Listing 5-1
- View source code for class **`Queen`**, Listing 5-2
- And inspect an implementation of **`placeQueen`**, Listing 5-D

Correctness of the Recursive Factorial Function

- A recursive function that computes the factorial of a nonnegative integer *n*

```
fact(n: integer): integer

   if (n is 0)
      return 1
   else
      return n * fact(n - 1)
```

Correctness of the Recursive Factorial Function

- Assume property true for k = n

$$factorial(k) = k! = k \cdot (k-1) \cdot (k-2) \cdot \ldots \cdot 1$$

- Now show

$$factorial(k+1) = (k+1) \cdot k \cdot (k-1) \cdot (k-2) \cdot \ldots \cdot 2 \cdot 1$$

The Cost of Towers of Hanoi

- Recall solution to the Towers of Hanoi problem

```
solveTowers(count, source, destination, spare)

   if (count is 1)
      Move a disk directly from source to destination
   else
   {
      solveTowers(count - 1, source, spare, destination)
      solveTowers(1, source, destination, spare)
      solveTowers(count - 1, spare, destination, source)
   }
```

The Cost of Towers of Hanoi

- Consider … begin with *N* disks, how many moves does **`solveTowers`** make to solve problem?
- We conjecture $$moves(N) = 2^N - 1 \quad \text{for all N} \geq 1$$

- Make assumption for N = k
- Must show $$moves(k+1) = 2^{k+1} - 1$$

End

Chapter 5