

Chapter 8: Data Abstractions

**Computer Science: An Overview
Eleventh Edition**

**by
J. Glenn Brookshear**

Addison-Wesley
is an imprint of
PEARSON

Copyright © 2012 Pearson Education, Inc.

Chapter 8: Data Abstractions

- 8.1 Data Structure Fundamentals
- 8.2 Implementing Data Structures
- 8.3 A Short Case Study
- 8.4 Customized Data Types
- 8.5 Classes and Objects
- 8.6 Pointers in Machine Language

Copyright © 2012 Pearson Education, Inc.

0-2

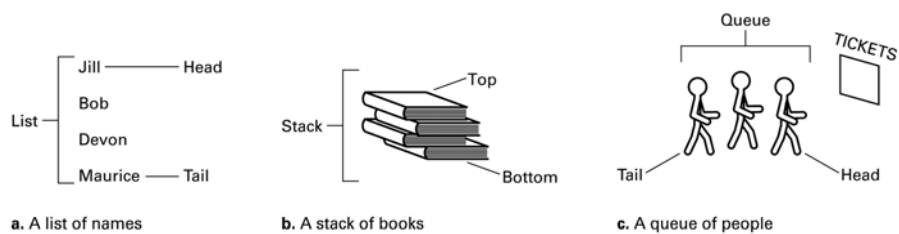
Basic Data Structures

- Homogeneous array
- Heterogeneous array
- List
 - Stack
 - Queue
- Tree

Copyright © 2012 Pearson Education, Inc.

0-3

Figure 8.1 Lists, stacks, and queues



Copyright © 2012 Pearson Education, Inc.

0-4

Terminology for Lists

- **List:** A collection of data whose entries are arranged sequentially
- **Head:** The beginning of the list
- **Tail:** The end of the list

Terminology for Stacks

- **Stack:** A list in which entries are removed and inserted only at the head
- **LIFO:** Last-in-first-out
- **Top:** The head of list (stack)
- **Bottom** or **base:** The tail of list (stack)
- **Pop:** To remove the entry at the top
- **Push:** To insert an entry at the top

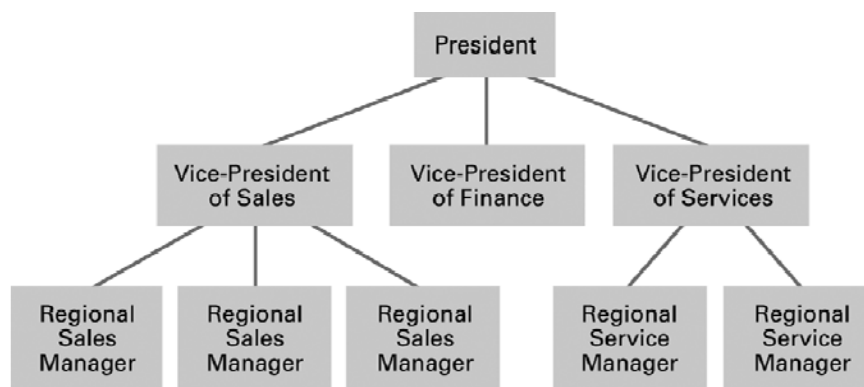
Terminology for Queues

- **Queue:** A list in which entries are removed at the head and are inserted at the tail
- **FIFO:** First-in-first-out

Copyright © 2012 Pearson Education, Inc.

0-7

Figure 8.2 An example of an organization chart



Copyright © 2012 Pearson Education, Inc.

0-8

Terminology for a Tree

- **Tree:** A collection of data whose entries have a hierarchical organization
- **Node:** An entry in a tree
- **Root node:** The node at the top
- **Terminal or leaf node:** A node at the bottom

Copyright © 2012 Pearson Education, Inc.

0-9

Terminology for a Tree (continued)

- **Parent:** The node immediately above a specified node
- **Child:** A node immediately below a specified node
- **Ancestor:** Parent, parent of parent, etc.
- **Descendent:** Child, child of child, etc.
- **Siblings:** Nodes sharing a common parent

Copyright © 2012 Pearson Education, Inc.

0-10

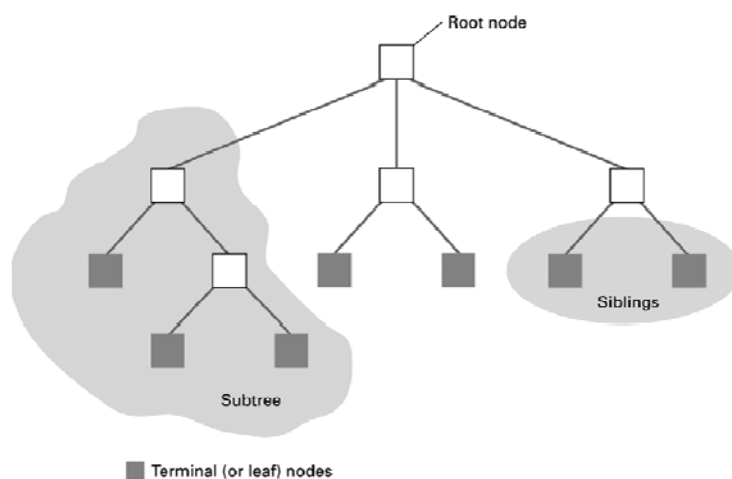
Terminology for a Tree (continued)

- **Binary tree:** A tree in which every node has at most two children
- **Depth:** The number of nodes in longest path from root to leaf

Copyright © 2012 Pearson Education, Inc.

0-11

Figure 8.3 Tree terminology



Copyright © 2012 Pearson Education, Inc.

0-12

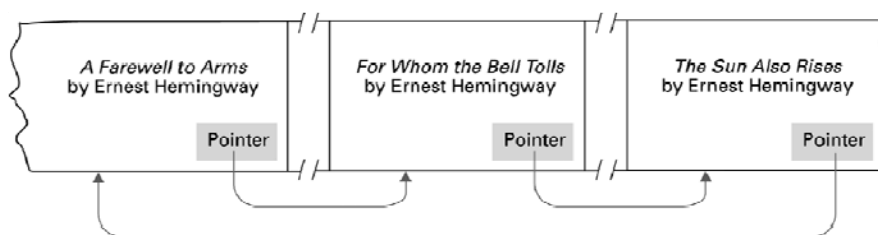
Additional Concepts

- Static Data Structures: Size and shape of data structure does not change
- Dynamic Data Structures: Size and shape of data structure can change
- Pointers: Used to locate data

Copyright © 2012 Pearson Education, Inc.

0-13

Figure 8.4 Novels arranged by title but linked according to authorship



Copyright © 2012 Pearson Education, Inc.

0-14

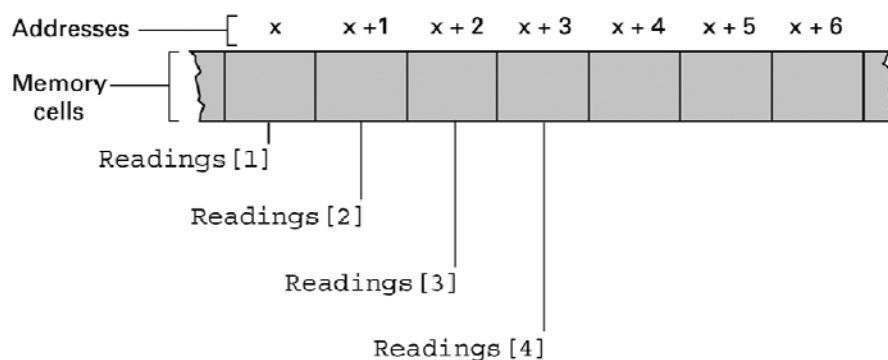
Storing Arrays

- Homogeneous arrays
 - **Row-major order** versus **column major order**
 - Address polynomial
- Heterogeneous arrays
 - Components can be stored one after the other in a contiguous block
 - Components can be stored in separate locations identified by pointers

Copyright © 2012 Pearson Education, Inc.

0-15

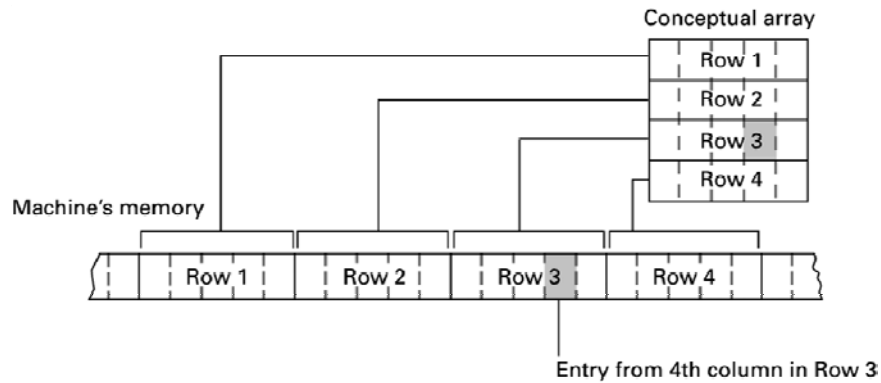
Figure 8.5 The array of temperature readings stored in memory starting at address x



Copyright © 2012 Pearson Education, Inc.

0-16

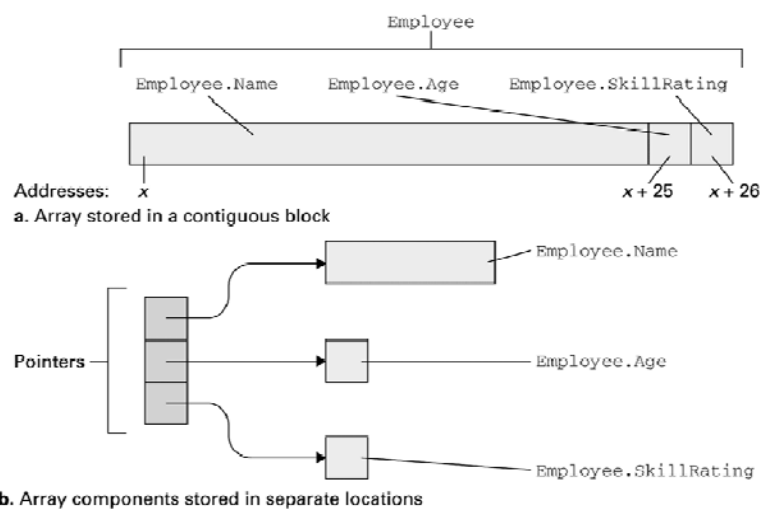
Figure 8.6 A two-dimensional array with four rows and five columns stored in row major order



Copyright © 2012 Pearson Education, Inc.

0-17

Figure 8.7 Storing the heterogeneous array Employee



Copyright © 2012 Pearson Education, Inc.

0-18

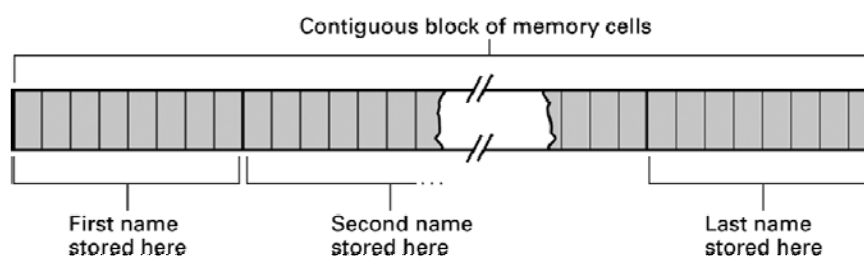
Storing Lists

- **Contiguous list:** List stored in a homogeneous array
- **Linked list:** List in which each entries are linked by pointers
 - **Head pointer:** Pointer to first entry in list
 - **NIL pointer:** A “non-pointer” value used to indicate end of list

Copyright © 2012 Pearson Education, Inc.

0-19

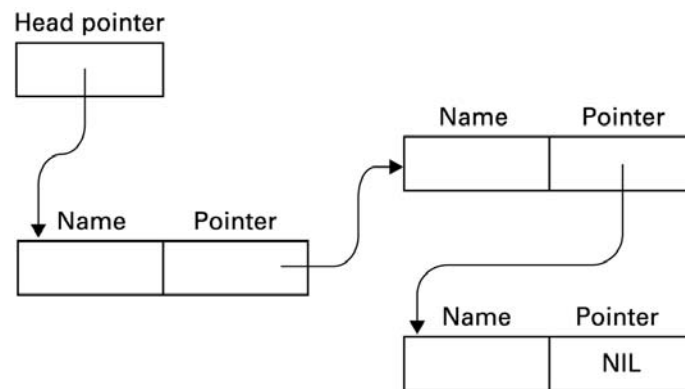
Figure 8.8 Names stored in memory as a contiguous list



Copyright © 2012 Pearson Education, Inc.

0-20

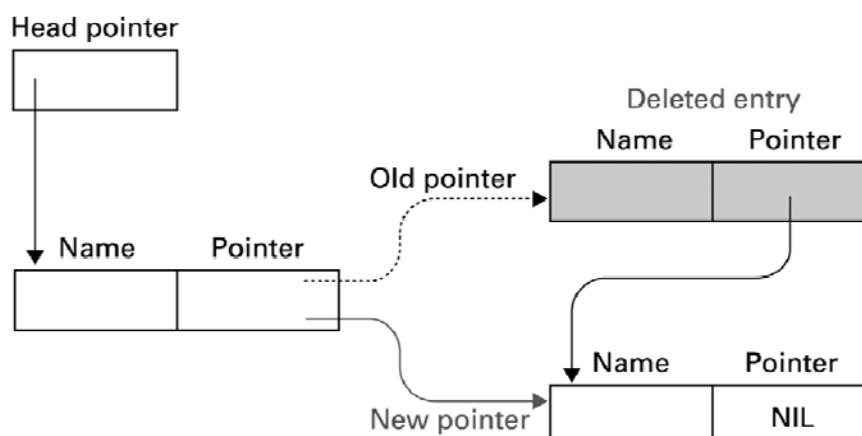
Figure 8.9 The structure of a linked list



Copyright © 2012 Pearson Education, Inc.

0-21

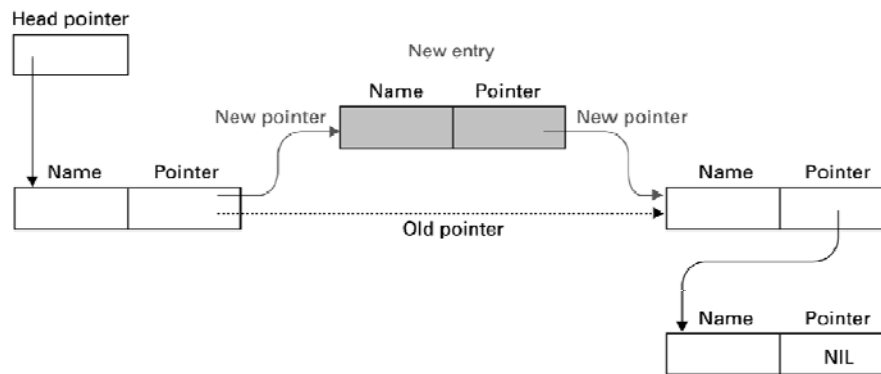
Figure 8.10 Deleting an entry from a linked list



Copyright © 2012 Pearson Education, Inc.

0-22

Figure 8.11 Inserting an entry into a linked list



Copyright © 2012 Pearson Education, Inc.

0-23

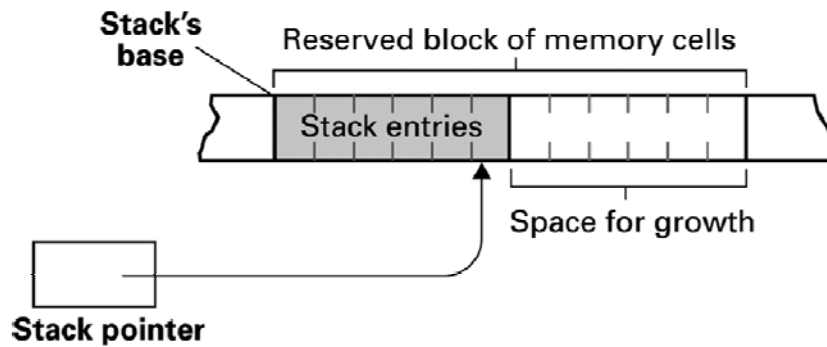
Storing Stacks and Queues

- Stacks usually stored as contiguous lists
- Queues usually stored as **Circular Queues**
 - Stored in a contiguous block in which the first entry is considered to follow the last entry
 - Prevents a queue from crawling out of its allotted storage space

Copyright © 2012 Pearson Education, Inc.

0-24

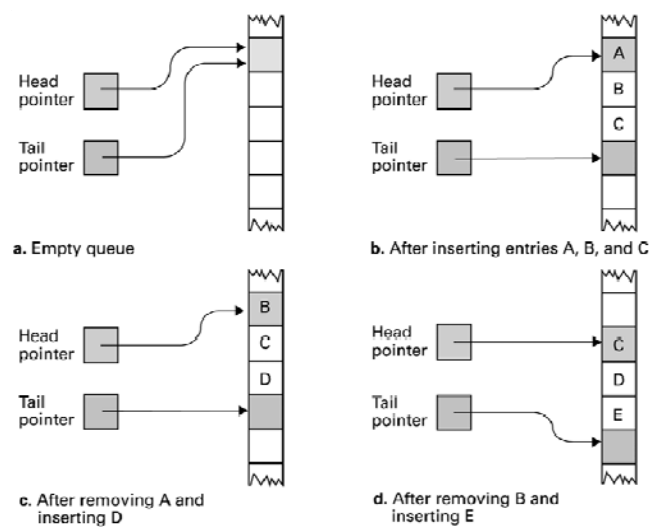
Figure 8.12 A stack in memory



Copyright © 2012 Pearson Education, Inc.

0-25

Figure 8.13 A queue implementation with head and tail pointers



Copyright © 2012 Pearson Education, Inc.

0-26

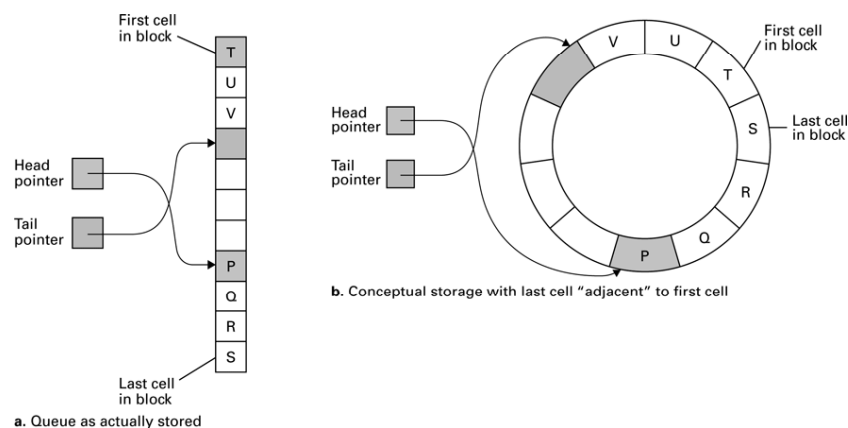
Storing Binary Trees

- Linked structure
 - Each node = data cells + two child pointers
 - Accessed via a pointer to root node
- Contiguous array structure
 - $A[1]$ = root node
 - $A[2], A[3]$ = children of $A[1]$
 - $A[4], A[5], A[6], A[7]$ = children of $A[2]$ and $A[3]$

Copyright © 2012 Pearson Education, Inc.

0-27

Figure 8.14 A circular queue containing the letters P through V



Copyright © 2012 Pearson Education, Inc.

0-28

Figure 8.15 The structure of a node in a binary tree

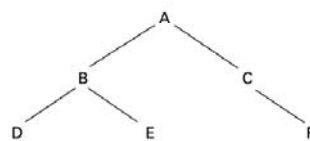


Copyright © 2012 Pearson Education, Inc.

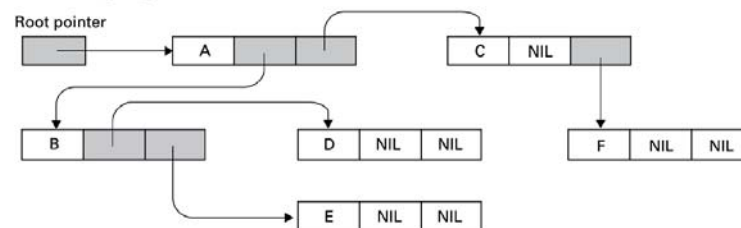
0-29

Figure 8.16 The conceptual and actual organization of a binary tree using a linked storage system

Conceptual tree



Actual storage organization

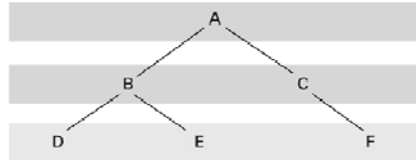


Copyright © 2012 Pearson Education, Inc.

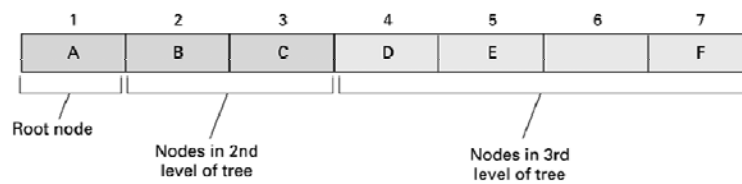
0-30

Figure 8.17 A tree stored without pointers

Conceptual tree



Actual storage organization

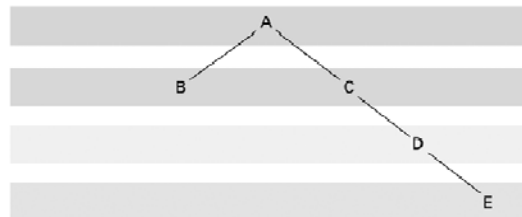


Copyright © 2012 Pearson Education, Inc.

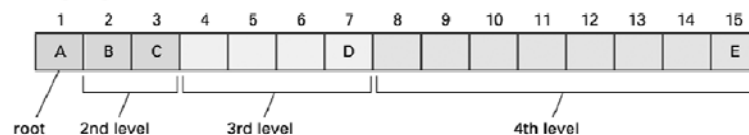
0-31

Figure 8.18 A sparse, unbalanced tree shown in its conceptual form and as it would be stored without pointers

Conceptual tree



Actual storage organization



Copyright © 2012 Pearson Education, Inc.

0-32

Manipulating Data Structures

- Ideally, a data structure should be manipulated solely by pre-defined procedures.
 - Example: A stack typically needs at least `push` and `pop` procedures.
 - The data structure along with these procedures constitutes a complete abstract tool.

Copyright © 2012 Pearson Education, Inc.

0-33

Figure 8.19 A procedure for printing a linked list

```
procedure PrintList (List)
  CurrentPointer ← head pointer of List.
  while (CurrentPointer is not NIL) do
    (Print the name in the entry pointed to by CurrentPointer;
     Observe the value in the pointer cell of the List entry
     pointed to by CurrentPointer, and reassign CurrentPointer
     to be that value.)
```

Copyright © 2012 Pearson Education, Inc.

0-34

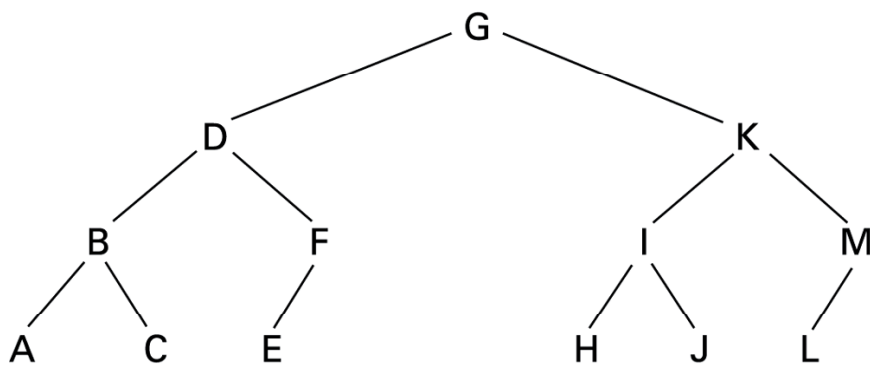
Case Study

Problem: Construct an abstract tool consisting of a list of names in alphabetical order along with the operations search, print, and insert.

Copyright © 2012 Pearson Education, Inc.

0-35

Figure 8.20 The letters A through M arranged in an ordered tree



Copyright © 2012 Pearson Education, Inc.

0-36

Figure 8.21 The binary search as it would appear if the list were implemented as a linked binary tree

```

procedure Search(Tree, TargetValue)

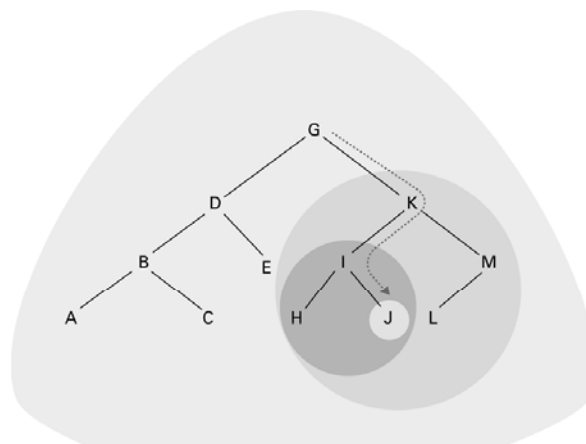
if (root pointer of Tree = NIL)
  then
    (declare the search a failure)
  else
    (execute the block of instructions below that is
     associated with the appropriate case)
    case 1: TargetValue = value of root node
      (Report that the search succeeded)
    case 2: TargetValue < value of root node
      (Apply the procedure Search to see if
       TargetValue is in the subtree identified
       by the root's left child pointer and
       report the result of that search)
    case 3: TargetValue > value of root node
      (Apply the procedure Search to see if
       TargetValue is in the subtree identified
       by the root's right child pointer and
       report the result of that search)
  ) end if

```

Copyright © 2012 Pearson Education, Inc.

0-37

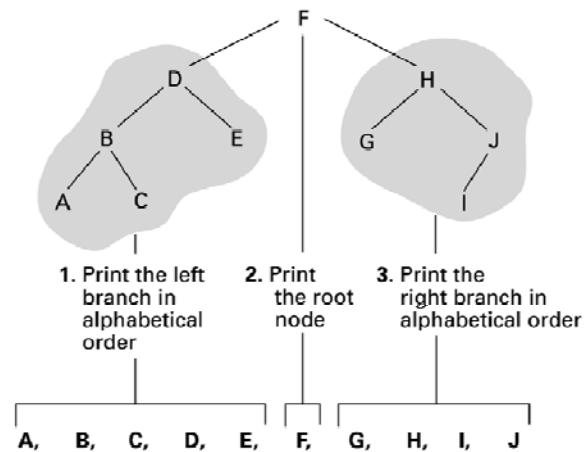
Figure 8.22 The successively smaller trees considered by the procedure in Figure 8.18 when searching for the letter J



Copyright © 2012 Pearson Education, Inc.

0-38

Figure 8.23 Printing a search tree in alphabetical order



Copyright © 2012 Pearson Education, Inc.

0-39

Figure 8.24 A procedure for printing the data in a binary tree

procedure PrintTree (Tree)

if (Tree is not empty)

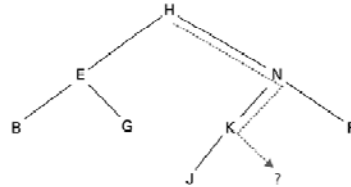
then (Apply the procedure PrintTree to the tree that appears as the left branch in Tree;
Print the root node of Tree;
Apply the procedure PrintTree to the tree that appears as the right branch in Tree)

Copyright © 2012 Pearson Education, Inc.

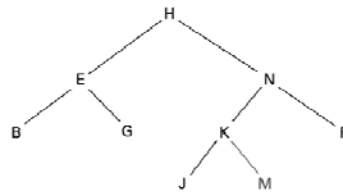
0-40

Figure 8.25 Inserting the entry M into the list B, E, G, H, J, K, N, P stored as a tree

a. Search for the new entry until its absence is detected



b. This is the position in which the new entry should be attached



Copyright © 2012 Pearson Education, Inc.

0-41

Figure 8.26 A procedure for inserting a new entry in a list stored as a binary tree

```

procedure Insert(Tree, NewValue)
if (root pointer of Tree = NIL)
  (set the root pointer to point to a new leaf
   containing NewValue)
else (execute the block of instructions below that is
  associated with the appropriate case)
  case 1: NewValue = value of root node
    (Do nothing)
  case 2: NewValue < value of root node
    if (left child pointer of root node = NIL)
      then (set that pointer to point to a new
        leaf node containing NewValue)
      else (apply the procedure Insert to insert
        NewValue into the subtree identified
        by the left child pointer)
  case 3: NewValue > value of root node
    if (right child pointer of root node = NIL)
      then (set that pointer to point to a new
        leaf node containing NewValue)
      else (apply the procedure Insert to insert
        NewValue into the subtree identified
        by the right child pointer)
    ) end if
  ) end if
  
```

Copyright © 2012 Pearson Education, Inc.

0-42

User-defined Data Type

- A template for a heterogeneous structure
- Example:

```
define type EmployeeType to be  
{char      Name[25];  
  int      Age;  
  real     SkillRating;  
}
```

Abstract Data Type

- A user-defined data type with procedures for access and manipulation
- Example:

```
define type StackType to be  
{int StackEntries[20];  
  int StackPointer = 0;  
  procedure push(value)  
    {StackEntries[StackPointer] ← value;  
    StackPointer ← StackPointer + 1;  
  }  
  procedure pop . . .  
}
```

Class

- An abstract data type with extra features
 - Characteristics can be inherited
 - Contents can be encapsulated
 - Constructor methods to initialize new objects

Copyright © 2012 Pearson Education, Inc.

0-45

Figure 8.27 A stack of integers implemented in Java and C#

```
class StackOfIntegers
{private int[] StackEntries = new int[20];
  private int StackPointer = 0;

  public void push(int NewEntry)
  {if (StackPointer < 20)
    StackEntries[StackPointer++] = NewEntry;
  }

  public int pop()
  {if (StackPointer > 0) return StackEntries[--StackPointer];
   else return 0;
  }
}
```

Copyright © 2012 Pearson Education, Inc.

0-46

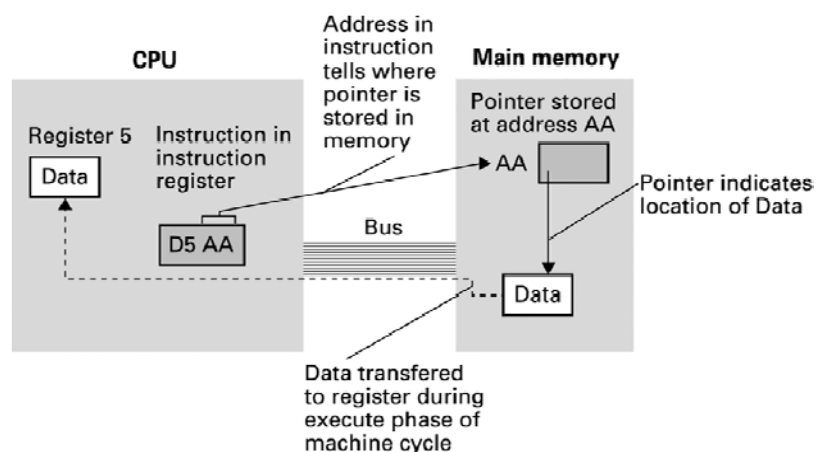
Pointers in Machine Language

- **Immediate addressing:** Instruction contains the data to be accessed
- **Direct addressing:** Instruction contains the address of the data to be accessed
- **Indirect addressing:** Instruction contains the location of the address of the data to be accessed

Copyright © 2012 Pearson Education, Inc.

0-47

Figure 8.28 Our first attempt at expanding the machine language in Appendix C to take advantage of pointers



Copyright © 2012 Pearson Education, Inc.

0-48

Figure 8.29 Loading a register from a memory cell that is located by means of a pointer stored in a register

