CMPS1134
Fundamentals of Computing

Data Abstractions 2

Computer Science: An Overview
Eleventh Edition
J. Glenn Brookshear
Chapter 8

Chapter 8: Data Abstractions

- Implementing Data Structures (continued)
  - Storing Stacks and Queues
  - Storing Binary Trees
  - Manipulating Data Structures
- A Short Case Study
- Customized Data Types
- Classes and Objects
- Pointers in Machine Language

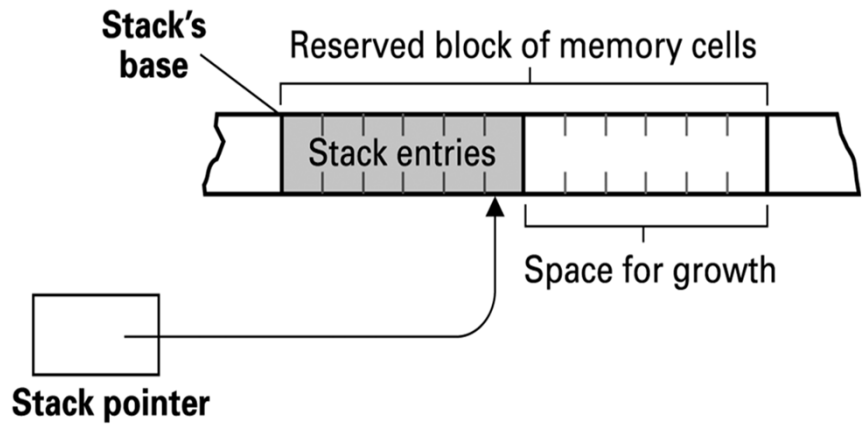
Implementing Data Structures

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

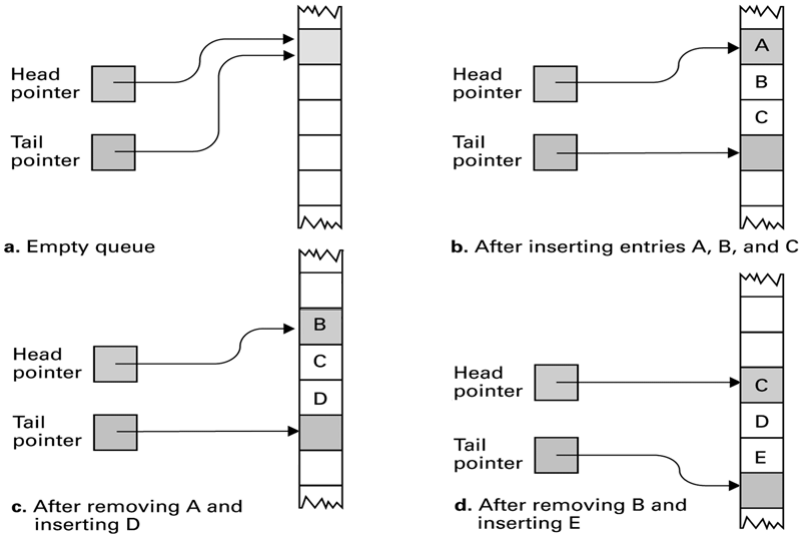
Implementing Data Structures - Storing Stacks and Queues

A stack in memory (Fig 8.12)



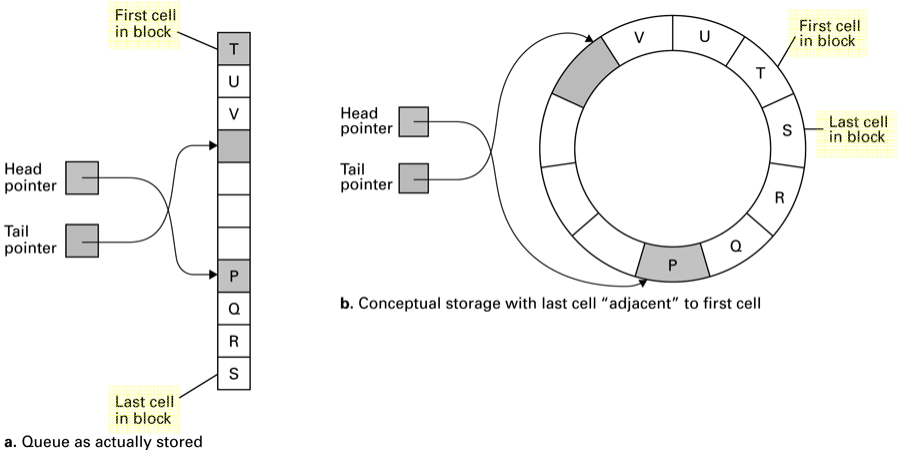
Implementing Data Structures - Storing Stacks and Queues

A queue implementation with head and tail pointers (Figure 8.13)



Implementing Data Structures - Storing Stacks and Queues

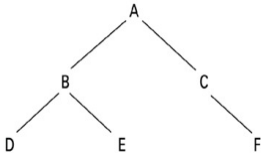
A circular queue containing the letters P through V (Fig 8.14)



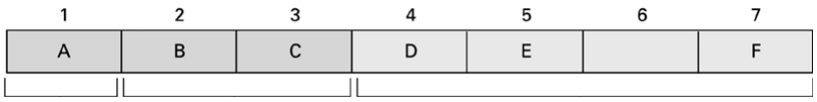
Implementing Data Structures

Storing Binary Trees

- Linked structure
  - Each node = data cells + two child pointers
  - Accessed via a pointer to root node

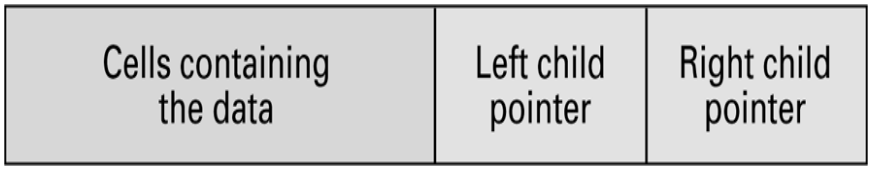


- Contiguous array structure
  - A[1] = root node
  - A[2],A[3] = children of A[1]
  - A[4],A[5],A[6],A[7] = children of A[2] and A[3]



Implementing Data Structures - Storing Binary Trees

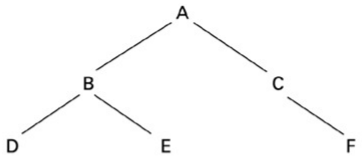
The structure of a node in a binary tree (Fig 8.15)



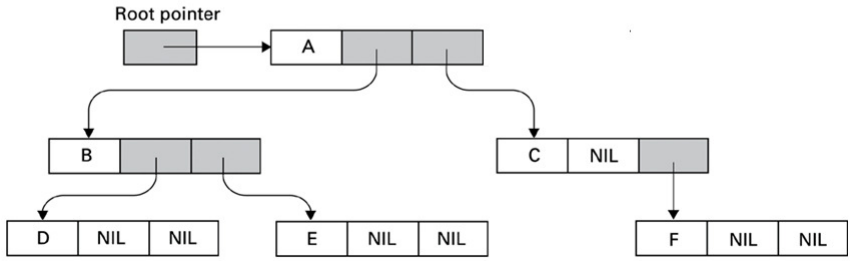
Implementing Data Structures - Storing Binary Trees

Conceptual & actual organization of a binary tree using a linked storage system (Fig 8.16)

Conceptual tree



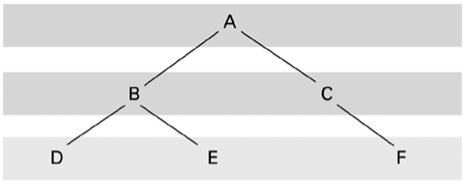
Actual storage organization



Implementing Data Structures - Storing Binary Trees

A tree stored without pointers (Fig 8.17)

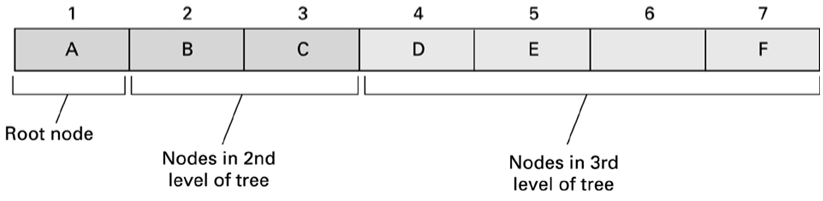
Conceptual tree



Node ***j*** :
Left child: **2*j***
Right child: **2*j*+1**
Parent: $\lfloor j/2 \rfloor$

Parental nodes:
First $\lfloor n/2 \rfloor$ locations

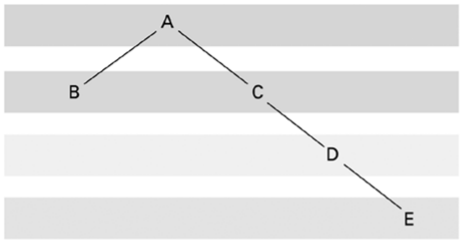
Actual storage organization



Implementing Data Structures - Storing Binary Trees

Sparse, unbalanced tree in its conceptual form & as it would be stored without pointers (Fig 8.18)

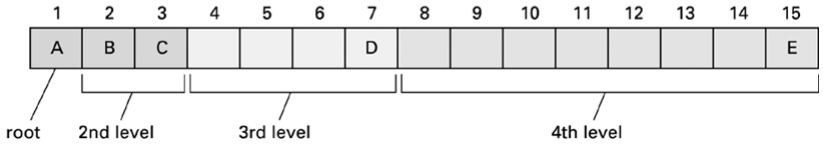
Conceptual tree



Node *j* :
Left child: ***2j***
Right child: ***2j+1***
Parent: $\lfloor j/2 \rfloor$

Parental nodes:
First $\lfloor n/2 \rfloor$ locations

Actual storage organization



Implementing Data Structures

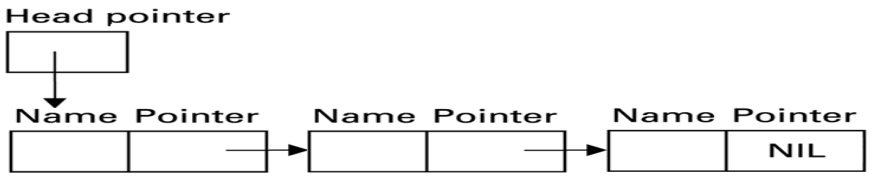
Manipulating Data Structures

- Ideally, a data structure should be manipulated solely by pre-defined procedures.
  - Example: A stack typically needs at least `push` and `pop` procedures.
  - Example: A list typically has a function `insert` for inserting new entries
  - The data structure along with the relevant procedures constitutes a complete abstract tool.

Implementing Data Structures - Manipulating Data Structures

A procedure for printing a linked list (Fig 8.19)

```
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.)
```



 13

Case Study

Problem:

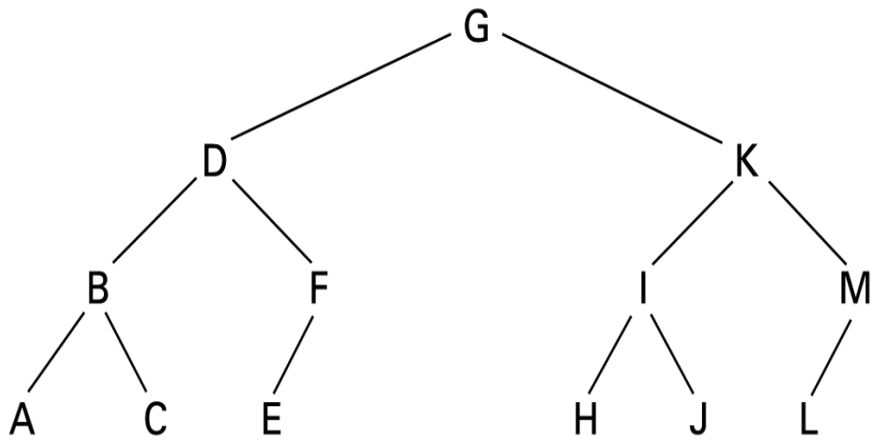
Construct an abstract tool consisting of:

- a list of names in alphabetical order
- along with the operations:
  - Search
  - Print
  - Insert

 14

Case Study

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



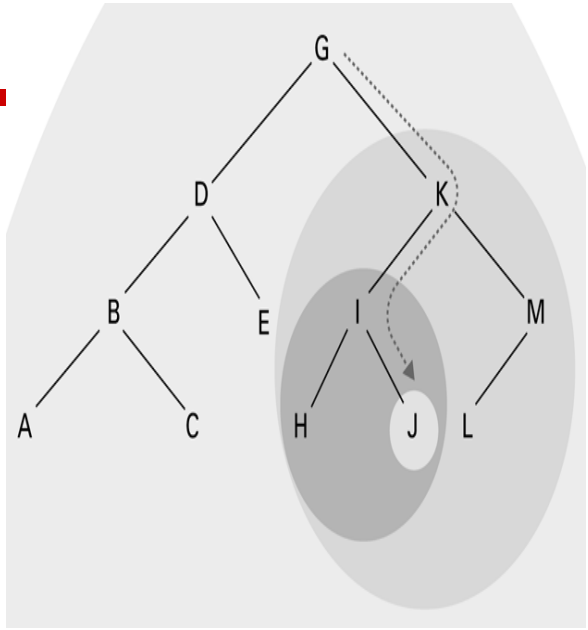
Case Study

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

```
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
```

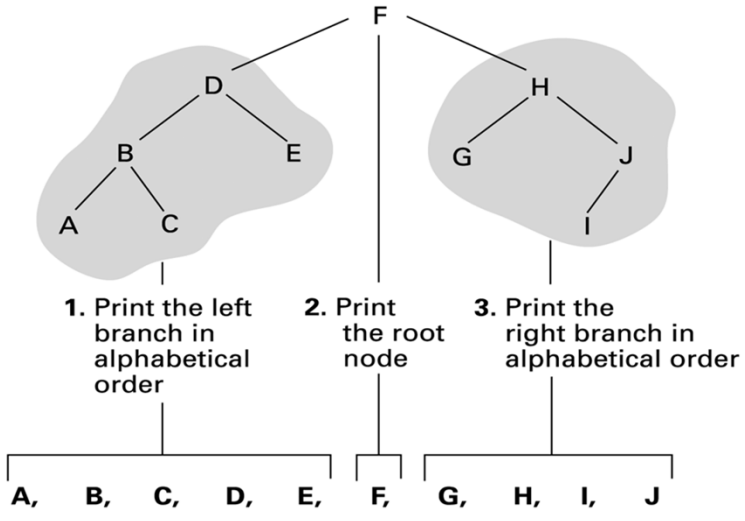

Case Study

The successively smaller trees considered by the procedure in Figure 8.21 when searching for the letter J (Fig 8.22)



Case Study

Printing tree in alphabetical order (Fig 8.23)



Case Study

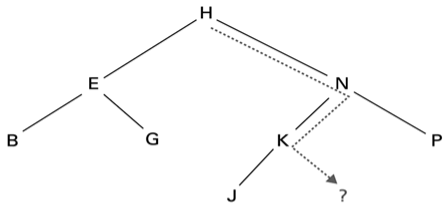
Procedure for printing binary tree data (Fig 8.24)

```
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)
```

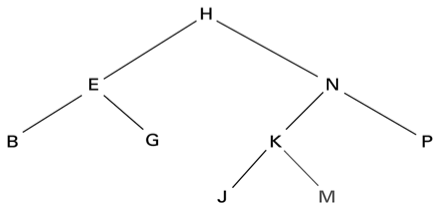

Case Study

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

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



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



Case Study

A procedure for inserting a new entry in a list stored as a binary tree (Fig 8.26)

```
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
```

 21

Customized Data Types

Data Types defined by programmers to fit more closely the needs of a particular application.

User-defined Data Type

- A template for a heterogeneous structure
  - Pseudocode 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
  - Pseudocode example:

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

 22

Classes and Objects

- An abstract data type with extra features
  - Characteristics can be inherited
  - Contents can be encapsulated
  - Constructor methods to initialize new objects
- A stack of integers implemented in Java and C# (Fig 8.27)

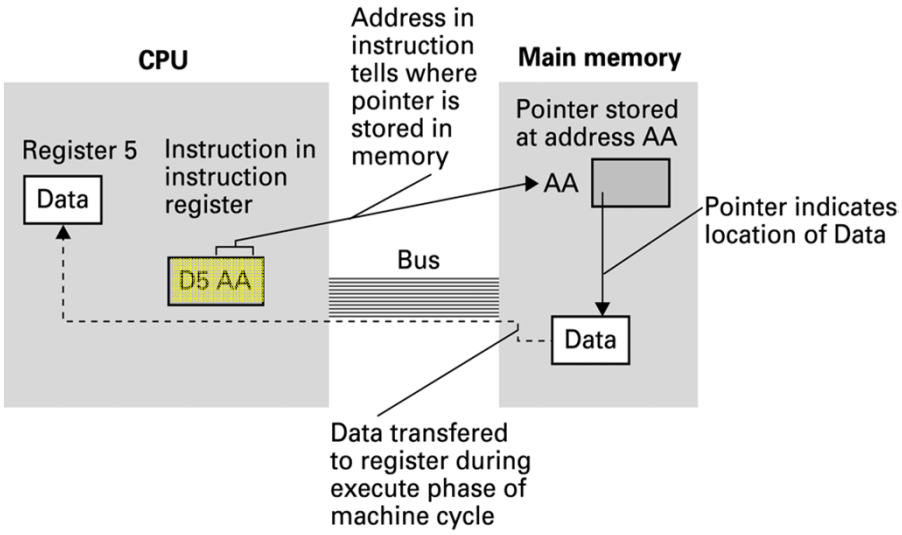
```
class StackOfIntegers
{private int[] StackEntries = new int[20];
 private int StackPointer = 0;

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

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


Pointers in Machine Language

- **Immediate addressing**: Instruction operand contains the data to be accessed
  - E.g. 20A3 : Load Register 0 with the value A3
- **Direct addressing**: Instruction operand is the address of the data to be accessed
  - E.g. 14A3 : Load Register 0 with value located at address A3
- **Indirect addressing**: Instruction operand is the address that contains the location of the data to be accessed
  - E.g. D5AA : Load Register 5 with the value located at the address found at address AA

Pointers in Machine Language

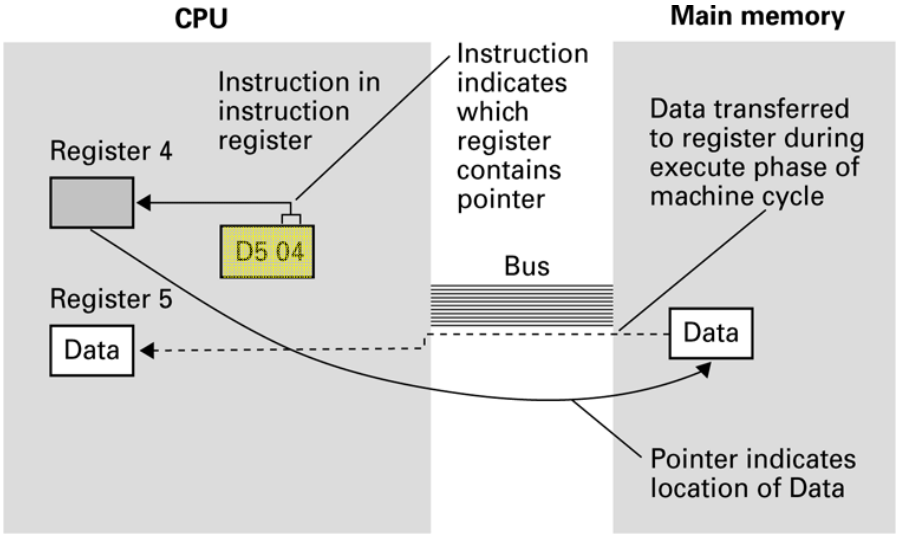
Loading a register from a memory cell located by means of a pointer stored in a memory cell (Fig 8.28)



 25

Pointers in Machine Language

Loading a register from a memory cell located by means of a pointer stored in a register (Fig 8.29)



 26

Chapter 8: Topics Covered

- Implementing Data Structures (continued)
  - Storing Stacks and Queues
  - Storing Binary Trees
  - Manipulating Data Structures
- A Short Case Study
- Customized Data Types
- Classes and Objects
- Pointers in Machine Language

