CSE 101
Slide Set 8

Doç. Dr. Mehmet Göktürk
Department of Computer Engineering

1

Data Abstractions

- Data Structure Fundamentals
- Implementing Data Structures
- A Short Case Study
- Customized Data Types
- Classes and Objects
- Pointers in Machine Language

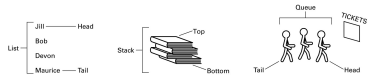
2

Basic Data Structures

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

3

Lists, stacks, and queues



a. A list of names b. A stack of books c. A queue of people

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

5

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

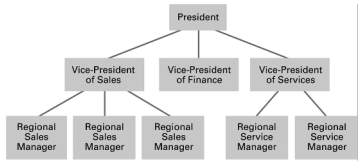
6

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

7

An example of an organization chart



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

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

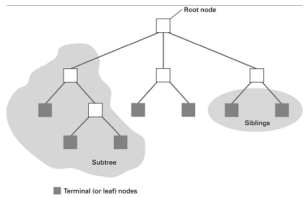
10

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

11

Figure 8.3 Tree terminology



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

13

Novels arranged by title but linked according to authorship

14

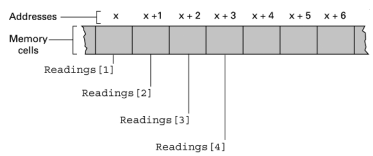
15

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

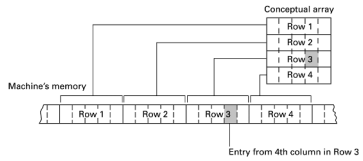
16

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



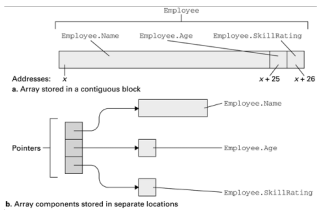
17

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



18

Storing the heterogeneous array Employee



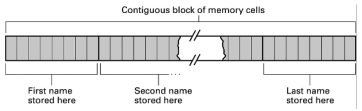
19

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

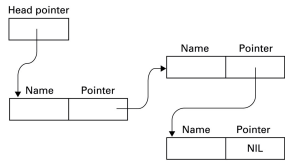
20

Names stored in memory as a contiguous list



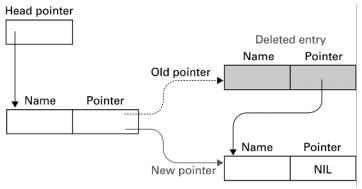
21

The structure of a linked list



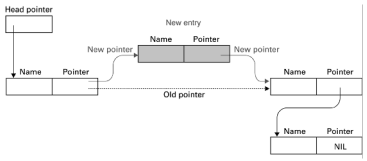
22

Deleting an entry from a linked list



23

Inserting an entry into a linked list



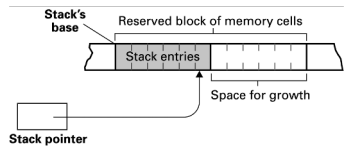
24

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

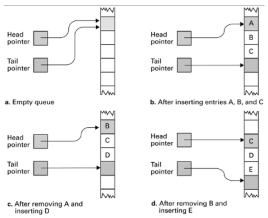
25

A stack in memory



26

A queue implementation with head and tail pointers



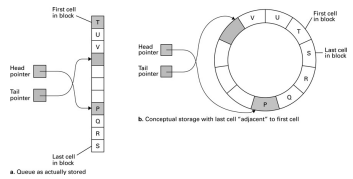
27

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]

28

A circular queue containing the letters P through V



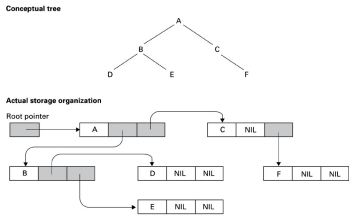
29

The structure of a node in a binary tree

Cells containing the data	Left child pointer	Right child pointer

30

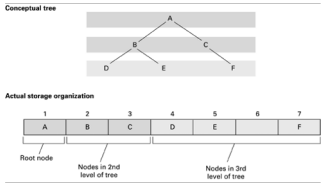
The conceptual and actual organization of a binary tree using a linked storage system



31

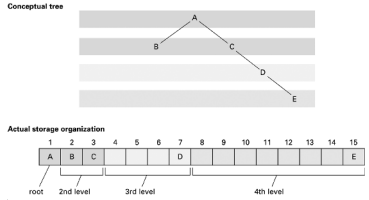
32

A tree stored without pointers



33

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



34

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.

35

Linked List

36

Linked List

37

Stack

38

Stack

39

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.)

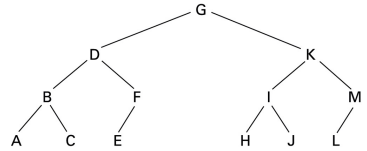
40

Case Study

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

41

The letters A through M arranged in an ordered tree



42

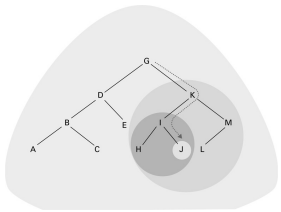
The binary search as it would appear if the list were implemented as a linked binary tree

```
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
```

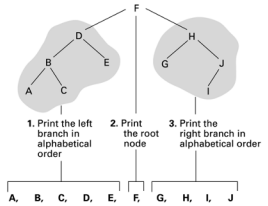

43

The successively smaller trees considered by the procedure when searching for the letter J



44

Printing a search tree in alphabetical order



45

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)

46

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

47

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
```


48

User-defined Data Type

- A template for a heterogeneous structure
- Example:

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


49

User-defined Data Type - Struct

50

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 . . .
}
```


51

Class

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

52

Abstract Data Type - Class

53

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;
 }
}
```

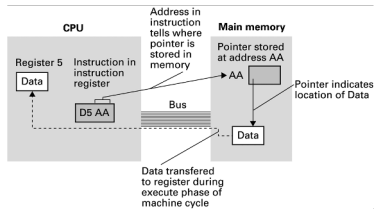

54

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

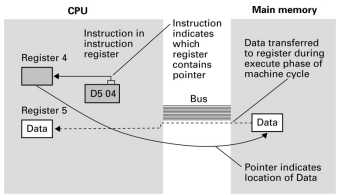
55

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



56

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

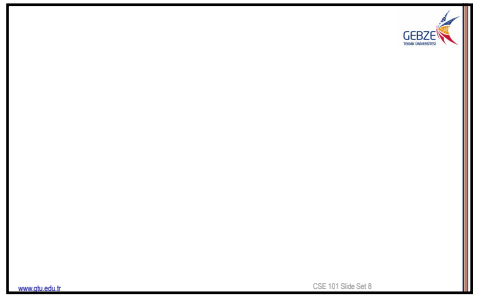


57

58

59

60



61