COM1013
INTRODUCTION TO COMPUTER SCIENCE

Lecturer: Begüm MUTLU BİLGE, PhD

begummutlubilge+com1013@gmail.com (recommended)
bmbilge@ankara.edu.tr

Data Abstractions

The data structures supported by a programming language are known as **primitive** structures.

We will today explore techniques by which data structures other than a language’s primitive structures

Basic Data Structures

Arrays

Aggregates

Lists

Stacks

Queues

Trees

Implementing Data Structures

Goal:

To understand how programs that deal with such structures are translated into machine-language programs

that manipulate data stored in main memory

Arrays & Aggregates

Array is a “rectangular” block of data whose entries are of the [redacted].

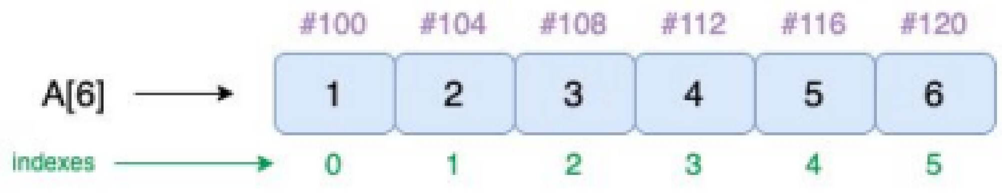
- The simplest form of array is the **one-dimensional array**, a single row of elements with each position identified by an index.
- A **two-dimensional array** consists of multiple rows

and columns in which positions are identified by pairs of indices

- the first index identifies the row associated with the position,
- the second index identifies the column.

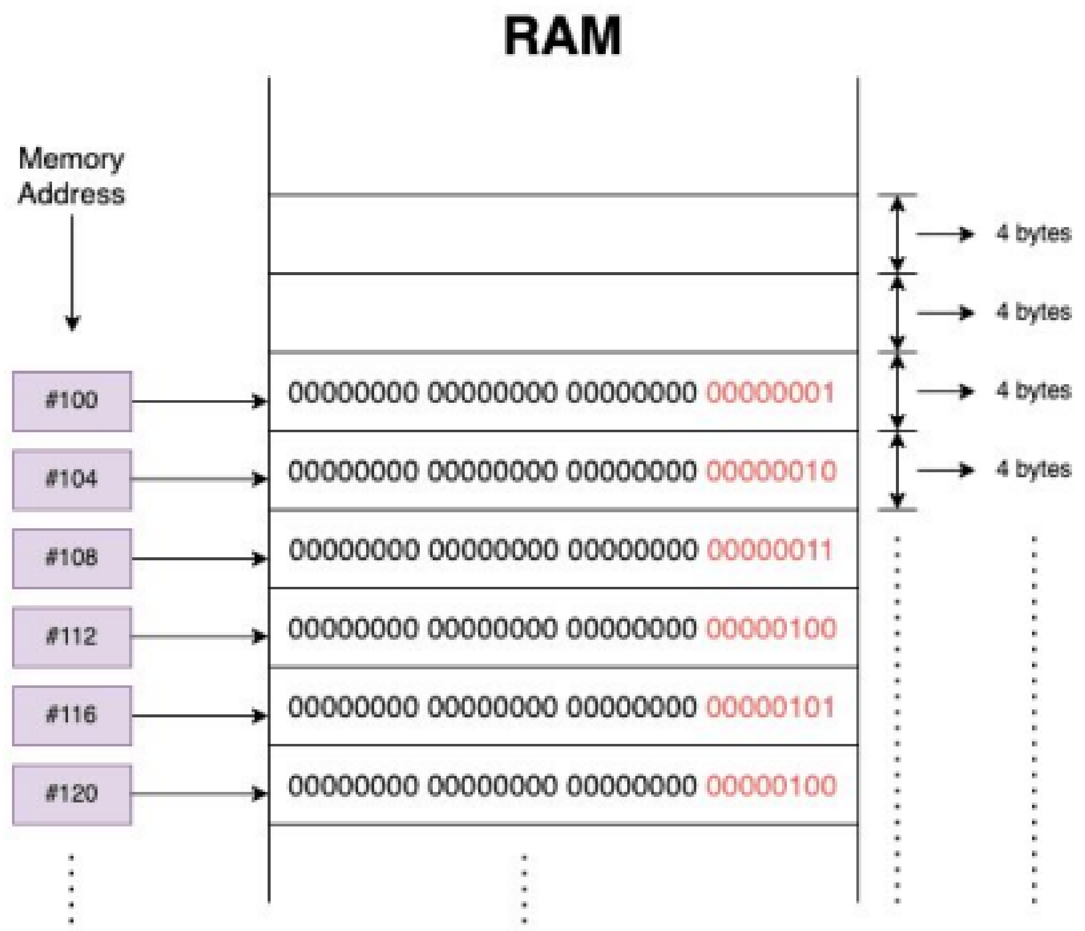
Arrays & Aggregates

Storing 1D Array



Integer in Java takes 4 bytes

Array above occupies the contiguous memory fr
#100 to #120



Arrays & Aggregates

Storing 2D Array (2D row-major)

Given a row index i_{row} and a column index i_{col}, the offset of the element they denote in the linear representation

is: offset = $i_{row} * n_{COLS} + i_{col}$

n_{COLS}: the number of columns per row in the matrix.

Arrays & Aggregates

Storing 2D Array (2D column-major)

Given a row index i_{row} and a column index i_{col}, the offset of the element they denote in the linear representation

is: offset = $i_{row} + i_{col} * n_{ROWS}$

n_{ROWS}: the number of rows per column in the matrix.

Arrays & Aggregates

Storing Multi dimensional Arrays [Beyond 2D]

In row-major layout of multi-dimensional arrays, the last index is the fastest changing.

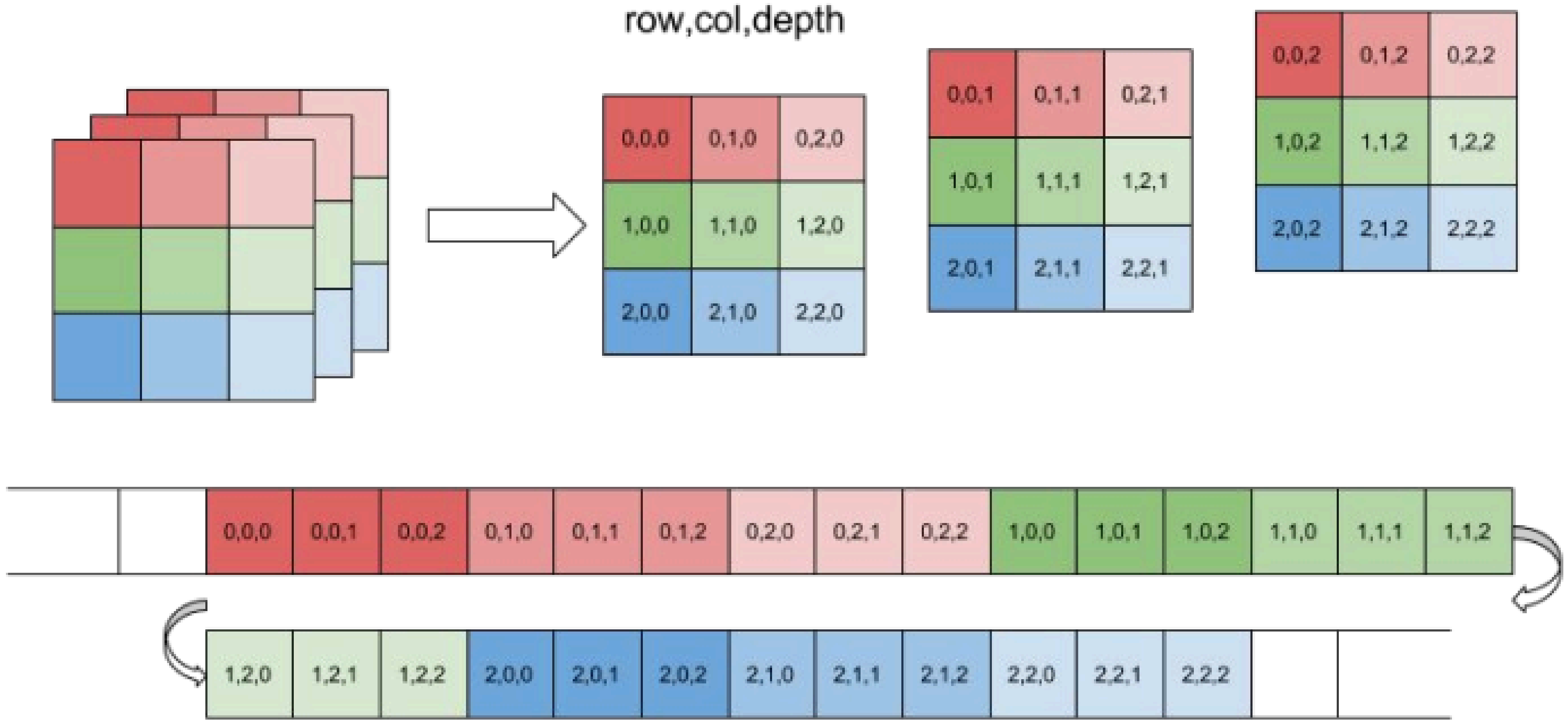
$$offset = n_d + N_d \cdot (n_{d-1} + N_{d-1} \cdot (n_{d-2} + N_{d-2} \cdot (\cdots + N_2 n_1) \cdots))) = \sum_{i=1}^{d} \left(\prod_{j=i+1}^{d} N_j \right) n_i$$

In column-major layout of multi-dimensional arrays, the first index is the fastest changing.

Arrays & Aggregates

Storing 3D Array: rows, columns and depth.

The memory layout of a 3D array with N_1=N_2=N_3=3, in row-major: $offset = n_3 + N_3 * (n_2 + N_2 * n_1)$



Arrays & Aggregates

An **aggregate** type is a block of data items that might be of different types and sizes

Example: The block of data relating to a single employee,

- the fields of which might be the employee’s name (an array of type character), age (of type integer), and skill rating (of type float).

Fields in an aggregate type are usually accessed by field name, rather than by a numerical index number.

Arrays & Aggregates

Storing Aggregates

Lists, Stacks, and Queues

Lists: a collection whose entries are arranged sequentially

- The beginning of a list is called the head of the list.

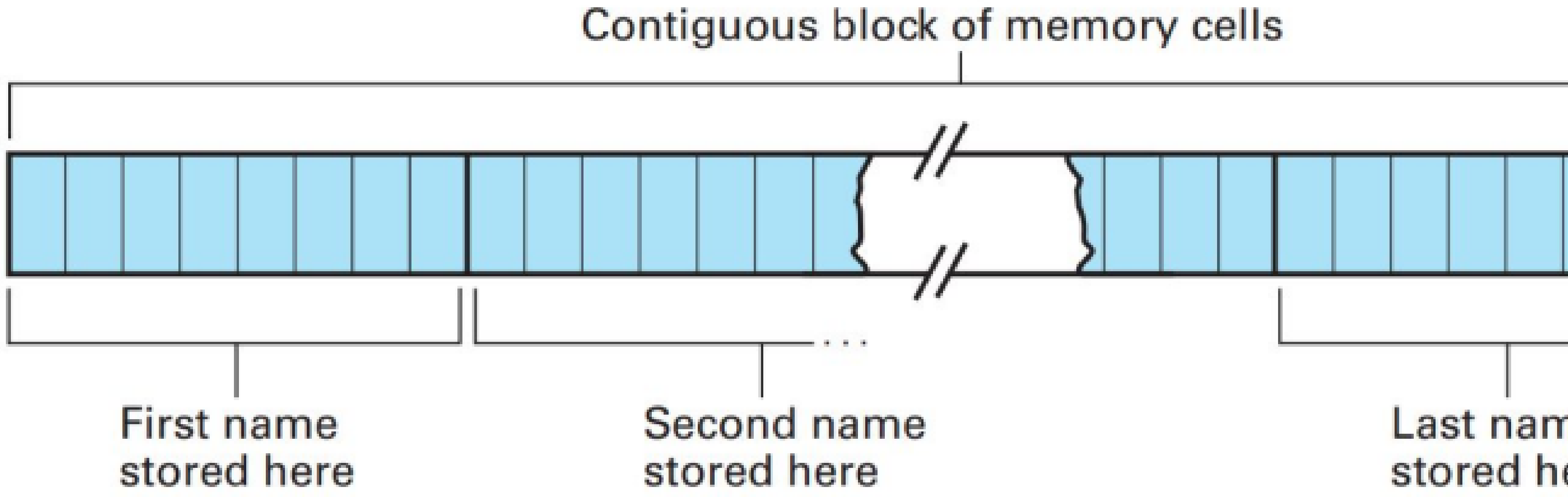
- The other end of a list is called the tail

Almost any collection of data can be envisioned as a list.

- Text can be envisioned as a list of symbols, ● A two-dimensional array can be envisioned as a list of rows, and
- Music recorded on a CD can be envisioned as a list of sounds.

Lists, Stacks, and Queues

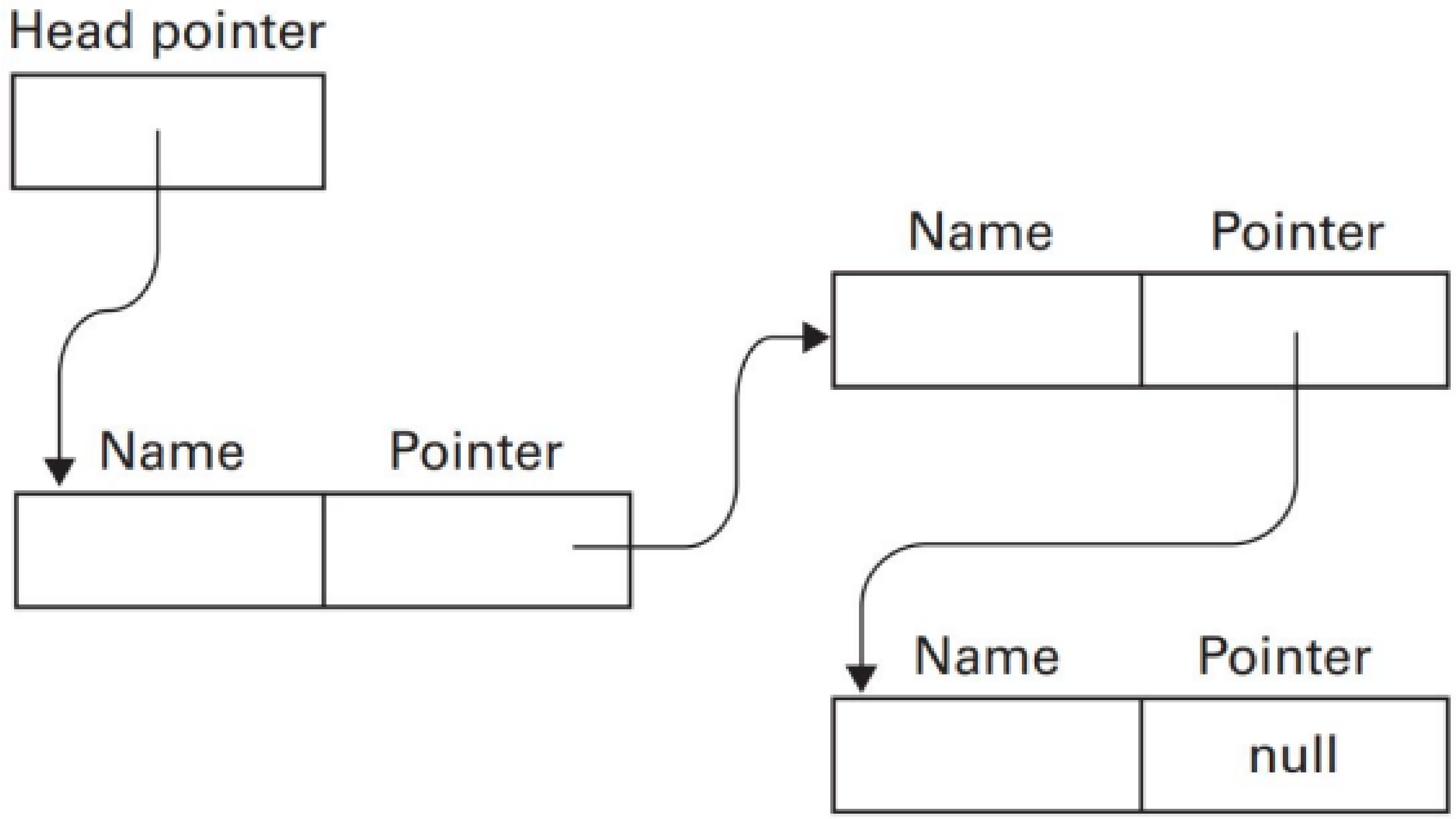
Storing Lists: CONTIGUOUS LIST



Names stored in memory as a contiguous list

Lists, Stacks, and Queues

Storing Lists: LINK LIST

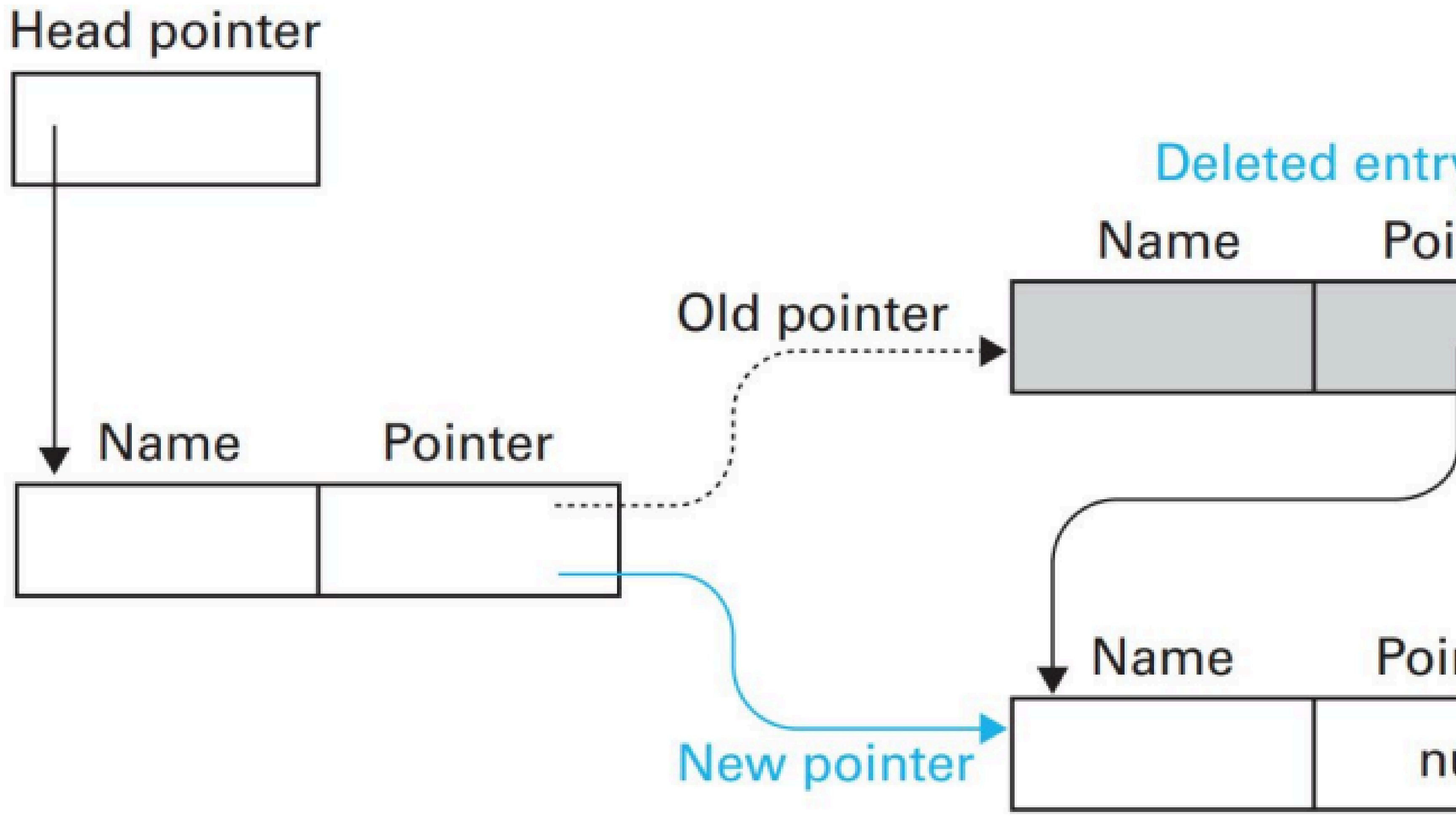


The structure of a linked list

Lists, Stacks, and Queues

To appreciate the advantages of a linked list over a contiguous one, consider the task of

deleting an entry



Lists, Stacks, and Queues

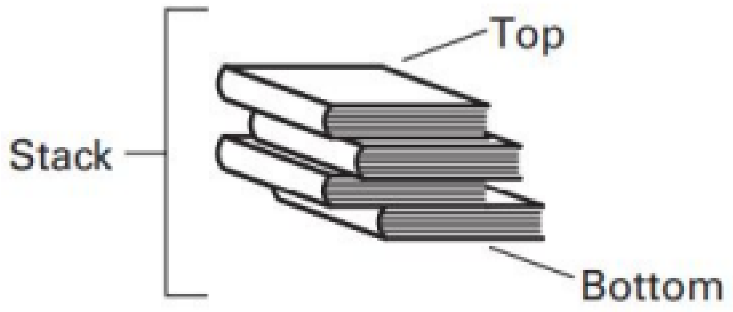
To appreciate the advantages of a linked list over a contiguous one, consider the task of **inserting** an entry

Lists, Stacks, and Queues

Stack: A list in which entries are inserted and removed only at the head.

Last in First Out (LIFO)

An example is a stack of books where physical restrictions dictate that all additions and deletions occur at the top



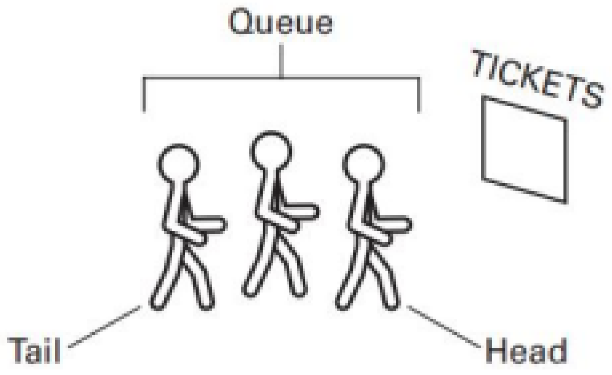
Lists, Stacks, and Queues

Queues: A list in which the entries are **removed** only at the **head** and new entries are **inserted** only at the **tail**.

First in First Out (FIFO)

An example is a line, or queue, of people waiting to buy tickets at a theater

● the person at the head of the queue is served while new arrivals step to the tail of the queue.



Lists, Stacks, and Queues

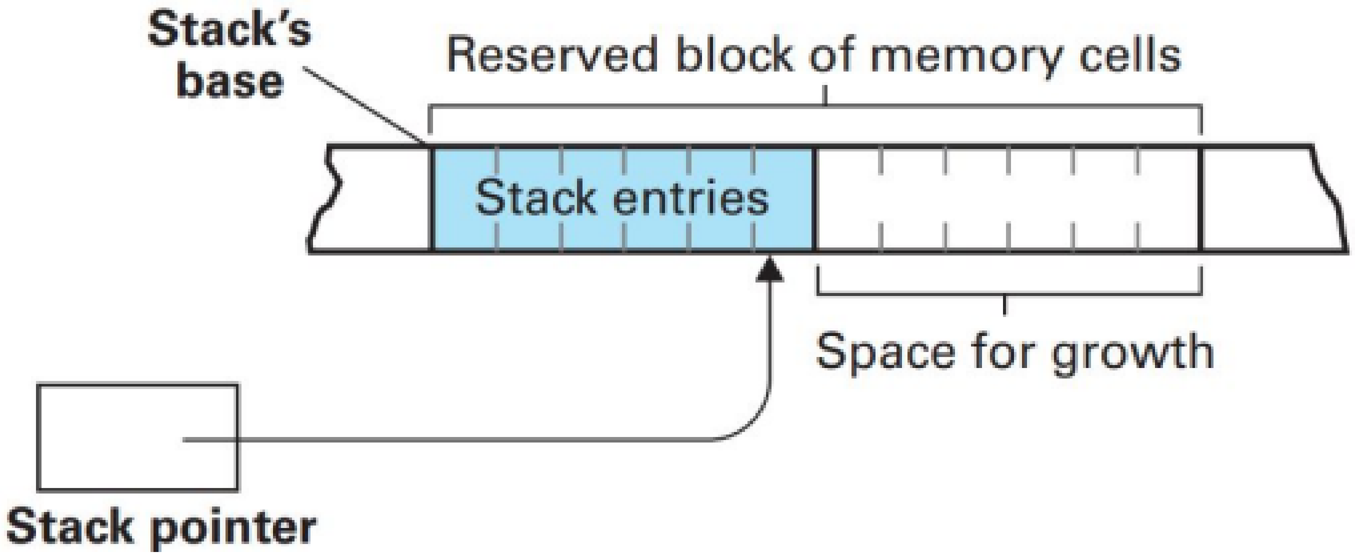
Storing Stack and Queue

- An organization similar to a contiguous list is often used
- In the case of a **stack**,
  - a block of memory, large enough to accommodate the stack at its maximum size, is reserved. (Critical design decision)
  - One end of this block is designated as the stack's base.
  - It is here that the first entry to be pushed onto the stack is stored.
  - Then each additional entry is placed next to its predecessor as the stack grows toward the other end of the reserved block.

Lists, Stacks, and Queues

Storing Stack

● As entries are pushed and popped, the location of the top of the stack will move back and forth ● To keep track of this location, its address is stored in an additional memory cell, **stack pointer**. ○ a pointer to the top of the stack.



Lists, Stacks, and Queues

Storing Stack

[Example]

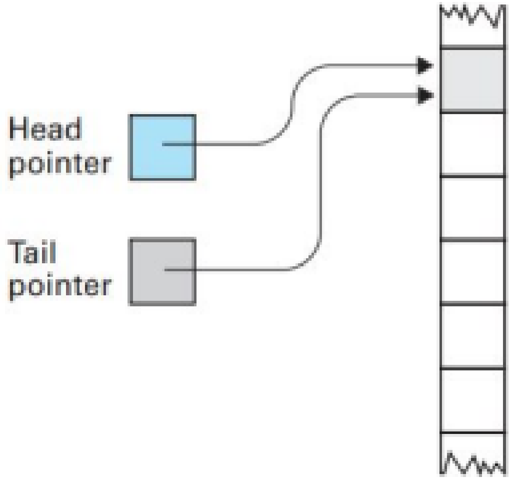
Lists, Stacks, and Queues

Storing Queue

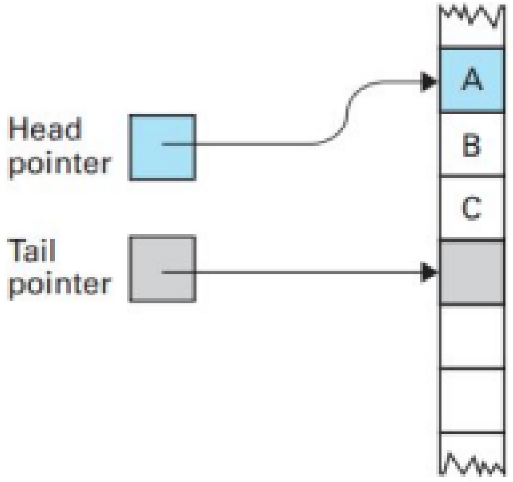
- Reserve a block of contiguous cells in main memory large enough to hold the queue at its projected maximum size.
- in the case of a queue we need to perform operations at both ends of the structure
  - so we set aside two memory cells to use as pointers ▪ head pointer, keeps track of the head of the queue ▪ tail pointer, keeps track of the tail

Lists, Stacks, and Queues

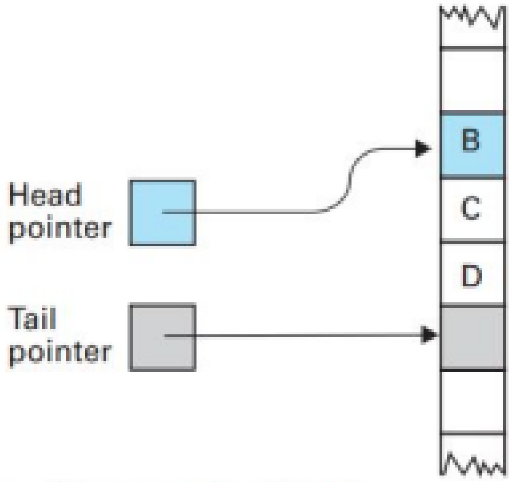
Storing Queue



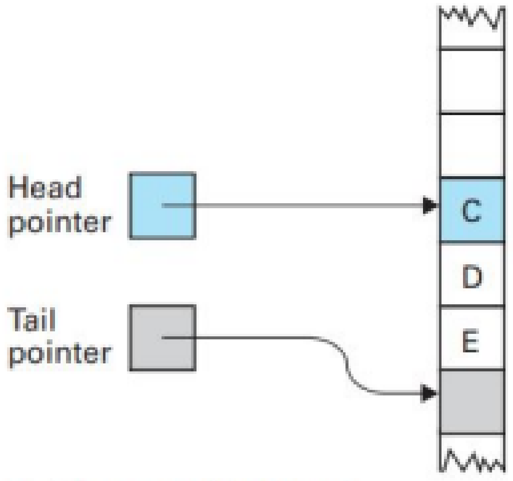
a. Empty queue



b. After inserting entries A, B, and C

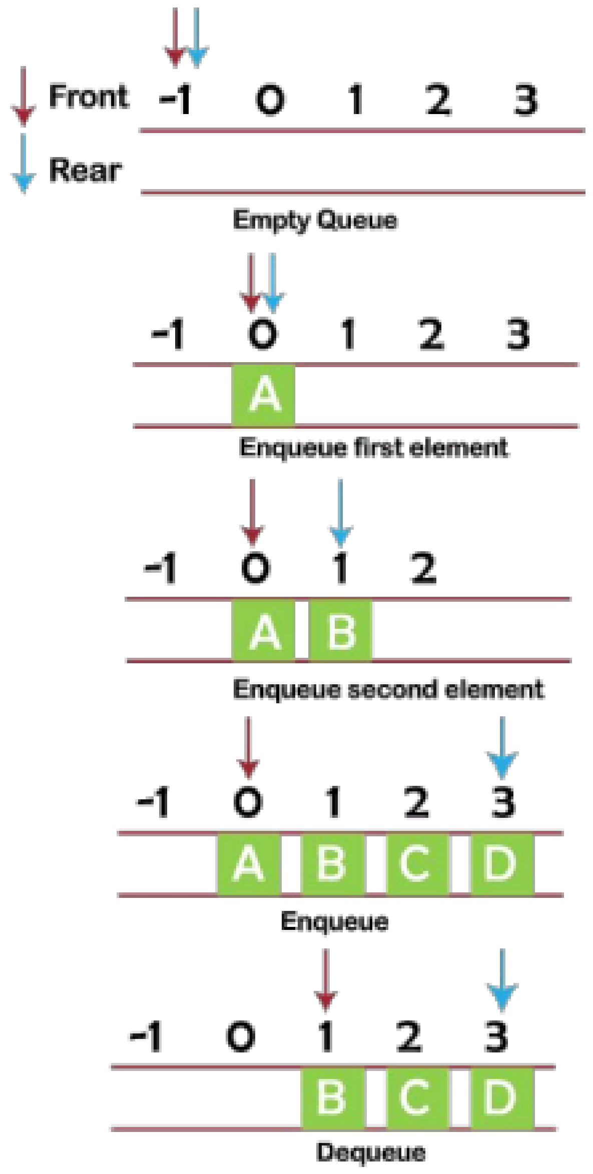


c. After removing A and inserting D



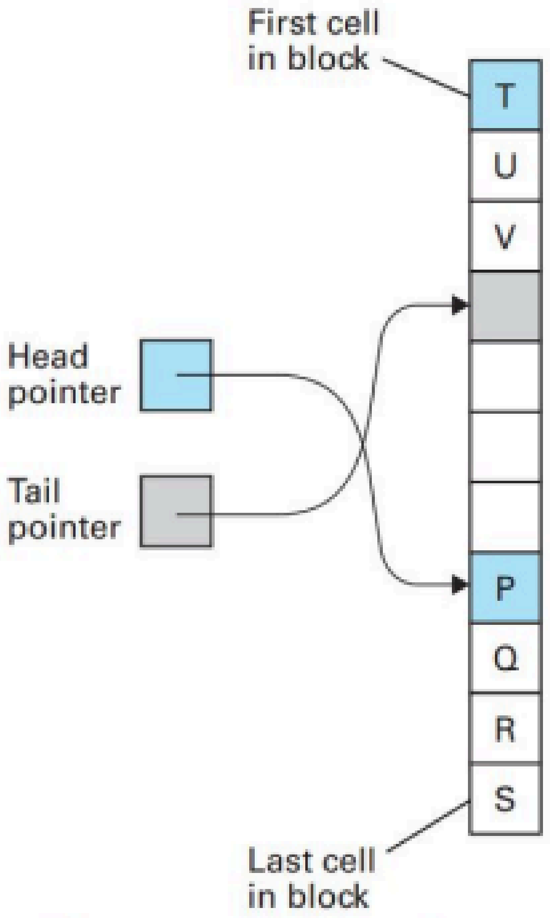
d. After removing B and inserting E

Lists, Stacks, and Queues Queue

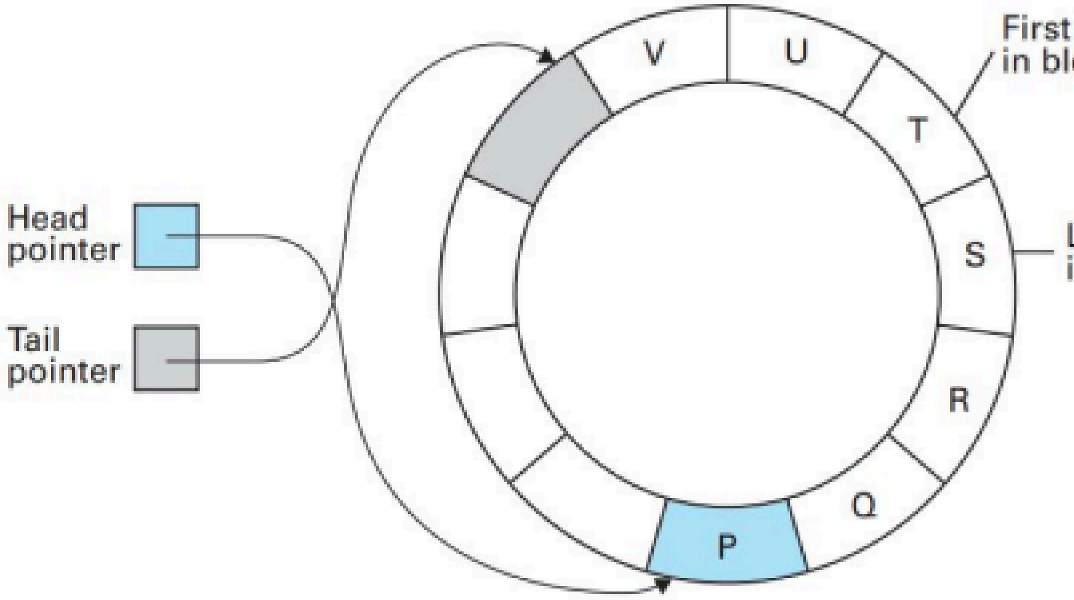


Lists, Stacks, and Queues

Storing Queue: circular queue



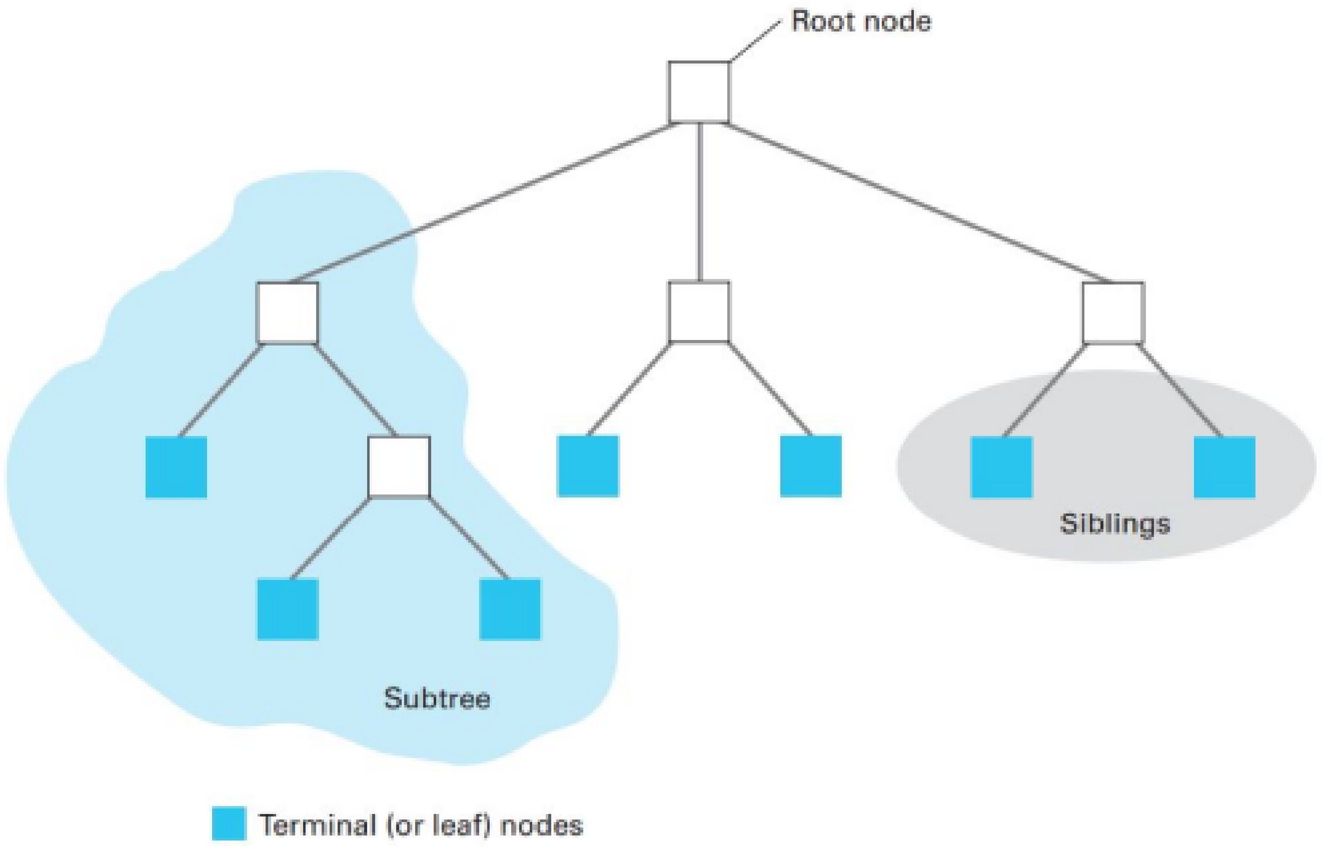
a. Queue as actually stored



b. Conceptual storage with last cell "adjacent" to first cell

Trees

A tree is a collection whose entries have a hierarchical organization



Storing Binary Trees

Trees in which each node has at most two children

Stored in memory using a linked structure similar to that of linked lists.

Each entry (or node) of the binary tree contains:

1. the data,
2. a pointer to the node’s first child, and
3. a pointer to the node’s second child.

Storing Binary Trees

Storing Binary Trees

Another approach: Using contiguous block of memory

Storing Binary Trees

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

Abstraction Again

The structures are often associated with **data**.

However, a computer's main memory is not organized as arrays, lists, stacks, queues, and trees but

- **is instead organized as a sequence of addressable memory cells.**

Thus, all other structures must be simulated.

[redacted] is the subject of today's lecture.

Static Versus Dynamic Structures

An important distinction in constructing abstract data structures is whether the structure being simulated is static or dynamic,

- whether the shape or size of the structure changes over time.

A general rule: static structures are more easily managed than dynamic ones.

Pointers

In the case of data structures, pointers are used to record the location where data items are stored.

Recap: A register called a program counter

- is used to hold the address of the next instruction to be executed.
- Thus, the program counter plays the role of a

pointer.
    - In fact, another name for a program counter is **instruction pointer**.

Pointers

Novels arranged by title but linked according to authorship

Manipulating Data Structures

A Short Case Study

Let us consider the task of storing a list of names in alphabetical order. We assume that the operations to be performed on this list are the following:

- search for the presence of an entry,
- print the list in alphabetical order, and
- insert a new entry

Our goal is to develop a storage system along with a collection of functions to perform these operations—thus producing a complete abstract tool.

A Short Case Study

We begin by considering options for storing the list.

- List in a sequential fashion ??

- Binary search tree ??

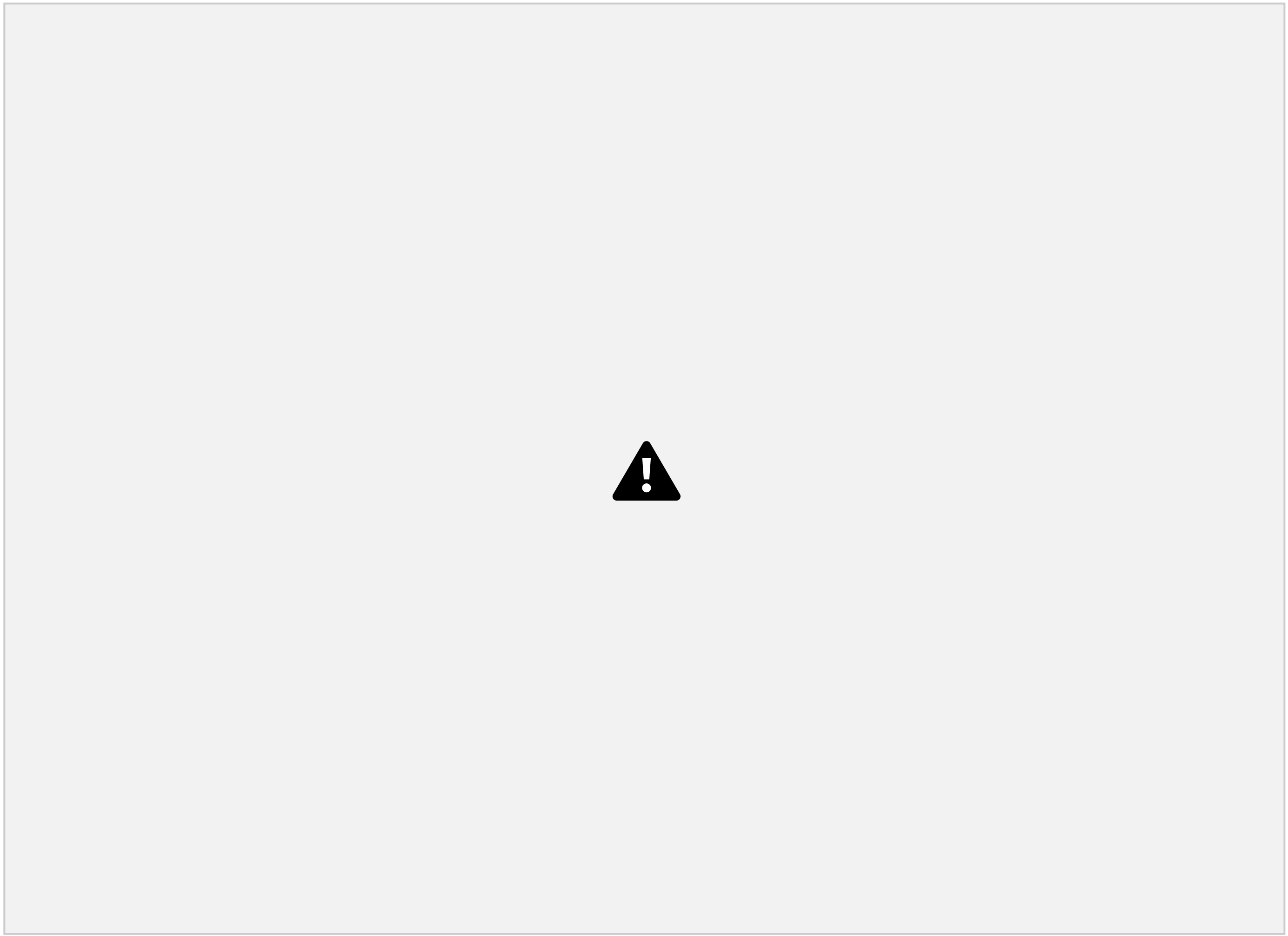
A Short Case Study

The letters A through M arranged in an ordered tree

A Short Case Study: Search

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

A Short Case Study: Search


The successively smaller trees considered by the function in Figure 8.21 when searching for the letter J

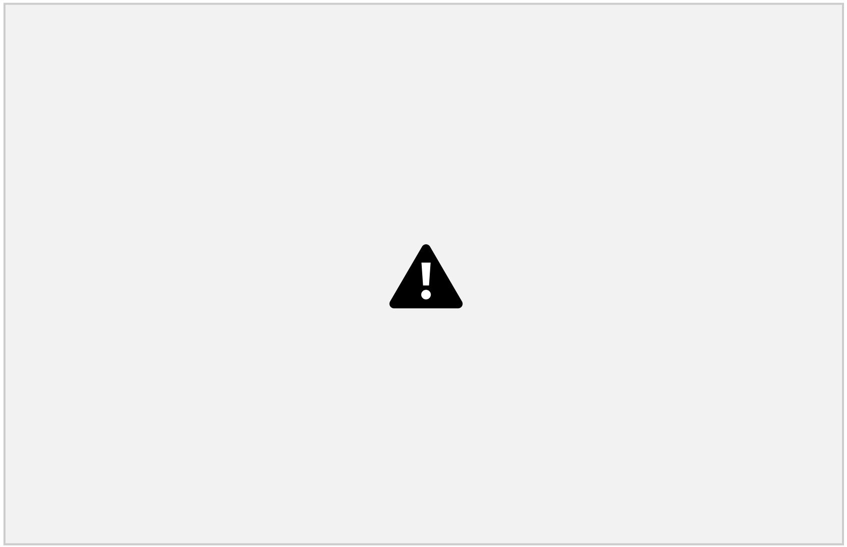
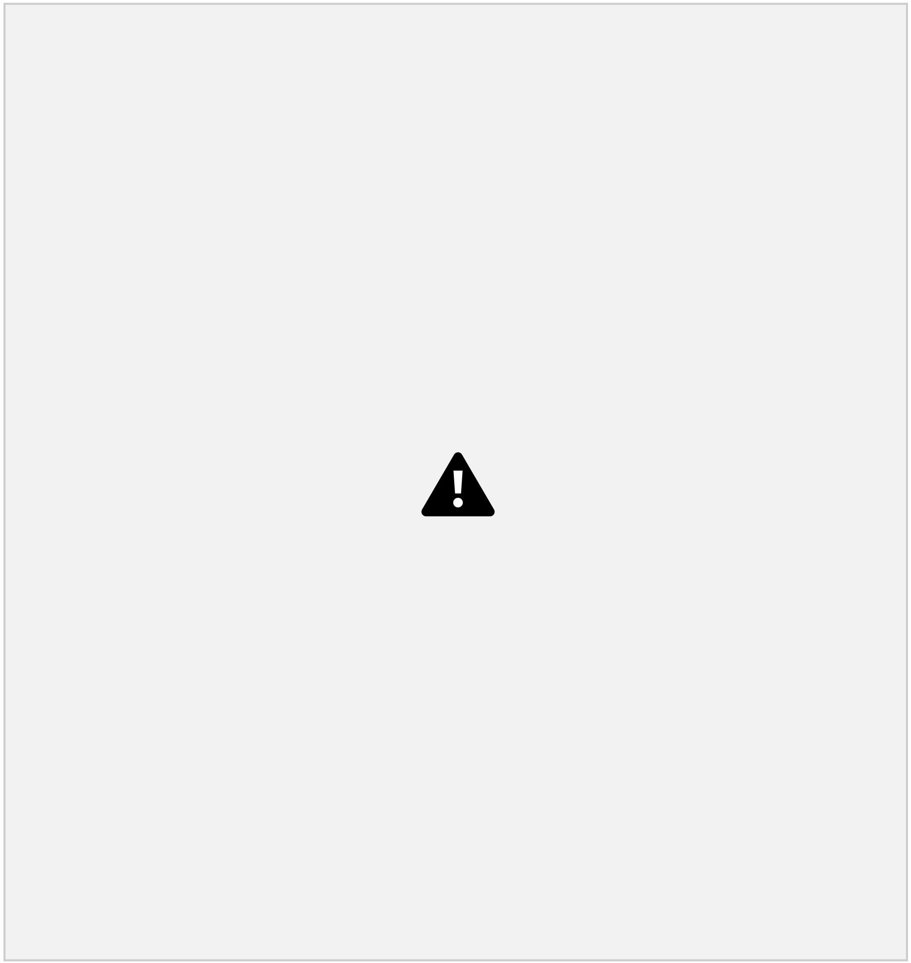
A Short Case Study: Print

Printing a search tree in alphabetical order

A Short Case Study: Print

A function for printing the data in a binary tree

A Short Case Study: Insert

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

A Short Case Study: Conclusion

- A software package consisting of a linked binary tree structure together with our functions for searching, printing, and inserting provides a complete package
  - that could be used as an abstract tool by our hypothetical application.
  - Indeed, when properly implemented, this package could be used without concern for the actual underlying storage structure.
- By using the procedures in the package, the user could envision a list of names stored in alphabetical order,
  - whereas the reality would be that the “list” entries are actually scattered among blocks of memory cells that are linked as a binary tree.

Customized Data Types

We introduced the concept of a data type and discussed such elementary types as integer, float, character, and Boolean.

- These data types are provided in most programming languages as **primitive** data types.

Now we consider ways in which **a programmer can define** her/his own data types to fit more closely the needs of a particular application

- User-Defined Data Types
- Abstract Data Types

User-Defined Data Types

LEFT: defines a new aggregate, called Employee, containing fields called Name (of type character), Age (of type integer), and SkillRating (of type float).

User-Defined Data Types

RIGHT: does not define a new aggregate variable, but defines a new aggregate type, EmployeeType.

Abstract Data Types

- Traditional user defined data types, merely allow programmers to define new storage systems.
  - They do not also provide operations to be performed on data with these structures.
- An abstract data type (ADT) is a user-defined data type that can include both data (representation) and functions (behavior)

Abstract Data Types

Programming languages that support creation of ADTs generally provide two features:

● syntax for defining the ADT as single unit, and ● a mechanism for hiding the internal structure of the ADT from other parts of the program that will make use of it.

Abstract Data Types

would reference a unique block of memory cells used to implement an individual stack.

But what if we now want to push the value 25 onto StackOne?

Abstract Data Types

What is needed is a mechanism for defining the operations that are allowed on our StackType, as well as for protecting the internal variables from outside interference.

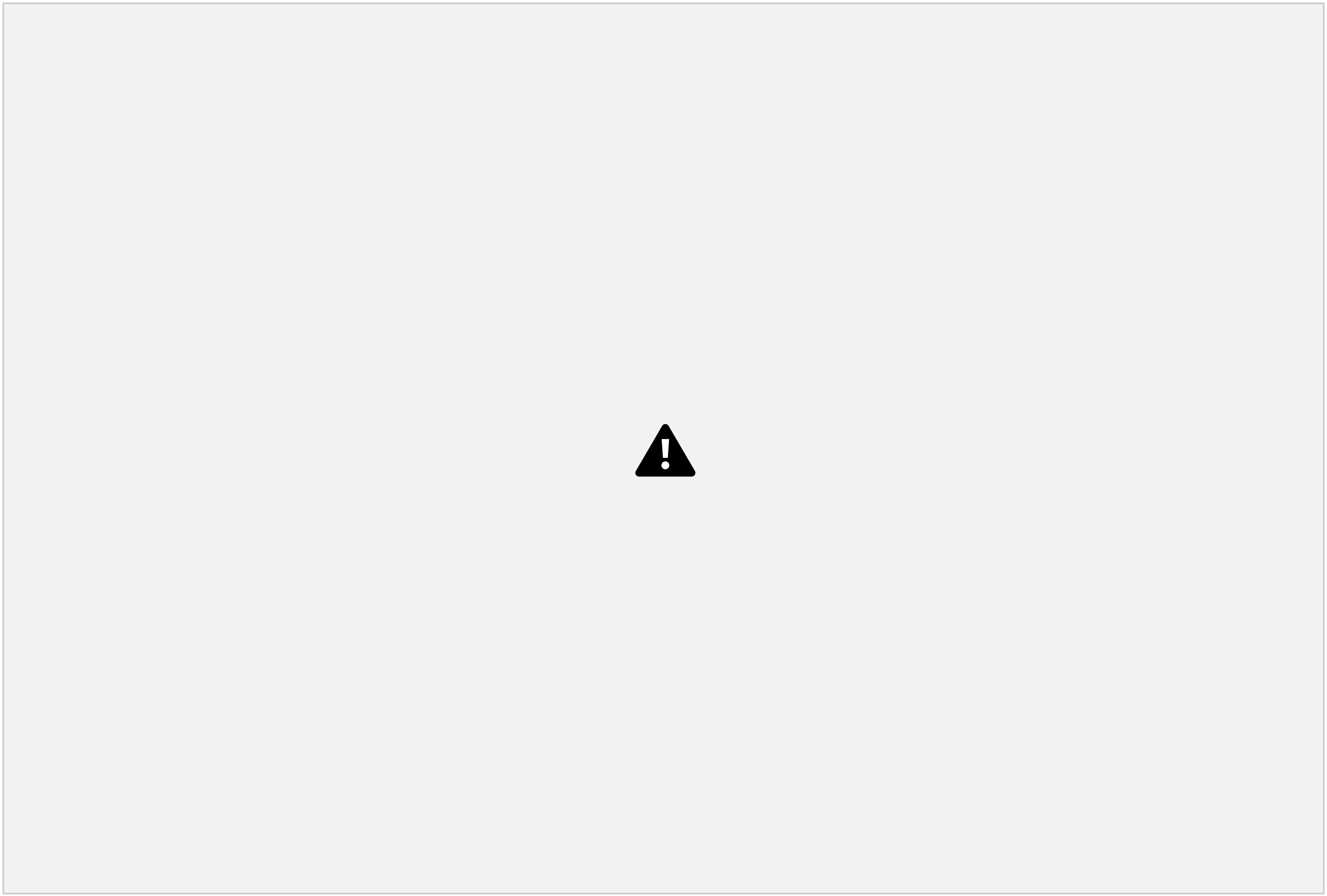
Classes and Objects

- The object-oriented paradigm leads to systems composed of units called objects that interact with each other to accomplish tasks.
- Each object is an entity that responds to messages

received from other objects.

- Objects are described by templates known as classes
- In many respects, these classes are actually descriptions of abstract data types

Classes and Objects



A stack of integers implemented in Java and C#

Classes and Objects Abstract Data Type Objects

Abstract data type is abstraction that define set of values and set of

operations on these values. It is a self-contained component which consists of methods and	properties to make certain type of data useful.
User-defined data type.	It is an instance of class.
ADT is made of with primitive datatypes.	An object is an abstract data type with the addition of polymorphism and inheritance.
It is a type (or class) for objects whose behaviour is defined by a set of value and a set of operations.	It is a basic unit of Object-Oriented Programming.
Is not necessarily an OOP concept.	Objects is an OOP concept.
Common examples include lists, stacks, sets, etc.	Objects have states and behaviours Test t = new Test();.
Allocate memory when data is stored.	When we instantiate an object then memory is allocated.

Extra Reading

Brookshear JG, Smith D, Brylow D.

"Computer science: an overview".

Read the following section in the textbook.

Chapter: 8.7. Pointers in Machine Language

Summarize the section in 1 page (no figure should be included.)

Template for summarization assignments:

https://docs.google.com/document/d/1i1tU3AdIvPgRxqzGmXrsnVVCRXws5V3caDTCPMbPJ_0/edit?usp=sharing

Research Themes

1. Hash-based data structures

2. Graphs

3. Self balancing binary trees 4.

Probabilistic data structures

COM1013
INTRODUCTION TO COMPUTER SCIENCE

Lecturer: Begüm MUTLU BİLGE, PhD

begummutlubilge+com1013@gmail.com (recommended)
bmbilge@ankara.edu.tr