

# Database Management Systems (DBMS)

# Data and Information

**DATA:** *Facts concerning people, objects, vents or other entities.  
Databases store data.*

**INFORMATION:** *Data presented in a form suitable for  
interpretation.*

Data is converted into information by programs and queries.  
Data may be stored in files or in databases. Neither one stores  
information.

**KNOWLEDGE:** *Insights into appropriate actions based on  
interpreted data.*

# Definitions

- ***Data***: stored representations of meaningful objects and events or
- Referred to facts concerning objects and events that could be recorded and stored on computer media
  - Structured: numbers, text, dates
  - Unstructured: images, video, documents
- ***Information***: data processed to increase knowledge in the person using the data
- ***Metadata***: data that describes the properties and context of user data

# What is a Database

- ❑ Shared collection of logically related data (and a description of this data), designed to meet the information needs of an organization.
- ❑ System catalog (metadata) provides description of data to enable program–data independence.
- ❑ Logically related data comprises entities, attributes, and relationships of an organization’s information.

# VERİTABANI NEDİR?

- Birbiri ile ilişkili veriler topluluğudur.
- Ya da, daha detaylı bir tanımla; veriyi yönetmek ve sunmak için kullanılan tablolar, formlar, veri erişim sayfaları, sorgular ve raporlardan oluşan nesnelere topluluğudur.
- Veritabanı sadece veriler yığını değil, bunlar arasındaki ilişkiyi de inceler.

# Basic Principles

**DATABASE:** *A shared collection of interrelated data designed to meet the varied information needs of an organization.*

**DATABASE MANAGEMENT SYSTEM:** *A collection of programs to create and maintain a database.*

Define

Construct

Manipulate

# Database Management System Facility

- Data definition language (DDL)
- Data manipulation language (DML)
- Structured query language (SQL)
- Security system
- Integrity system
- Concurrency control system
- Backup & recovery system
- View mechanism

# DBMS Environment

- Hardware
  - Client-server architecture
- Software
  - dbms, os, network, application
- Data
  - Schema, subschema, table, attribute
- People
  - Data administrator & database administrator
  - Database designer: logical & physical
  - Application programmer
  - End-user: naive & sophisticated
- Procedure
  - Start, stop, log on, log off, back up, recovery



# VERİTABANININ FAYDALARI

- Veri tekrarları ortadan kaldırılır ya da en aza indirilir.
- Bellek alanı israfı önlenir.
- Standart bir sorgu dili kullanmak mümkündür.
- Veri bütünlüğünün bozulması önlenir.

# Advantages of Database Processing

- More information from same data
- Shared data
- Balancing conflicts among users
- Controlled redundancy
- Consistency
- Integrity
- Security
- Increased productivity
- Data independence

# Advantages of DBMS

- Control redundancy
- Consistency
- Integrity
- Security
- Concurrency control
- Backup & recovery
- Data standard
- More information
- Data sharing & conflict control
- Productivity & accessibility
- Economy of scale
- Maintenance

# VERİTABANININ RİSKLERİ

- kurulum ve bakımı klasik dosya sisteminden pahalıdır.
- sistem içinde bazı bileşenler iyi tasarlanmazsa sistem bir bütün olarak başarısızlığa uprayabilir.

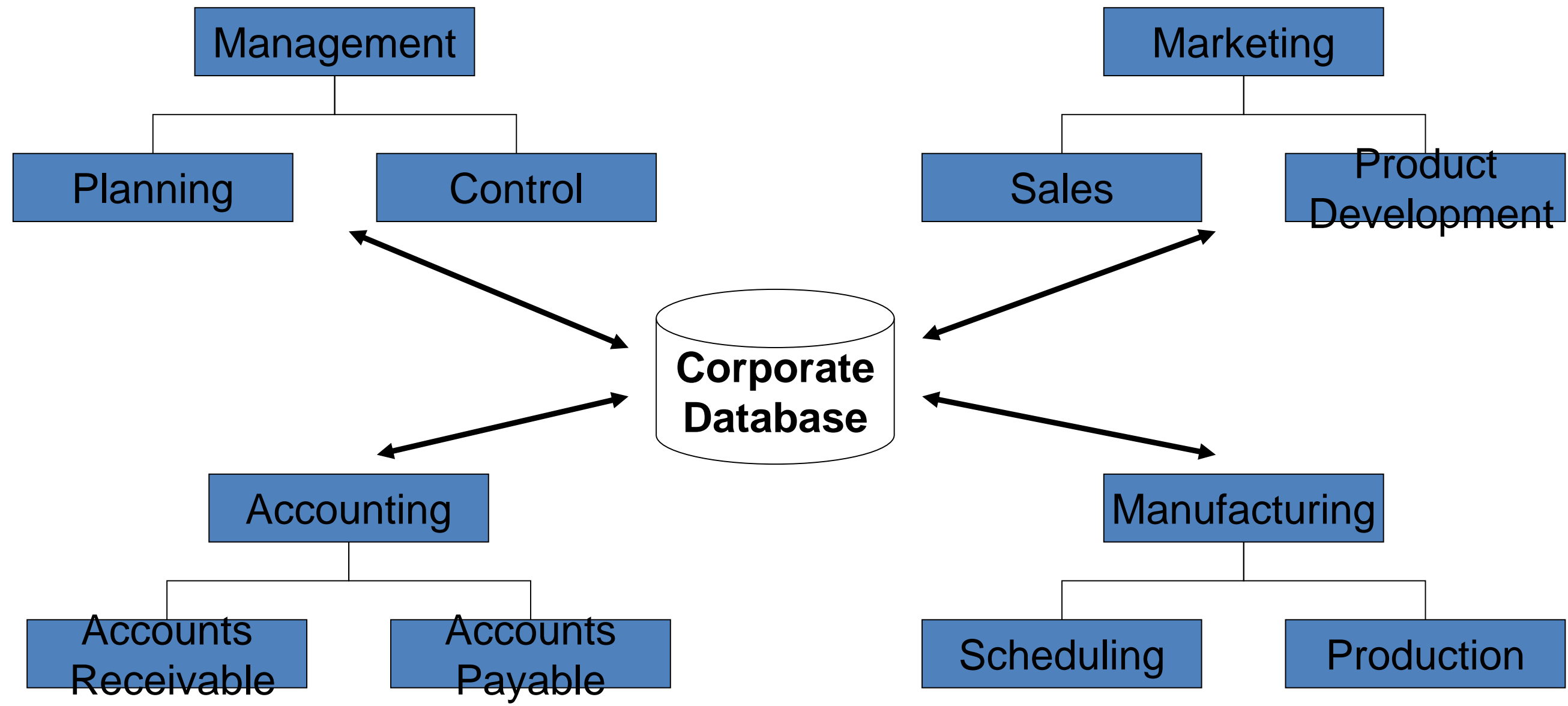
# Disadvantages of Database Processing

- Increased size
- Increased complexity
  - More expensive personnel
- Increased impact of failure
- Difficulty of recovery
- Cost
  - Especially server and mainframe systems

# Limitations of DBMS

- Complexity
- Size
- Cost
  - Software
  - Hardware
  - Conversion
- Performance
- Vulnerability

# The concept of a shared organizational database



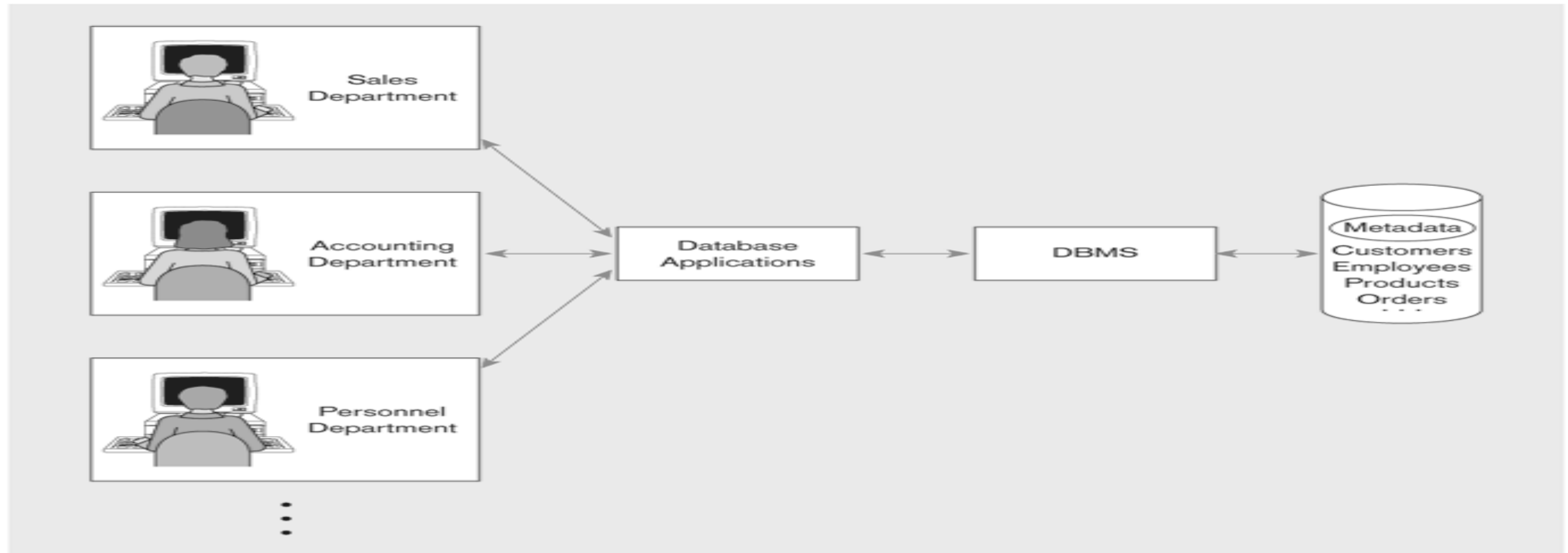
# Database Management System

- ❑ A software system that is used to create, maintain, and provide controlled access to users of a database
- ❑ (Database) application program: A computer program that interacts with database by issuing an appropriate request (SQL statement) to the DBMS



# Database Management System

**Figure 1-3** Database approach at Pine Valley Furniture Company



*DBMS manages data resources like an operating system manages hardware resources*

# VERİTABANI YÖNETİM SİSTEMİ NEDİR? (DATABASE MANAGEMENT SYSTEM)

- Bir veri tabanı oluşturup üzerinde çeşitli işlemler yapılmasını sağlayan programlar topluluğudur.

# İLİŞKİSEL VERİ TABANI NEDİR?

- Veritabanı uygulamaları iki temel türe ayrılabilir.
  - Düz-dosya veritabanları oluşturan lar
  - İlişkisel (relational) veritabanları oluşturanlar.

# DÜZ-DOSYA VERİTABANI NEDİR?

- Bu tür programlarda tüm veritabanı tek bir tabloya sığdırılmalıdır.
- Bu, birkaç kayıta ortak olan herhangi bir bilginin her kayıta tekrarlanacağı anlamına gelir.
- Word ve excel bu tür programlara örnektir.

# İLİŞKİSEL VERİTABANI NEDİR?

- Bu tür veri tabanında bir çok farklı tablo kullanılır ve tablolar arasında ilişkiler oluşturulur.
- Bir ilişki, bir tabloya, başka bir başka bir tablodaki kaydı bağlanmamızı sağlar.
- Bu şekilde veriler daha az yer kaplar ve güncelleme kolaylaşır.

# VERİTABANI BİLEŞENLERİ

- Tablolar
- Formlar
- Veri erişim sayfaları
- Sorgular
- Raporlar

# BYTE

- 0-255 arası pozitif tamsayıları saklar.
- Bellekte 1 byte yer kaplar.

# INTEGER (TAMSAYI)

- 2 byte'lık işaretli tamsayı tipidir.
- -32.768 ile 32.767 arasında bir değer alabilir.



# LONG (UZUN TAMSAYI)

- 4 byte'lık işaretli tamsayı tipidir.
- -2.147.483.648 ile 2.147.483.647 arasında bir değer alabilir.

# SINGLE (TEK)

- 4 byte'lık ondalık sayı tipidir.
- $(+/-)3.402823e38$  ile  $(+/-)1.401298e-45$  arasında değer alabilir.
- Ondalık olarak en fazla 7 hane saklayabilir.

# DOUBLE (ÇİFT)

- 8 byte'lık ondalık sayı tipidir.
- $(+/-)1.79769313486232e308$  ile  $(+/-)4.94065645841247e-324$  arasında değer alabilir.
- Ondalık olarak en fazla 7 hane saklayabilir.

# CURRENCY

- 8 byte'lık ondalık sayı tipidir.
- Ancak sayının ondalık kısmı 4 basamaktan fazla olamaz.
- Bu tip, daha çok para hesapları ve virgülden sonraki hassasiyeti önemsiz olan işlemler için kullanılır.

# DECIMAL (ONDALIK)

- 14 byte'lık veri tipidir.
- Bu tipin en önemli özelliği, sayıdaki bütün basamakların tutulmasıdır.
- Bu veri türü 28 ondalık karakter saklayabilir.

# INPUT MASK (MASKE)

- Verilerin belirli kurallara uymasını sağlayan kısıtlamalardır.
- Örneğin bir alana sadece sayısal değerlerin girilmesi zorlanabilir.

# INPUT MASK (MASKE)

- Maskenin oluşturulmasında bazı özel işaretlerden yararlanır.
- ?:A-z arası alfabetik karakter.
- L: a-z arası alfabetik karakter.\*
- #:0-9 arası rakam veya boşluk. + Ve – kullanılabilir.
- 0:0-9 arası rakam.+ Ve – girilmez.\*
- 9: 0-9 arası rakam veya boşluk.

\*Giriş zorunlu

# SORGU NEDİR?

- Veritabanı sistemi içinde yer alan tablolardaki verilerin isteğe uygun olarak seçilerek , belirli bir düzen içinde sunulmasıdır.



# FORMLAR

- Access veritabanı sisteminde uygulama geliştirirken, yapılması gereken işlemlerden biri de kullanıcı arayüzünün, yani formların oluşturulmasıdır.
- Formlar, program ile kullanıcı arasında bilgi iletişimini, yani etkileşimi sağlayan ortamlardır.

# FORM

- Her form bir veritabanı nesnesidir.
- Formların oluşturulması ve formlara girilen verilerin veritabanına kaydedilmesi için uygun tanımların yapılması gerekir.

# ÖZELLİKLER PENCERESİ

- Araç çubuğunda “properties” butonuna tıklayarak formla ilgili özellikleri belirleyebileceğimiz özellikler penceresi açılır.
- Bu pencereden nesnenin her türlü özelliği değiştirilebilir.

# ÖZELLİKLER PENCERESİ

- Bu pencerede özellikler 5 gruba ayrılmıştır.
- Format: nesnelere biçimlendirilmesi ile ilgili özellikleri barındırır.
- Data: formun ilişkide olduğu verileri ve kaynağını belirler.
- Olay: nesnelere bağlı olayları tanımlar.
- Other: yukarıda sayılanların dışındaki özellikleri barındırır.
- All: tüm özellikler görülür.

# OLAYLARIN KULLANIMI

- Nesnelere üzerinde herhangi bir işlem yapıldığında bir olay gerçekleşir.
- Örneğin bir düğmeye tıklanması bir olaydır.
- Metin kutusunun değiştirilmesi bir olaydır.

# BİR NESNE İÇİN OLAYIN TANIMLANMASI

- nesne seçilir.
- özellikler penceresinde “event” tabına gelinir.
- ilgili olay seçilip “üç noktalı” butona tıklanır ve “choose builder” penceresi açılır.
- buradaki seçeneklerden biri ile olay tamamlanır.

# BUILDER

- Bu pencereden ilgili olaya “ifade”, “makro” veya “kod” yazılabilir.

# BUILDER\EXPRESSION BUILDER

- Buradan bir takım komutların ve fonksiyonların gerçekleşmesi sağlanır.
- Bunlar visual basic komut ve fonksiyonlarıdır.



# BUILDER\MACRO BUILDER

- Bazı işlemleri otomatik hale getirmek için makrolar kullanılır.
- Builder'da macro builder seçilir.
- Macro'ya isim verilir.
- Komutlar/fonksiyonlar ve özellikleri belirlenir.

# Data Types & Data Structures

# Data Types & Data Structures

- Applications/programs read data, store data temporarily, process it and finally output results.
- What is data? Numbers, Voice, Picture, Video Characters, etc.
- Data is classified into data types. e.g. char, float, int, etc.
- A data type is (i) a domain of allowed values and (ii) a set of operations on these values.
- Compiler signals an error if wrong operation is performed on data of a certain type. For example, `char x,y,z; z = x*y` is not allowed.



# Data Types & Data Structures

- Examples

Data Type	Domain	Operations
boolean	0,1	and, or, =, etc.
char	ASCII	=, <>, <, etc.
integer	-maxint to +maxint	+, -, =, ==, <>, <, etc.

# Data Types & Data Structures

- `int i, j;` → `i, j` can take only integer values and only integer operations can be carried out on `i, j`.
- **Built-in** types: defined within the language e.g. `int`, `float`, etc.
- **User-defined** types: defined and implemented by the user e.g. using `typedef` or `class`.

# Data Types & Data Structures

- **Simple Data** types: also known as atomic data types  
→ have no component parts. E.g. int, char, float, etc.

21

3.14

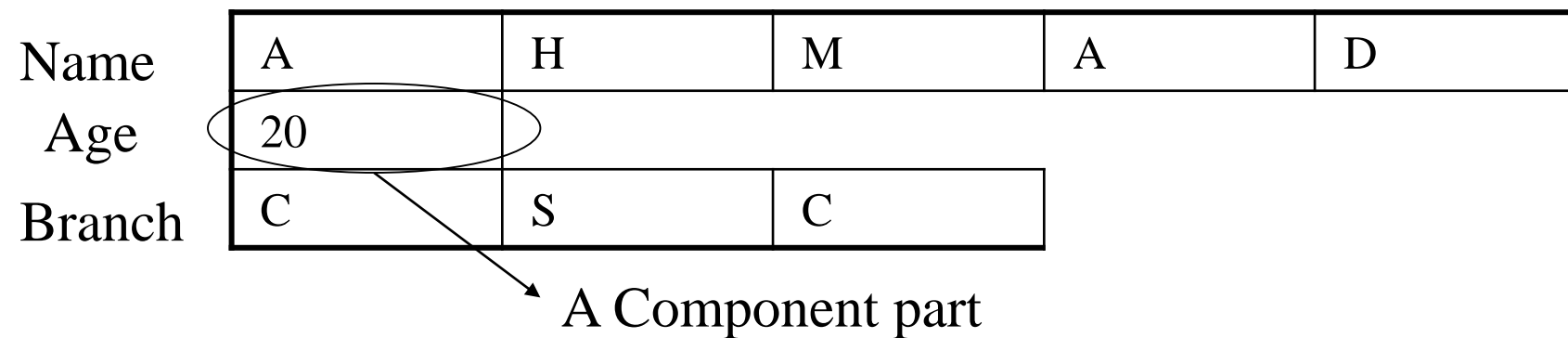
'a'

# Data Types & Data Structures

- **Structured Data** types: can be broken into component parts. E.g. an object, array, set, file, etc. Example: a student object.

Name	A	H	M	A	D
Age	20				
Branch	C	S	C		

A Component part



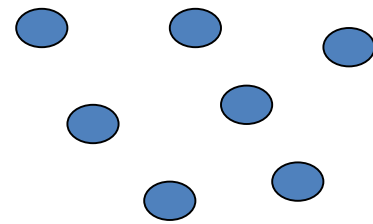
# Data Types & Data Structures

- A **data structure** is a data type whose values (i) can be decomposed into a set of component elements each of which is either simple (atomic) or another data structure (ii) include a structure involving the component parts.



# Data Types & Data Structure

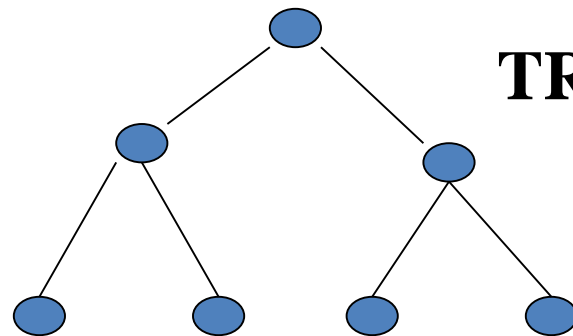
Possible Structures: Set, Linear, Tree, Graph.



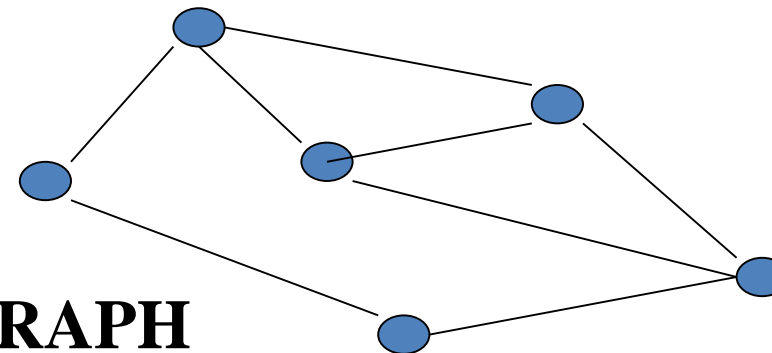
**SET**



**LINEAR**



**TREE**

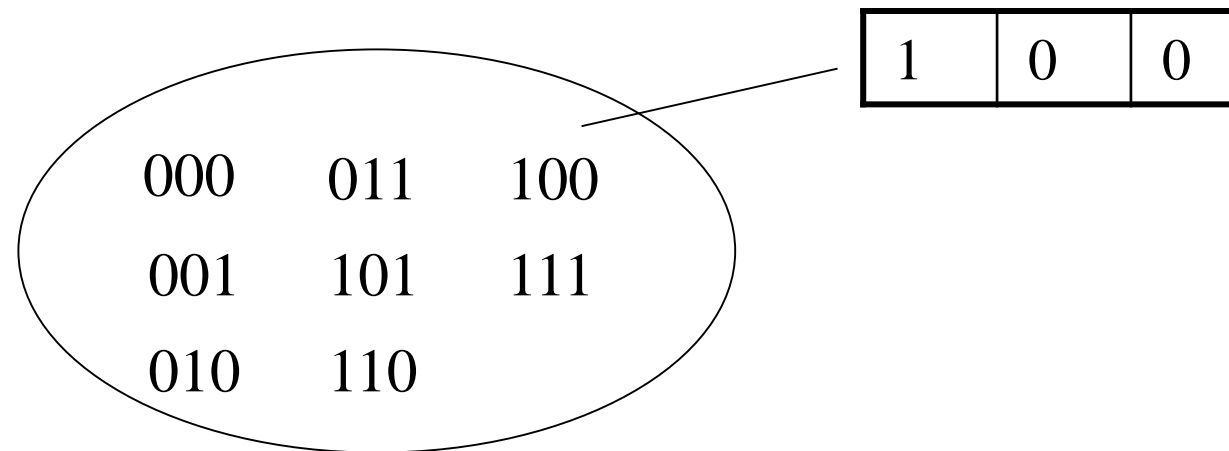


**GRAPH**

# Data Types & Data Structures

- What is the domain of a structured data type?  
Operations?
- Example: `boolean[] Sample[3];`

**Domain**

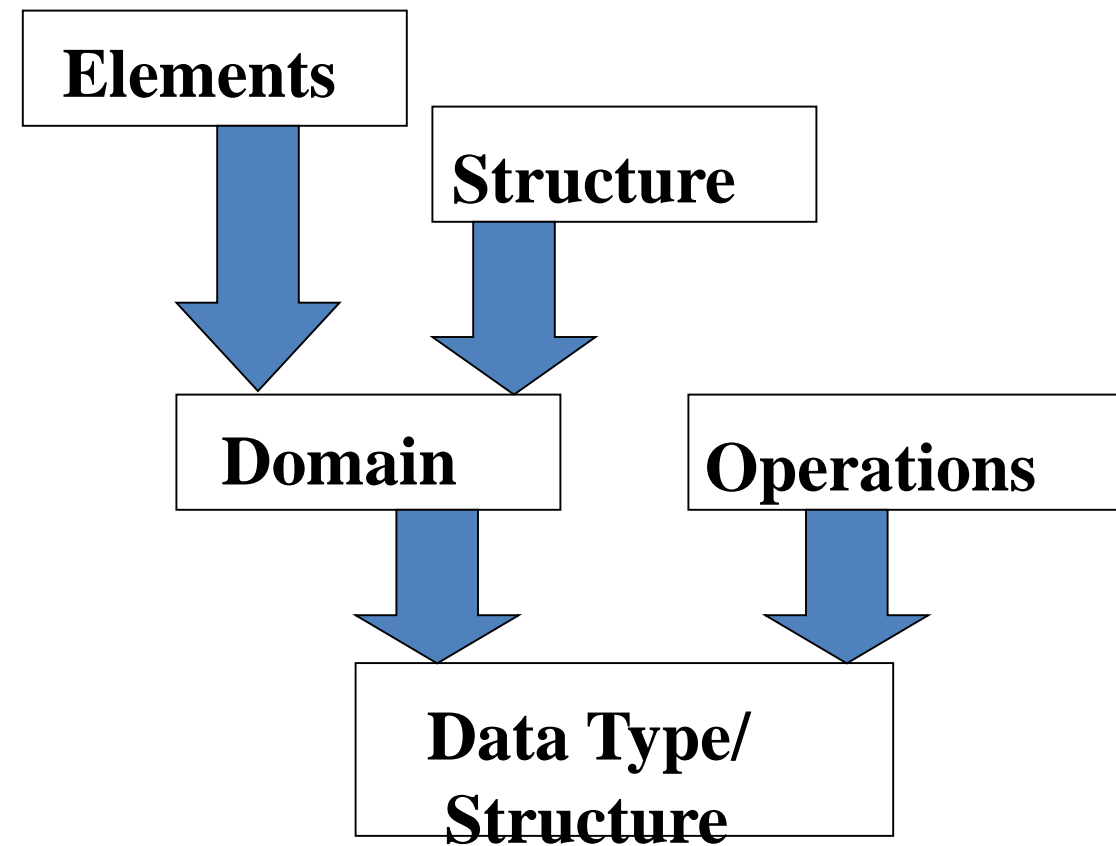


# Data Types & Data Structures

- Example: Operations:

`Sample[0] = True;`

`C = Sample[1]; etc.`



# Abstract Data Types (ADTs)

- **Abstraction?** Anything that hides details & provides only the essentials.
- Examples: an integer  $165 = 1 \cdot 10^2 + 6 \cdot 10^1 + 5 \cdot 10^0$ , procedures/subprograms, etc.
- **Abstract Data Types (ADTs):** Simple or structured data types whose implementation details are hidden...

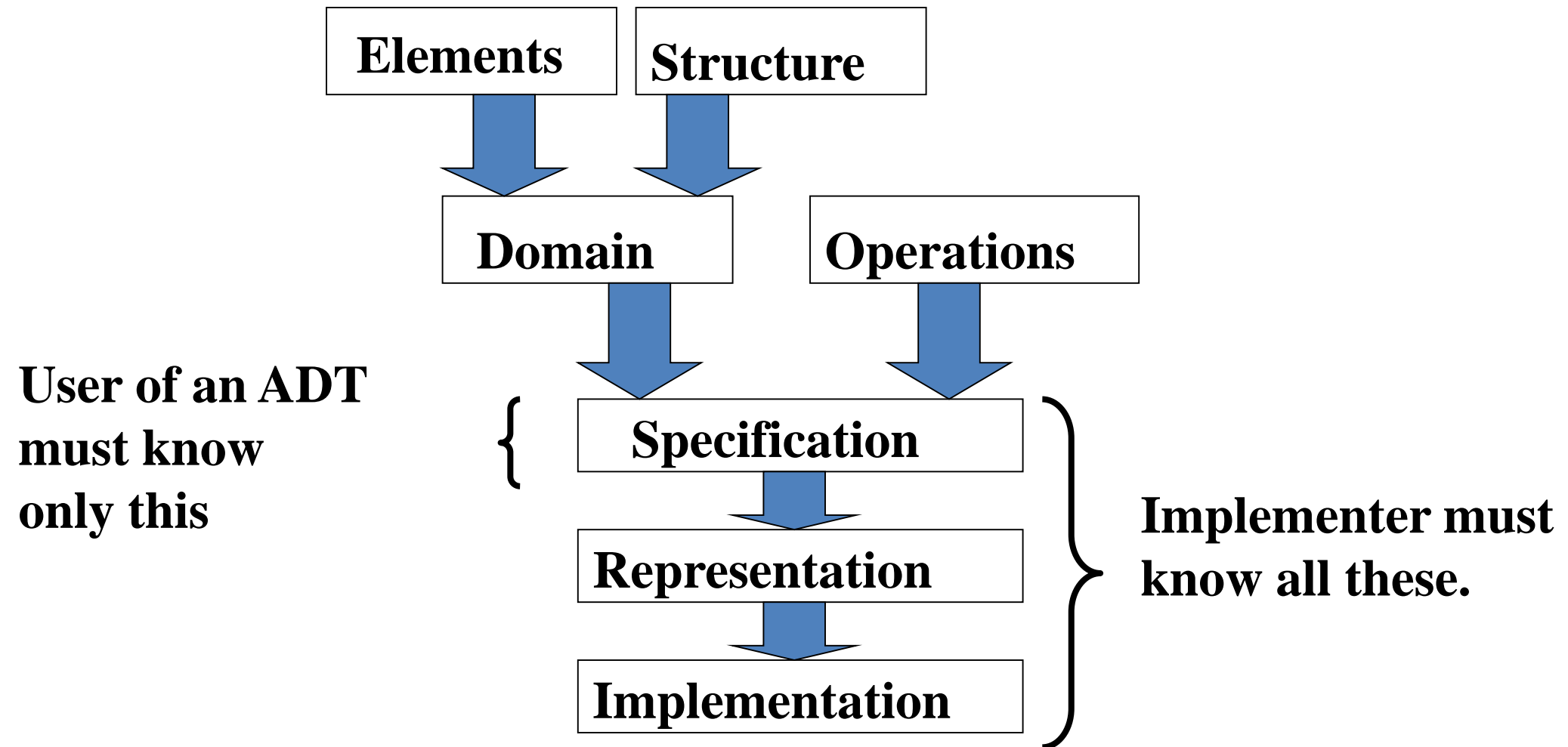
# ADTs

- While designing ADTs, a designer has to deal with two types of questions:
  - (i) **What** values are in the domain? **What** operations can be performed on the values of a particular data type?
  - (ii) **How** is the data type represented? **How** are the operations implemented?

# ADTs

- ADTs **specification** answers the ‘what’ questions. Specification is written first.
- ADTs **implementation** answers the ‘how’ questions. Done after specification.
- Users & Implementers.
- Users of an ADT need only know the specification ... no implementation details. ← advantage
- Programmer (Implementer) who implements ADT is concerned with..specification, representation, implementation.

# ADTs



# ADT: Example

## ADT String1

### Specification:

**Elements:** type char.

**Structure:** elements (characters) are linearly arranged.

**Domain:** type String, finite domain, there are 0 to 80 chars in a string, therefore  $1+128+128^2+\dots+128^{80}$  possible strings in the domain.

**Operations:** Assume that there is a string S.

1. **Procedure** Append (c: char)

**Requires:**  $\text{length}(S) < 80$ .

**Results:** c is appended to the right end of S.



# ADT: Example

2. Procedure Remove (c: char)

Requires:  $\text{length}(S) > 0$ .

Results: The rightmost character of S is removed and placed in c, S's length decreases by 1.

3. Procedure MakeEmpty ()

Results: all characters are removed.

4. Procedure Concatenate (R: String)

Results: String R is concatenated to the right of string S, result placed into S.

5. Procedure Reverse ()

6. Procedure Length (L: int)

7. Procedure Equal (S: String, flag: boolean)

8. Procedure GetChar (int i)

# Note ..

- In Java the *class* construct is used to declare new data types.
- In Java operations are implemented as function members of classes or methods.

# ADT String: Implementation

```
public class String1 extends Object {  
    private char[] str;  
    private int    size;  
  
    public String1 () {  
        size = -1;  
        str = new char[80];  
    }  
    public void Append (char c) {  
        size++;  
        str[size] = c;  
    }  
}
```



**Representation**



**Implementation**

# ADT String: Implementation

```
public char Remove () {  
    char c = str[size];  
    size--;  
    return(c);  
}  
public char GetChar(int i) {  
    return(str[i]);  
}  
public void MakeEmpty () {  
    size = -1;  
}  
public int Length () {  
    return(size); }  
}
```

# ADT String: Implementation

```
public void Concatenate (String1 s) {  
    for (int i = 0; i<=s.Length(); i++) {  
        char c = s.GetChar(i);  
        Append(c);  
    }  
}  
public boolean Equal (String1 s) {  
}  
public void Reverse () {  
}  
}
```

# Using ADT String

```
import java.lang.*;
public class Test {
    public static void main(String[] args) {
        String1 s = new String1();
        String1 s1 = new String1();
        System.out.println("Hello, World");
        s.Append(' a' );
        s1.Append(' b' );
        s.Concatenate(s1);
        System.out.print(s.GetChar(0));
        System.out.println(s.GetChar(1)); }
}
```

# Data Structures

# Data Structures

A data structure is a scheme for organizing data in the memory of a computer.

Some of the more commonly used data structures include lists, arrays, stacks, queues, heaps, trees, and graphs.



# Data Structures

The way in which the data is organized affects the performance of a program for different tasks.

Computer programmers decide which data structures to use based on the nature of the data and the processes that need to be performed on that data.

# Example: A Queue

A *queue* is an example of commonly used simple data structure. A queue has beginning and end, called the *front* and *back* of the queue.

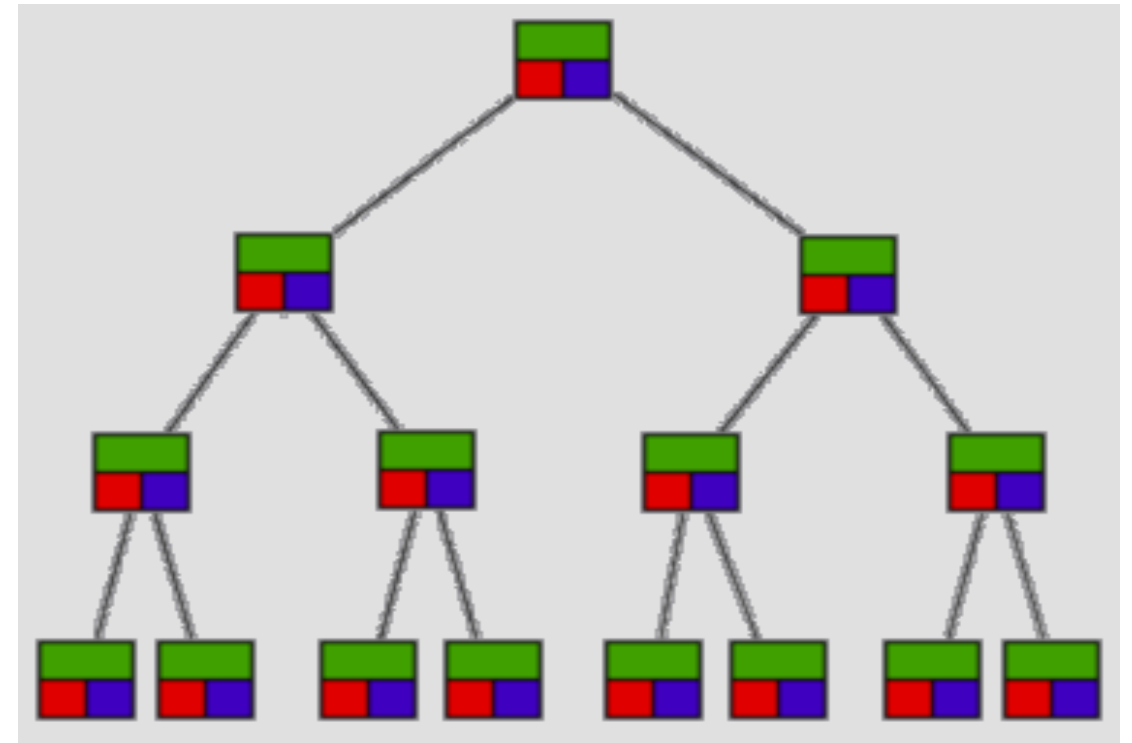


Data enters the queue at one end and leaves at the other. Because of this, data exits the queue in the same order in which it enters the queue, like people in a checkout line at a supermarket.

# Example: A Binary Tree

A *binary tree* is another commonly used data structure. It is organized like an upside down tree.

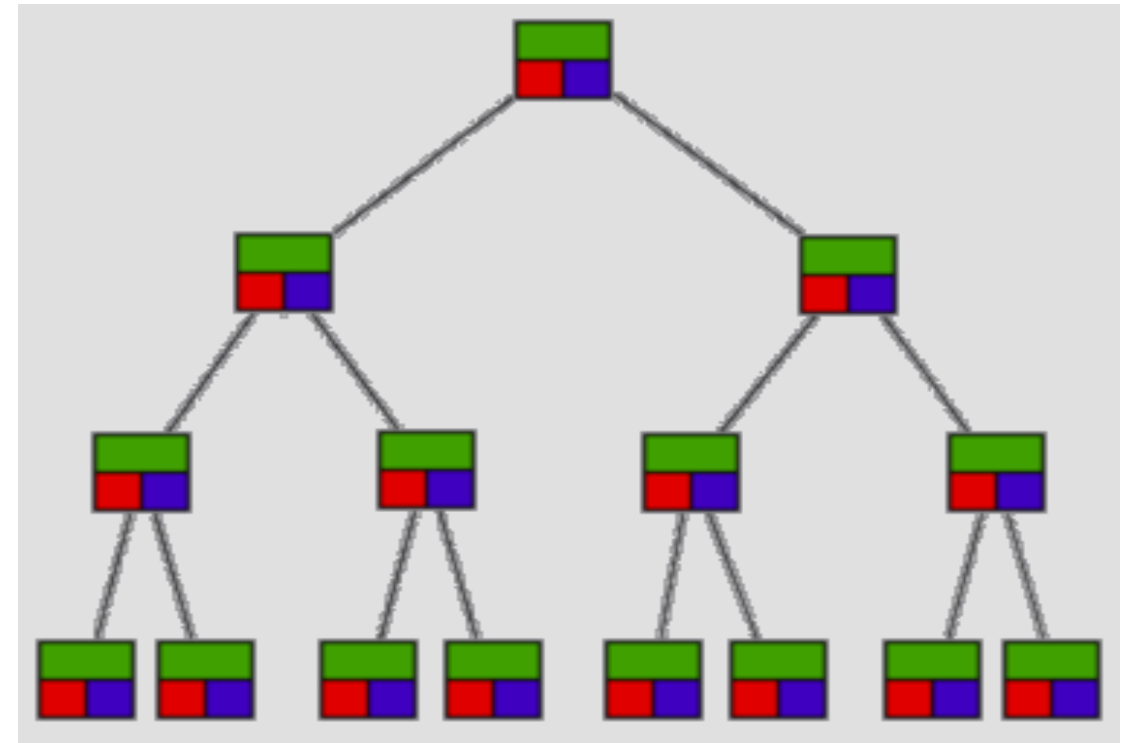
Each spot on the tree, called a *node*, holds an item of data along with a left pointer and a right pointer.



Binary Tree

# Example: A Binary Tree

The pointers are lined up so that the structure forms the upside down tree, with a single node at the top, called the root node, and branches increasing on the left and right as you go down the tree.



Binary Tree

# Choosing Data Structures -1

- By comparing the queue with the binary tree, you can see how the structure of the data affects what can be done efficiently with the data.
- A queue is a good data structure to use for storing things that need to be kept in order, such as a set of documents waiting to be printed on a network printer.
- The jobs will be printed in the order in which they are received.
- Most network print servers maintain such a print queue.
- A binary tree is a good data structure to use for searching sorted data.
- The middle item from the list is stored in the root node, with lesser items to the left and greater items to the right.
- A search begins at the root. The computer either find the data, or moves left or right, depending on the value for which you are searching.
- Each move down the tree cuts the remaining data in half.

# Choosing Data Structures -2

- Items can be located very quickly in a tree.
- Telephone directory assistance information is stored in a tree, so that a name and phone number can be found quickly.
- For some applications, a queue is the best data structure to use.
- For others, a binary tree is better.
- Programmers choose from among many data structures based on how the data will be used by the program.
  
- An array is an indexed set of variables, such as `dancer[1]`, `dancer[2]`, `dancer[3]`,... It is like a set of boxes that hold things.
- A list is a set of items.
- An array is a set of variables that each store an item.

# Data Structures

The any system has two built-in data structures that can be used to organize data, or to create other data structures:

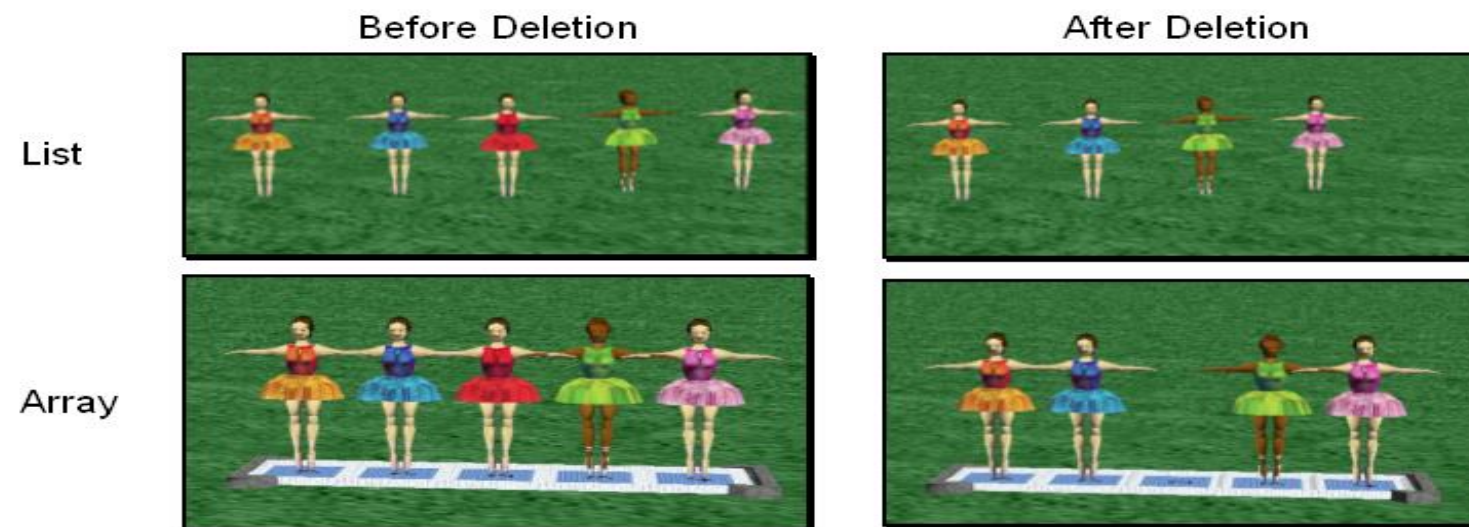
- Lists
- Arrays
- A list is an ordered set of data. It is often used to store objects that are to be processed sequentially.
- A list can be used to create a queue.

# Arrays and Lists

You can see the difference between arrays and lists when you delete items.

In a list, the missing spot is filled in when something is deleted.

In an array, an empty variable is left behind when something is deleted.

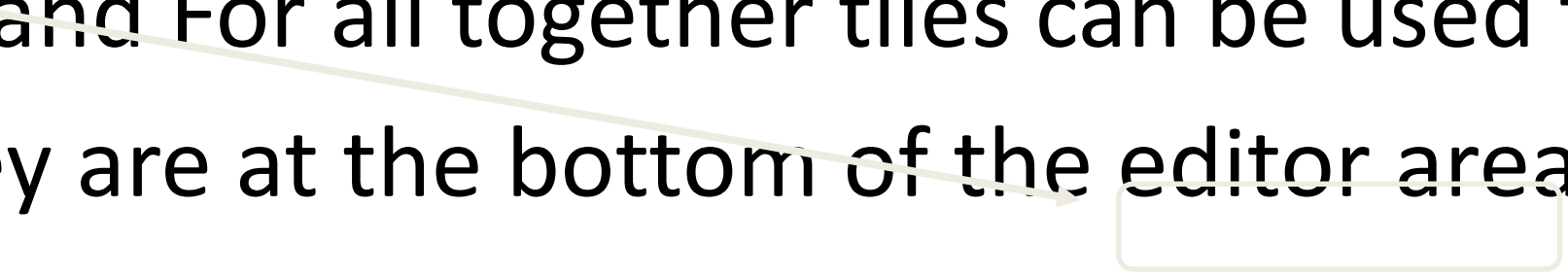




# Lists

A list is created in any system by checking the make a list box when creating a new variable.

The For all in order and For all together tiles can be used to work with lists. They are at the bottom of the editor area.



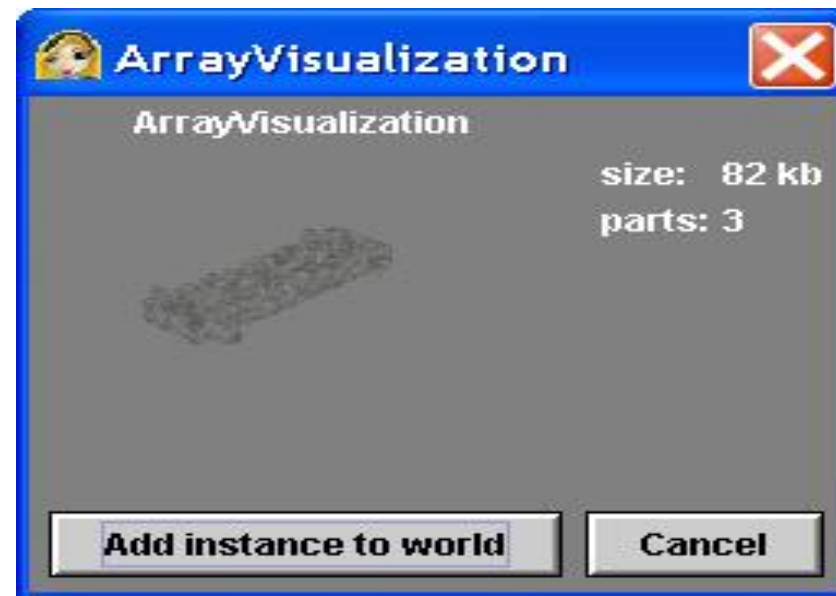
# Arrays

Arrays can be created in a similar manner, but more often they are created using the array visualization object from any system local gallery.

The Array Visualization object has special properties and methods for manipulating the elements in an array.

Any system has a set of built-in functions that can be performed on arrays.

# Arrays



# Data Structures : Algorithms

# Data Structures : Algorithms

- Algorithm
  - A high level, language independent description of a step-by-step process for solving a problem
- Data Structure
  - A set of algorithms which implement an ADT

# Why so many data structures?

Ideal data structure:

fast, elegant, memory efficient

Generates tensions:

- time vs. space
- performance vs. elegance
- generality vs. simplicity
- one operation's performance vs. another's

Dictionary ADT

- list
- binary search tree
- AVL tree
- Splay tree
- Red-Black tree
- hash table

# Code Implementation

- Theoretically
  - abstract base class describes ADT
  - inherited implementations implement data structures
  - can change data structures transparently (to client code)
- Practice
  - different implementations sometimes suggest different interfaces (*generality vs. simplicity*)
  - performance of a data structure may influence form of client code (*time vs. space, one operation vs. another*)

# ADT Presentation Algorithm

- Present an ADT
- Motivate with some applications
- Repeat until browned entirely through
  - develop a data structure for the ADT
  - analyze its properties
    - efficiency
    - correctness
    - limitations
    - ease of programming
- Contrast data structure's strengths and weaknesses
  - understand when to use each one



# Queue ADT

- Queue operations
  - create
  - destroy
  - enqueue
  - dequeue
  - is\_empty



- Queue property: if  $x$  is enQed before  $y$  is enQed, then  $x$  will be deQed before  $y$  is deQed

FIFO: First In First Out

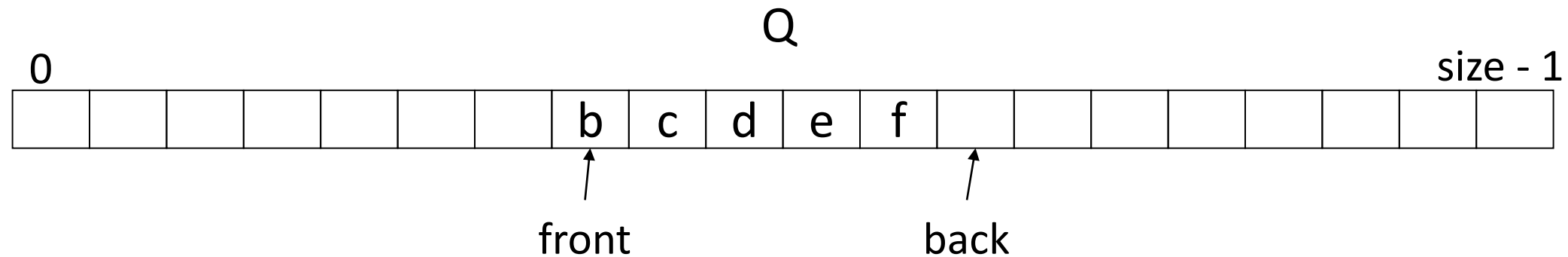
# Queue ADT

- You've probably seen the Queue before. If so, this is a review and a way for us to get comfortable with the format of data structure presentations in this class. If not, this is a simple but very powerful data structure, and you should make sure you understand it thoroughly.
- This is an ADT description of the queue. Notice that there are no implementation details. Just a general description of the interface and important properties of those interface methods.

# Applications of the Q

- Hold jobs for a printer
- Store packets on network routers
- Hold memory “freelists”
- Make waitlists fair
- Breadth first search

# Circular Array Q Data Structure



```
void enqueue(Object x) {
    Q[back] = x
    back = (back + 1) % size
}
Object dequeue() {
    x = Q[front]
    front = (front + 1) % size
    return x
}
```

When is the Q empty?

Are there error situations this code will not catch?

What are some limitations of this structure?

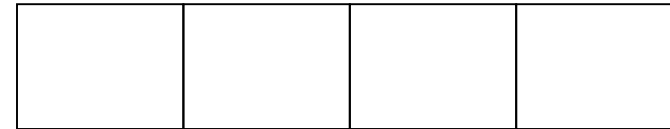
This is *pseudocode*. Do not correct my semicolons.

# Here is a data structure implementation of the Q.

- The queue is stored as an array, and, to avoid shifting all the elements each time an element is dequeued, we imagine that the array wraps around on itself.
- This is an excellent example of how implementation can affect interface: notice the “is\_full” function.
- There’s also another problem here. What’s wrong with the Enqueue and Dequeue functions?
- Your data structures should be robust! Make them robust before you even consider thinking about making them efficient! That is an order!

# Q Example

enqueue R



enqueue O

dequeue

enqueue T

enqueue A

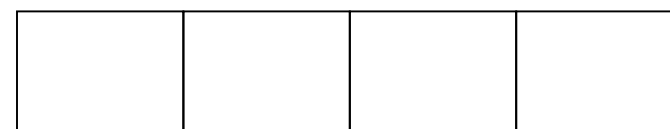
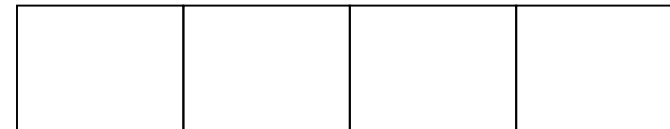
enqueue T

dequeue

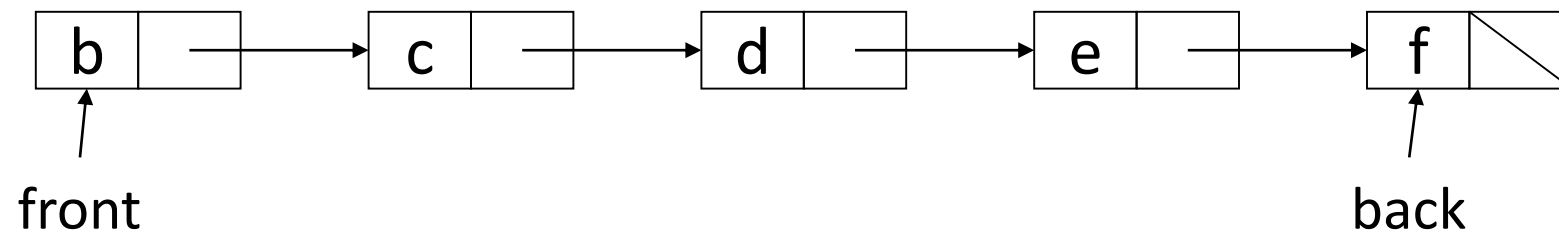
dequeue

enqueue E

dequeue



# Linked List Q Data Structure

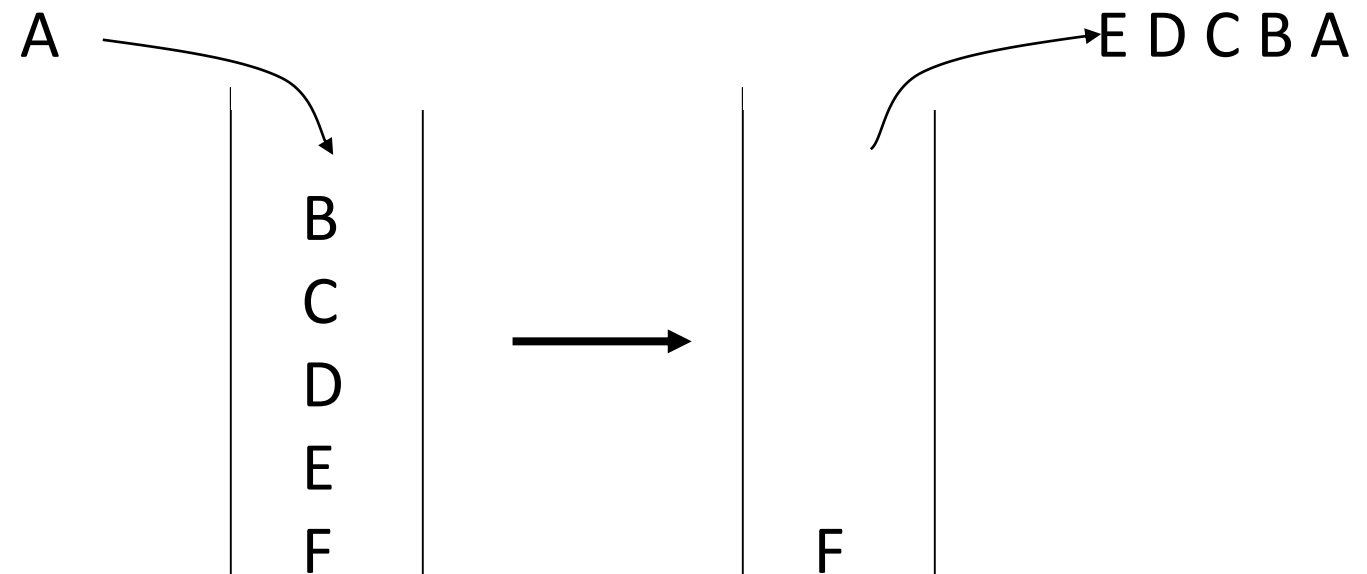


```
void enqueue(Object x) {  
    if (is_empty())  
        front = back = new Node(x)  
    else  
        back->next = new Node(x)  
        back = back->next  
}
```

```
Object dequeue() {  
    assert(!is_empty)  
    return_data = front->data  
    temp = front  
    front = front->next  
    delete temp  
    return temp->data  
}
```

# LIFO Stack ADT

- Stack operations
  - create
  - destroy
  - push
  - pop
  - top
  - is\_empty



- Stack property: if x is on the stack before y is pushed, then x will be popped after y is popped

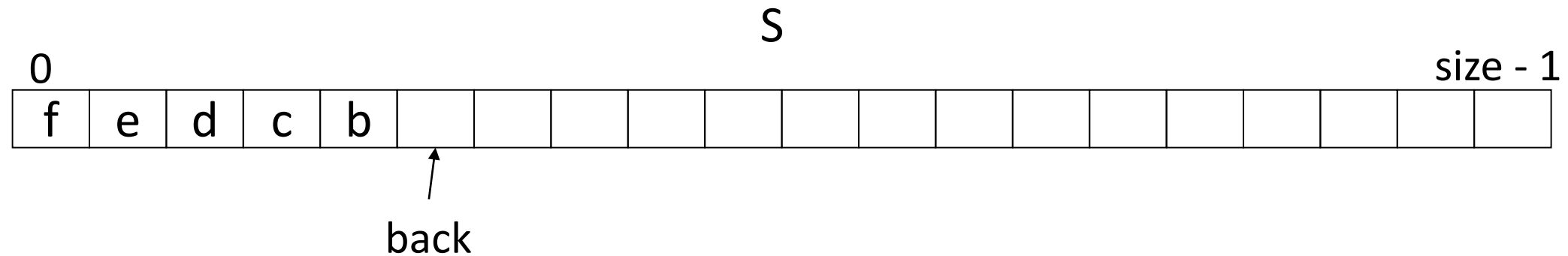
LIFO: Last In First Out



# Stacks in Practice

- Function call stack
- Removing recursion
- Balancing symbols (parentheses)
- Evaluating Reverse Polish Notation
- Depth first search

# Array Stack Data Structure

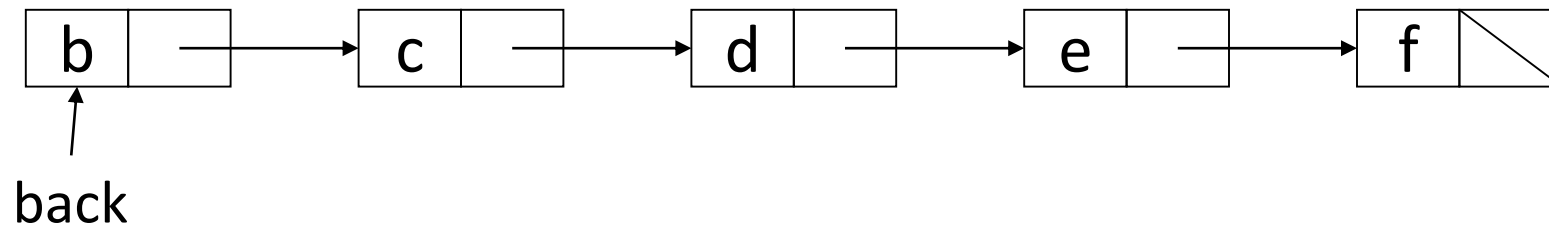


```
void push(Object x) {
    assert(!is_full())
    S[back] = x
    back++
}
Object top() {
    assert(!is_empty())
    return S[back - 1]
}
```

```
Object pop() {
    back--
    return S[back]
}

bool is_full() {
    return back == size
}
```

# Linked List Stack Data Structure



```
void push(Object x) {  
    temp = back  
    back = new Node(x)  
    back->next = temp  
}
```

```
Object top() {  
    assert(!is_empty())  
    return back->data  
}
```

```
Object pop() {  
    assert(!is_empty())  
    return_data = back->data  
    temp = back  
    back = back->next  
    return return_data  
}
```

# Data structures you should already know

- Arrays
- Linked lists
- Trees
- Queues
- Stacks

# Database Applications

## CS 15-415

Introduction

Lecture 1, January 7, 2018

Mohammad Hammoud

# Data and *Big* Data

- The value of data as an organizational asset is widely recognized
- Data is literally exploding and is occurring along three main dimensions
  - “Volume” or the amount of data
  - “Velocity” or the speed of data
  - “Variety” or the range of data types and sources
- What is **Big Data**?
  - It is the proliferation of data that floods organizations on a daily basis
  - It is *high* volume, *high* velocity, and/or *high* variety information assets
  - It requires new forms of processing to enable *fast* mining, enhanced decision-making, insight discovery and process optimization

# What Do We Do With Data and Big Data?



We want to do these *seamlessly* and *fast*...


# Why Studying Databases?


- Data is *everywhere* and is *critical* to our lives
- Data need to be recorded, maintained, accessed and manipulated *correctly, securely, efficiently* and *effectively*
  - At the “low end”: scramble to web-scale (a mess!)
  - At the “high end”: scientific applications
- Database management systems (DBMSs) are indispensable software for achieving such goals
- The principles and practices of DBMSs are now an integral part of computer science curricula
  - They encompass OS, languages, theory, AI, multimedia, and logic, among others


As such, the study of database systems can prove to be richly rewarding in more ways than one!



# List of Topics

 **Considered:** a reasonably critical and comprehensive understanding.

 **Thoughtful:** fluent, flexible and efficient understanding.

 **Masterful:** a powerful and illuminating understanding.

.1.

**The Entity-Relationship Model**

.2.

**The Relational Model**

.3.

**Relational Algebra and Calculus**

.4.

**SQL**

.5.

**Data Storage and Organization**

.6.

**Tree-Based and Hash-Based Indexing**

.7.

**Query Evaluation and Optimization**

.9.

**Concurrency Control and Crash Recovery**

.10.

**Advanced Topics: Distributed Databases, Hadoop, and NoSQL and NewSQL Databases**

# Data Base Management Systems

- A special software is accordingly needed to make the preceding tasks easier
- This software is known as **Data Base Management System (DBMS)**
- DBMSs provide automatic:
  - Data independence
  - Efficient data access
  - Data integrity and security
  - Data administration
  - Concurrent access and crash recovery
  - Reduced application development and tuning time

# Some Definitions

- A **database** is a collection of data which describes one or many real-world enterprises
  - E.g., a university database might contain information about **entities** like students and courses, and **relationships** like a student enrollment in a course
- A **DBMS** is a software package designed to store and manage databases
  - E.g., DB2, Oracle, MS SQL Server, MySQL and Postgres
- A **database system** = (Big) Data + DBMS + Application Programs

# Data Models

- The user of a DBMS is ultimately concerned with some real-world enterprises (e.g., a University)
- The data to be stored and managed by a DBMS *describes* various aspects of the enterprises
  - E.g., The data in a university database describes students, faculty and courses entities and the relationships among them
- A **data model** is a collection of high-level data description constructs that hide many low-level storage details
- A widely used data model called the **entity-relationship (ER) model** allows users to pictorially denote entities and the relationships among them

# The Relational Model

- The **relational model** of data is one of the most widely used models today
- The central data description construct in the relational model is the **relation**
- A relation is basically a **table** (or a **set**) with **rows** (or **records** or **tuples**) and **columns** (or **fields** or **attributes**)
- Every relation has a **schema**, which describes the columns of a relation
- Conditions that records in a relation must satisfy can be specified
  - These are referred to as **integrity constraints**

# People Who Work With Databases

- There are five classes of people associated with databases:
  1. **End users**
    - Store and use data in DBMSs
    - Usually not computer professionals
  2. **Application programmers**
    - Develop applications that facilitate the usage of DBMSs for end-users
    - Computer professionals who know how to leverage host languages, query languages and DBMSs altogether
  3. **Database Administrators (DBAs)**
    - Design the conceptual and physical schemas
    - Ensure security and authorization
    - Ensure data availability and recovery from failures
    - Perform database tuning
  4. **Implementers**
    - Build DBMS software for vendors like IBM and Oracle
    - Computer professionals who know how to build DBMS internals
  5. **Researchers**
    - Innovate new ideas which address evolving and new challenges/problems

# **Features of Database Management System**

# Database Management System

**Database:** A collection of related data. It should support

- Definition
- Construction
- Manipulation

**Database Management System:** A collection of programs that enable the users to create and maintain a database.



# Features of DBMS

1. Data storage, retrieval, and update: The ability to store, retrieve, and update the data that are in the database.
2. User-accessible catalog: where descriptions of database components are stored and are accessible to the users
3. Shared update support: A mechanism to ensure accuracy when several users are updating the database at the same time
4. Backup and Recovery Services: Mechanisms for recovering the database in the event that a database is damaged somehow.
5. Security Services: Mechanisms to ensure that certain rules are followed with regard to data in the database and any changes that are made in the data

# Features of DBMS

5. Integrity services: Mechanisms to ensure that certain rules are followed with regard to data in the database and any changes that are made in the data.
6. Data Independence: Facilities to support the independence of programs from the structure of the database.
7. Replication support: A facility to manage copies of the same data at multiple locations.
8. Utility Services: DBMS provided services that assist in the general maintenance of the database.

# Shared Updates

- Multiple users are making updates to the database at the same time.

## Problem:

- Multiple people updating the database simultaneously can override each other

## Example:

- Agents T1 & T2 simultaneously read the seats reserved on Flight 890 i.e. 80
- T1 cancels 5 seats updating the seats reserved on Flight 890 to 75
- T2 reserves 4 additional seats on the flight and updates the seats reserved on Flight 890 to 84.
- If T1 updates the database before T2. T2 will override T1's change and make reservations to 84 rather than getting the correct value of 79.
- Similarly if T2 updates before T1 the seats reserved will be 75

# Shared Updates: Solution

- Batch Processing
  - Allow multiple users to retrieve data simultaneously
  - Updates are added to a batch file which does the appropriate processing
  - Does not work for real time situations
- Locking
  - Restrict access to the record being updated by a user till the transaction is complete.

# Two Phase Lock

- Required when multiple records are updated as a result of a user action (e.g. filling form etc.)
- All the records accessed are locked progressively till the required updates are completed
  - Growing Phase: More and more locks are added without releasing locks
  - After all locks are placed the database is updated
  - Shrinking Phase: All locks are removed and no new ones are added

# Deadlock

- When two transactions require a common set of records.
- Both of them are in growing phase and each locks some of the records
- None of the records are released and they wait for each other to release the locked records

They will wait forever!!!

# Breaking Deadlock

## Facilities

- Programs can lock entire tables or an individual row
- Programs can release any or all of the locks they currently hold
- Programs can inquire whether a given row or table is locked

## Rules

- If more than one row is required then the entire table must be locked
- Limit the amount of wait for a lock to be released beyond which a transaction is aborted
- A well designed transaction should lock all the rows and tables before starting the transaction
- Users should release locks as soon as possible to improve the efficiency of the database

# Security

- Protection against unauthorized access: either intentional or accidental.
- Three main features for protection
  - **Passwords:** Allows only authorized users to access the database. Access privileges can be provided based on access needs
  - **Encryption:** Encodes data to non-decipherable. Data decoded on demand to prevent hackers from accessing data
  - **Views:** Different snapshot of the data ensures that users only get access to data they need



# Integrity

- Integrity Constraints are the conditions that data must satisfy during initial input & updates.
- There are four categories of constraints
  - Data Type
  - Legal Values
  - Format
  - Key Constraints
    - Entity Integrity Constraints (Primary Key)
      - Enforces the uniqueness of the primary key
    - Referential Integrity Constraints (Foreign Key)
      - Value of foreign key must match the value of primary key for some row in another table

# Integrity: Solutions

- Ignore constraint
  - Undesirable as it can lead to inconsistent data
- Let user enforce the constraint
  - Undesirable since user mistakes can be disastrous
- Let programmer build the logic of constraints in the programs
  - Makes programs complex: harder to write, harder to maintain, and expensive
- Place burden on the DBMS.
  - Preferred way: Cost of DBMS development amortized over large user base, hence economical

# Replication

- Duplication of data at multiple physical locations
- Each replica of the data can be changed independently
- Periodically the replicas update their data to the master database – this process is called synchronization

# Disaster Planning: Backup & Recovery

- Database can be damaged in a number of ways
  - Power outage, disk crashes, floods, user errors
- Periodic backups limit the loss due to sudden failures
- Data can be recovered from the latest backup and the changes since the backup need to be done in either of two ways
  - Manually
  - From a catalog (if exists) recording all updates to the database since the last backup.

# Catalog/Data Dictionary

- Contains information describing the database
  - Schema for the database
  - Characteristic for each field
  - Possible values for each field
  - Description of the data
  - Relationships
  - Description of the programs
- Data Dictionary is same as catalog but may contain wider set of information than catalog