COURSENOTES

CS2604:
Data Structures and File Processing

Java Edition

Clifford A. Shaffer
Department of Computer Science
Virginia Tech

The Need for Data Structures

Data structures organize data

$\Rightarrow$ **more efficient programs**.

- More powerful computers $\Rightarrow$ more complex applications.
- More complex applications demand more calculations.
- Complex computing tasks are unlike our everyday experience.

Any organization for a collection of records can be searched, processed in any order, or modified.

- The choice of data structure and algorithm can make the difference between a program running in a few seconds or many days.

Efficiency

A solution is said to be **efficient** if it solves the problem within its **resource constraints**.

- space
- time

The **cost** of a solution is the amount of resources that the solution consumes.

Selecting a Data Structure

Select a data structure as follows:

1. Analyze the problem to determine the resource constraints a solution must meet.
2. Determine the basic operations that must be supported. Quantify the resource constraints for each operation.
3. Select the data structure that best meets these requirements.

Some questions to ask:

- Are all data inserted into the data structure at the beginning, or are insertions interspersed with other operations?
- Can data be deleted?

- Are all data processed in some well-defined order, or is random access allowed?

Data Structure Philosophy

Each data structure has costs and benefits.

Rarely is one data structure better than another in all situations.

A data structure requires:

- space for each data item it stores,
- time to perform each basic operation,
- programming effort.

Each problem has constraints on available space and time.

Only after a careful analysis of problem characteristics can we know the best data structure for the task.

Bank example:

- Start account: a few minutes
- Transactions: a few seconds
- Close account: overnight

Goals of this Course

1. Reinforce the concept that there are costs and benefits for every data structure.

2. Learn the commonly used data structures. These form a programmer's basic data structure "toolkit."

3. Understand how to measure the effectiveness of a data structure or program.
   - These techniques also allow you to judge the merits of new data structures that you or others might invent.

Definitions

A **type** is a set of values.

A **data type** is a type and a collection of operations that manipulate the type.

A **data item** or **element** is a piece of information or a record.

A data item is said to be a **member** of a data type.

A **simple data item** contains no subparts.

An **aggregate data item** may contain several pieces of information.

Abstract Data Types

Abstract Data Type (ADT): a definition for a data type solely in terms of a set of values and a set of operations on that data type.

Each ADT operation is defined by its inputs and outputs.

Encapsulation: hide implementation details

A **data structure** is the physical implementation of an ADT.

- Each operation associated with the ADT is implemented by one or more subroutines in the implementation.

Data structure usually refers to an organization for data in main memory.

File structure: an organization for data on peripheral storage, such as a disk drive or tape.

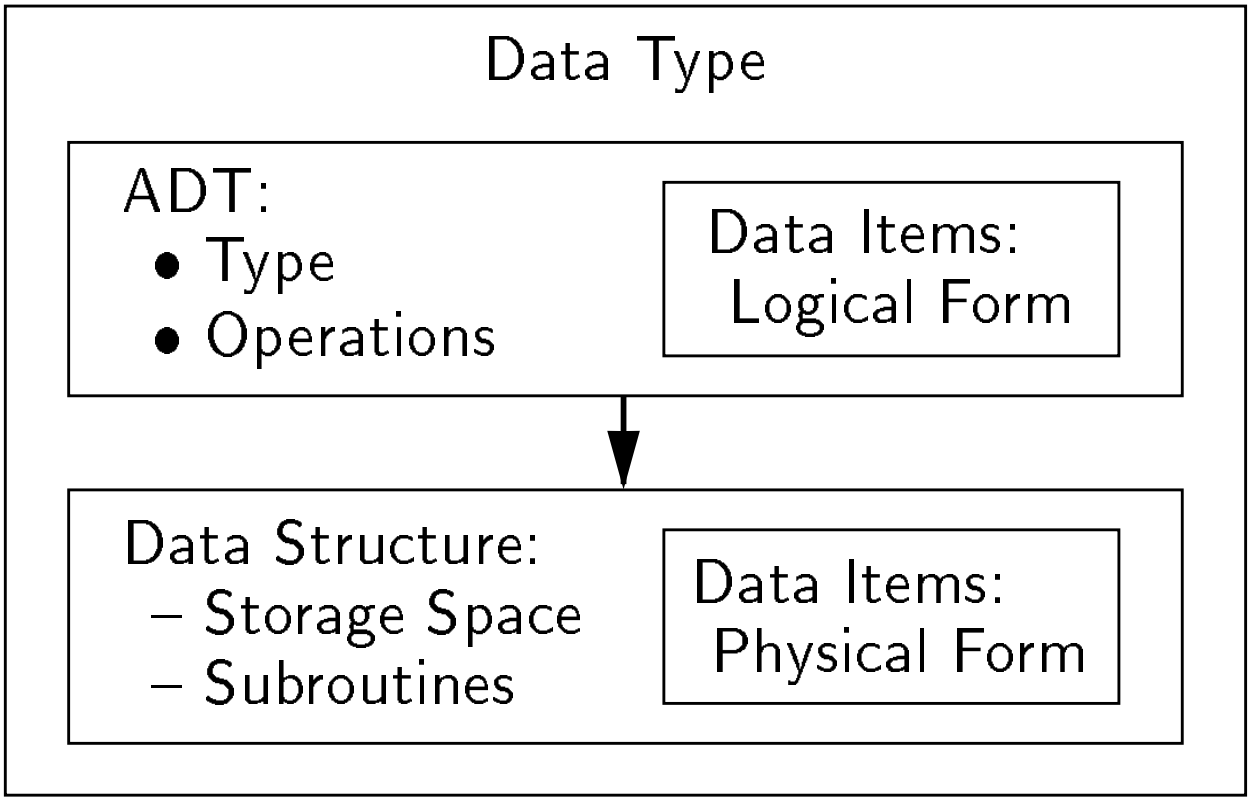
An ADT manages complexity through abstraction: **metaphor**.

Logical vs. Physical Form

Data items have both a **logical** and a **physical** form.

Logical form: definition of the data item within an ADT.

Physical form: implementation of the data item within a data structure.



Problems

Problem: a task to be performed.

- Best thought of as inputs and matching outputs.
- Problem definition should include constraints on the resources that may be consumed by any acceptable solution.

Problems $\Leftrightarrow$ mathematical functions

- A **function** is a matching between inputs (the **domain**) and outputs (the **range**).
- An **input** to a function may be single number, or a collection of information.
- The values making up an input are called the **parameters** of the function.
- A particular input must always result in the same output every time the function is computed.

Algorithms and Programs

Algorithm: a method or a process followed to solve a problem.

An algorithm takes the input to a problem (function) and transforms it to the output.

A problem can have many algorithms.

An algorithm possesses the following properties:

1. It must be **correct**.
2. It must be composed of a series of **concrete steps**.
3. There can be **no ambiguity** as to which step will be performed next.
4. It must be composed of a **finite** number of steps.
5. It must **terminate**.

A **computer program** is an instance, or concrete representation, for an algorithm in some programming language.

Mathematical Background

Set concepts and notation

Recursion

Induction proofs

Logarithms

Summations

Estimation Techniques

Known as "back of the envelope" or "back of the napkin" calculation.

1. Determine the major parameters that affect the problem.
2. Derive an equation that relates the parameters to the problem.
3. Select values for the parameters, and apply the equation to yield an estimated solution.

Example:

How many library bookcases does it take to store books totaling one million pages?

Estimate:

- pages/inch
- feet/shelf
- shelves/bookcase