

# Intro to Data Structures & Algorithms

Wagner Truppel  
Lecturer, Dept. of Computer Science & Engineering  
UC Riverside

[wagner@cs.ucr.edu](mailto:wagner@cs.ucr.edu)  
<http://www.cs.ucr.edu/~wagner>

<http://www.cs.ucr.edu/cs14>

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 1

---

---

---

---

---

---

---

---

# Today's Topics

- Overview of Data Structures & Algorithms
  - ◆ The "bookstore" example
- Abstraction:
  - ◆ Interface
  - ◆ Implementation
- Some notation to describe interfaces
- Syntax and Semantics
- Abstract Data Types (ADTs)
- ADT example: Counter
- Summary

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 2

---

---

---

---

---

---

---

---

# Overview of DS & As

- CS 14 is about how to **store**, **retrieve**, and **manipulate** data *efficiently*
- storage/retrieval → Data Structures
- data manipulation → Algorithms
  
- In CS 10, you learned to crawl...
- In CS 12, you learned to walk...
- In CS 14, you'll learn how to **run** !  
[ and break your leg... :)]

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 3

---

---

---

---

---

---

---

---

## Overview of DS & As

- The “bookstore” example
  - ◆ How do you find the book you’re looking for ? You need to use some kind of algorithm and...
  - ◆ ... of course, we want the best algorithm possible, but...
  - ◆ ... what “best” means will depend on how the books are organized (stored)
- Data structures and algorithms are always intertwined

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 4

---

---

---

---

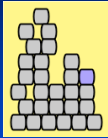
---

---

---

---

## MessyBookPile, Inc.



- This bookstore keeps its books all in one big pile, with no organization of any kind
- Your algorithm is then:
  - ◆ For each book
    - \* If it is not the book I want, continue searching
- If the bookstore has **1,000,000 books**, and if it takes **10 secs** for you to check each book, it may take you as long as **4 months** to find the book you want
- Not a very good algorithm, is it ?
- **Note:** search time is directly proportional to number of books:  **$O(n)$**

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 5

---

---

---

---


---

---

---

---

## LongBookShelf, Inc.



- This competitor bookstore keeps its books all in one very long **array, sorted by ISBN**
- One possible algorithm is again **linear search**:
  - ◆ For each book
    - \* If it is not the book I want, continue searching
- This is the same algorithm used by MessyBookPile, Inc.
- Again... search time is directly proportional to number of books:  **$O(n)$**
- Note that the algorithm did **not** make use of the fact that the books are sorted

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 6

---

---

---

---

---

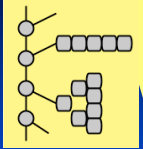
---

---

---



### CategoriesRUs, Inc.



- This other competitor bookstore keeps its books all organized by **sections**
- The search algorithm is now:
  - ◆ For each section
    - ★ If it is the section of the book I want
      - Search each book in that section using one of the previous algorithms
- This is typically much faster than the corresponding algorithm used in the previous slides

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 10

---

---

---

---

---

---

---

---

### Bottom line...

- The choice and efficiency of an algorithm depends on how the data it accesses is organized
- Likewise, how you're going to store your data depends on what you want to do with it.
- For example, if you don't care about searching, you might not care if the algorithm is **linear search** or **binary search**. That, in turn, helps you decide on which data structure to use to store your data.
- In addition, a data structure depends on the properties of the data they store
- Pile of books: requires **equality** operator
- Sorted books: requires **less than** operator

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 11

---

---

---

---

---

---

---

---

### Interface x Implementation

- Consider an **int** variable
- **int** is a data type
- It could be that the computer you're using stores **int**'s in **1's complement** or **2's complement** format
- When you use it, you don't care how it's really implemented inside the computer
- All you care is that the following operations are defined for that particular data type:
  - ◆ +, -, \*, /, <, >, <=, >=, ==, !=, ++, --
- A data type like that is called an **Abstract Data Type** or **ADT**

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 12

---

---

---

---

---

---

---

---

## Interface x Implementation

- So what you *usually* care about is the **interface** of an ADT, not its **implementation**
- *The interface of an ADT is the set of operations defined for it*
- So, in the bookstore example, you could create a general interface for your bookstore, and then have as many individual implementations as you wanted
- This sounds like OOP...

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 13

---

---

---

---

---

---

---

---

## Interface x Implementation

```

classDiagram
    class BookSet {
        +void add(Book b)
        +void remove(Book b)
        +bool has(Book b)
    }
    class SortedArrayBookSet {
        +Attributes
        +Operations
    }
    class BinaryTreeBookSet {
        +Attributes
        +Operations
    }
    class HashBookSet {
        +Attributes
        +Operations
    }
    BookSet <|-- SortedArrayBookSet
    BookSet <|-- BinaryTreeBookSet
    BookSet <|-- HashBookSet
    
```

Note that *BookSet* is an **abstract class** !

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 14

---

---

---

---

---

---

---

---

## Interface x Implementation

- Consider an `int` variable
- `int` is a data type
- It could be that the computer you're using stores `int`'s in **1's complement** or **2's complement** format
- When you use it, you don't care how it's really implemented inside the computer
- All you care is that the following operations are defined for that particular data type:  
`+`, `-`, `*`, `/`, `<`, `>`, `<=`, `>=`, `==`, `!=`, `++`, `--`
- A data type like that is called an **Abstract Data Type** or **ADT**

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 15

---

---

---

---

---

---

---

---

## Interface x Implementation

- A data type like that is called an **Abstract Data Type** or **ADT**
- Well... not quite: an interface alone does not define an ADT (more on this later)
- Still... an interface **is** part of the definition of an ADT
- How do you describe an interface?
- We need some notation...

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 16

---

---

---

---

---

---

---

---

## Notation

- Say we want to describe what the operation  $+$  does to integers
  - ♦  $+$  takes *two* ints as its operands  $\rightarrow$  it's a *binary* operation
  - ♦  $+$  produces an *int* as a result
- Same with  $-$
- But there's another kind of  $-$  operation:
  - ♦ It takes *one* int and changes its sign  $\rightarrow$  a *unary* operation
- In general, then, we need to specify the number of operands and their types (can anyone say *signature*?), as well as the type of the result

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 17

---

---

---

---

---

---

---

---

## Notation

$+$  : int X int  $\rightarrow$  int (binary  $+$ )  
 $-$  : int X int  $\rightarrow$  int (binary  $-$ )  
 $-$  : int  $\rightarrow$  int (unary  $-$ )

- Don't panic ! This is just a convenient notation. No deep concepts involved here...
- Now, is this enough though ?
- The function
 

```
int f(int a, int b) { return a; }
```

 satisfies the definition for  $+$  but it does not correspond to addition !

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 18

---

---

---

---

---

---

---

---

## Syntax and Semantics

- The interface (ie, set of operations) defines the **syntax** of the ADT
- We also need to define its **semantics**
- Say again ?
- Take English as an example:
  - ◆ The computer went out for lunch.
  - ◆ It satisfies the grammatical rules of English (its *syntax*)
  - ◆ It makes no sense, though
  - ◆ It violates the *semantics* of English
- **Semantics** defines *meaning*

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 19

---

---

---

---

---

---

---

---

## Quick Summary

- Thus, we have:
  - ◆ ADT
    - ★ Interface (operations' syntax)
    - ★ Axioms (expected semantics)
  - ◆ Implementation
- *An ADT is made of an interface and its semantics*
- The semantics is like a contract:
  - ◆ It specifies what is expected from the operations defined by the interface

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 20

---

---

---

---

---

---

---

---

## Back to + and -

- Interface (syntax):
  - + : int X int → int (binary +)
  - : int X int → int (binary -)
  - : int → int (unary -)
- Semantics (contract):
  - ◆ + does indeed add its operands
  - ◆ Binary - does indeed subtract its operands from one another in a specified order
  - ◆ Unary - does indeed change its operand's sign
- **Interface + semantics = ADT**

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 21

---

---

---

---

---

---

---

---

## Another example: Counter

- We want to define an ADT called Counter
- What is it good for?
- What should its interface (syntax) be?
  - ◆ `get`: to access the counter's current value
  - ◆ `inc`: to increment the counter's current value
- Hmm... ok, but just b/c we called them that, it doesn't mean that they do what we expect them to do
- How do we describe the semantics of the Counter interface?
- We need more notation: **axioms**

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 22

---

---

---

---

---

---

---

---

## Counter

```

adt CounterADT // "filename"
uses Integer // other adt's used
defines Counter // name of type defined
operations
  get: Counter → int // "observer"
  inc: Counter → Counter // "modifier"
  new: → Counter // "constructor"
preconditions // none for this adt
axioms // "contract"
  get( new() ) = 0 // new starts at 0
  get( inc(c) ) = get(c) + 1 // inc's by 1
    
```

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 23

---

---

---

---

---

---

---

---

## Counter

- Ok, now what?
- Now we have an ADT (an interface) for a data type which counts up from zero in steps of 1
- Nowhere we said anything about how to implement this ADT
- Data structures are typically ADTs
  - ◆ They define a set of operations and the corresponding semantics
  - ◆ But they can be implemented in many different ways, each with its own advantages and disadvantages
- And that's really *it*... we've already covered **all** of CS 14's *basics*.

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 1 24

---

---

---

---

---

---

---

---