THE UNIVERSITY OF AUCKLAND
NEW ZEALAND

COMPSCI 105 S1 2017
Principles of Computer Science

12 Abstract Data Type

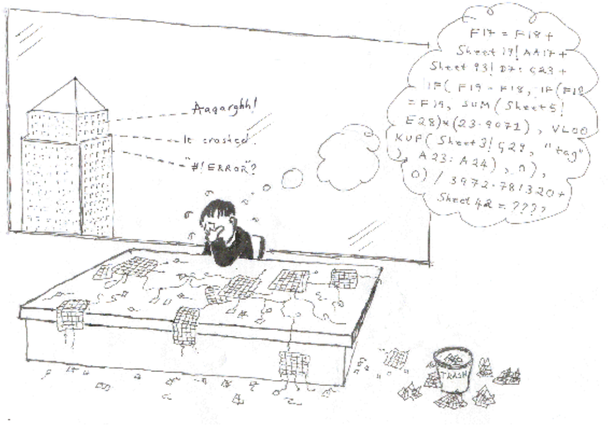
Agenda & Readings

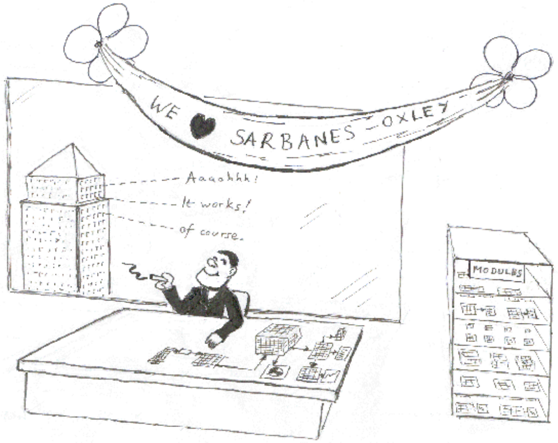
- Agenda
  - Some Software design Principles
  - Abstract Data Type (ADT)
    - What is an ADT?
    - What is a Data Structure?
  - Examples on ADT:
    - Integer, Set, and List
  - Reference:
    - Textbook: Problem Solving with Algorithms and Data Structures
      - □ Chapter 3: Basic Data Structures
    - Data Abstraction & Problem Solving with Java
      - □ Chapter 4 – Data Abstraction: The Walls

12.1 Some Software design Principles

Modularity

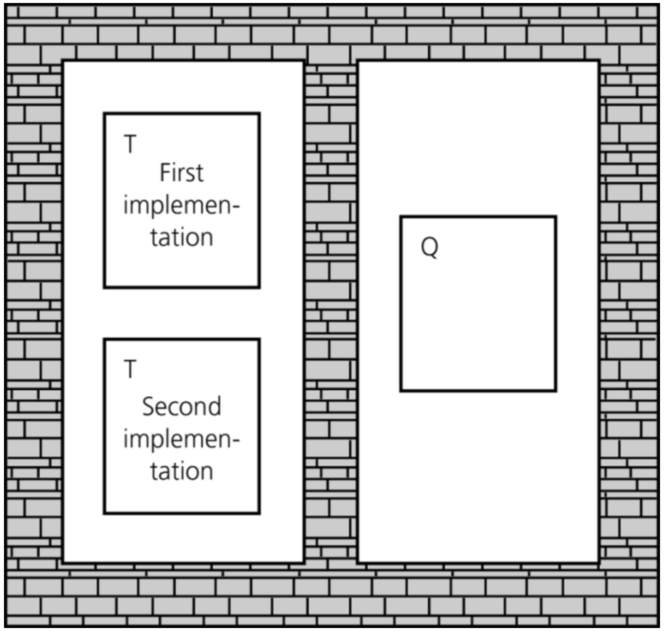
- Modularity (divide a program into manageable parts)
  - Keeps the complexity of a large program manageable
  - Isolates errors
  - Eliminates redundancies
  - Encourages reuse (write libraries)
  - A modular program is
    - Easier to write / Easier to read / Easier to modify


Aagarghh!
It crashed.
"#!ERROR"?
F17 = F18 + Sheet 17! AA17 + Sheet 93! D7: G23 + IF(F19 = F18, IF(F19 = F19, SUM(Sheet5! E28)×(23-9071), VLOOKUP(Sheet3! G29, "tag", A23: A24), n), 0) / 3972-781320 + Sheet 42 = ????
TRASH



Information Hiding

- Hides certain implementation details within a module
- Makes these details inaccessible from outside the module
- Isolate the implementation details of a module from other modules

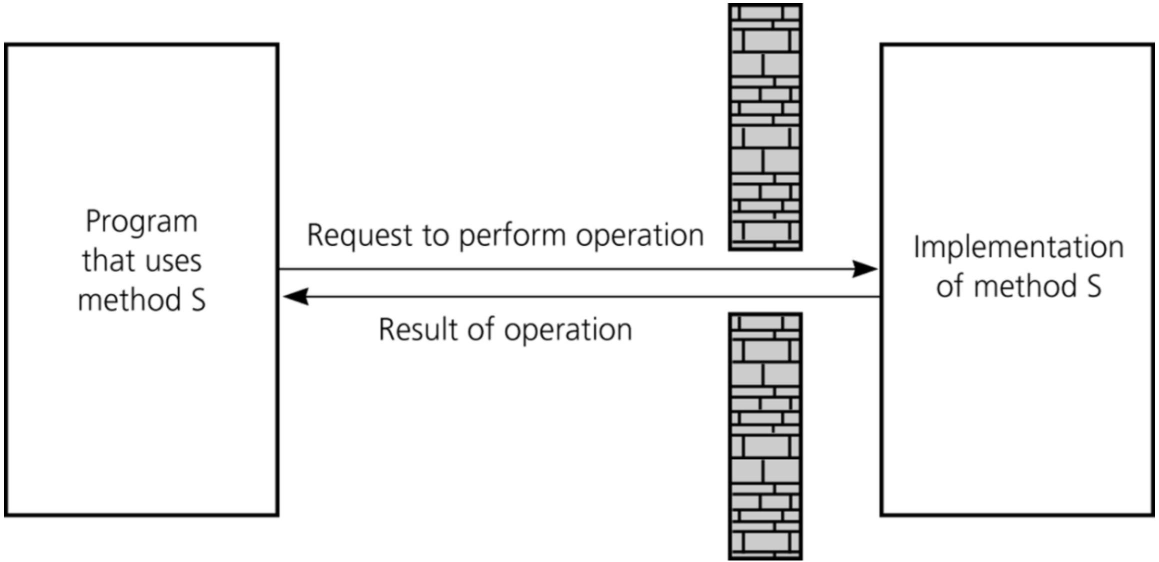


Isolated tasks: the implementation of task *T* does not affect task *Q*

12.1 Some Software design Principles

Isolation of Modules

- The isolation of modules is not total (otherwise module would be useless)
  - Methods' specifications, or contracts, govern how they interact with each other
  - Similar to having a wall hiding details, but being able to access through "hole in the wall"


Program that uses method S
Request to perform operation
Result of operation
Implementation of method S

Abstraction

- What does 'abstract' mean?
  - From Latin: to 'pull out'—the essentials
    - To defer or hide the details
    - Abstraction emphasizes essentials and defers the details, making engineering artifacts easier to use
  - Example:
    - I don't need a mechanic's understanding of what's under a car's hood in order to drive it
      - □ What's the car's interface?
      - □ What's the implementation?

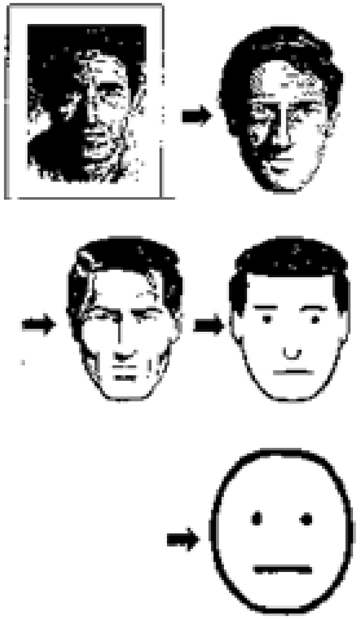
Abstraction

- Abstraction
  - The principle of ignoring those aspects of a subject that are not relevant to the current purpose in order to concentrate solely on those aspects which are **relevant**
- How do we achieve:
  - Modularity
  - Information hiding
  - Isolation of modules
  - i.e. The abstraction of **what** from the **how**

12.1 Some Software design Principles

Abstraction

- Separates the purpose and use of a module from its implementation
- A module's specifications should
  - Detail how the module behaves
  - Identify details that can be hidden within the module
- Advantages
  - Hides details (easier to use)
  - Allows us to think about the general framework (overall solution) & postpone details for later
  - Can easily replace implementations by better ones



Data and Operations

- What do we need in order to achieve the above Software Engineering design principles?
  - What are the typical operations on data? (example: database)
    - Add data to the database
    - Remove data from the database
    - Find data (or determine that it is not in the data base)
      - □ Ask questions about the data in a data collection (e.g. how many CS105 students do stage 1 Math courses?)
- Question:
  - Do we need to know what data structures used?
    - No, better make implementation independent of it!

Data and Operations

- Asks you to think **what** you can do to a collection of data **independently** of **how** you do it
- Allows you to develop each data structure in relative isolation from the rest of the solution
- A natural extension of procedural abstraction

```
CAR:
Start()
Stop()
TurnLeft()
TurnRight()
```

```
CAR:
Start(){
  Select first gear
  Release parking brake
  Bring clutch up to the
     friction point
  Press gas pedal
  Release clutch
}
```

NOTE: Implementation can be different for different cars, e.g. automatic transmission

Abstract Data Types

- An Abstract Data Types (ADT) is composed of
  - A collection of data
  - A set of operations on that data
- Specifications of an ADT indicate What the ADT operations do, not how to implement them
- Implementation of an ADT
  - Includes choosing a particular data structure

Properties and Behaviors of the ADT

- The properties of the ADT are described by the collection of data
  - The data can be in terms of simple data types of complex data types
  - Simple data types
    - Integer
    - Floating point
    - Character
  - Complex data types
    - Multimedia

Brand: BMW

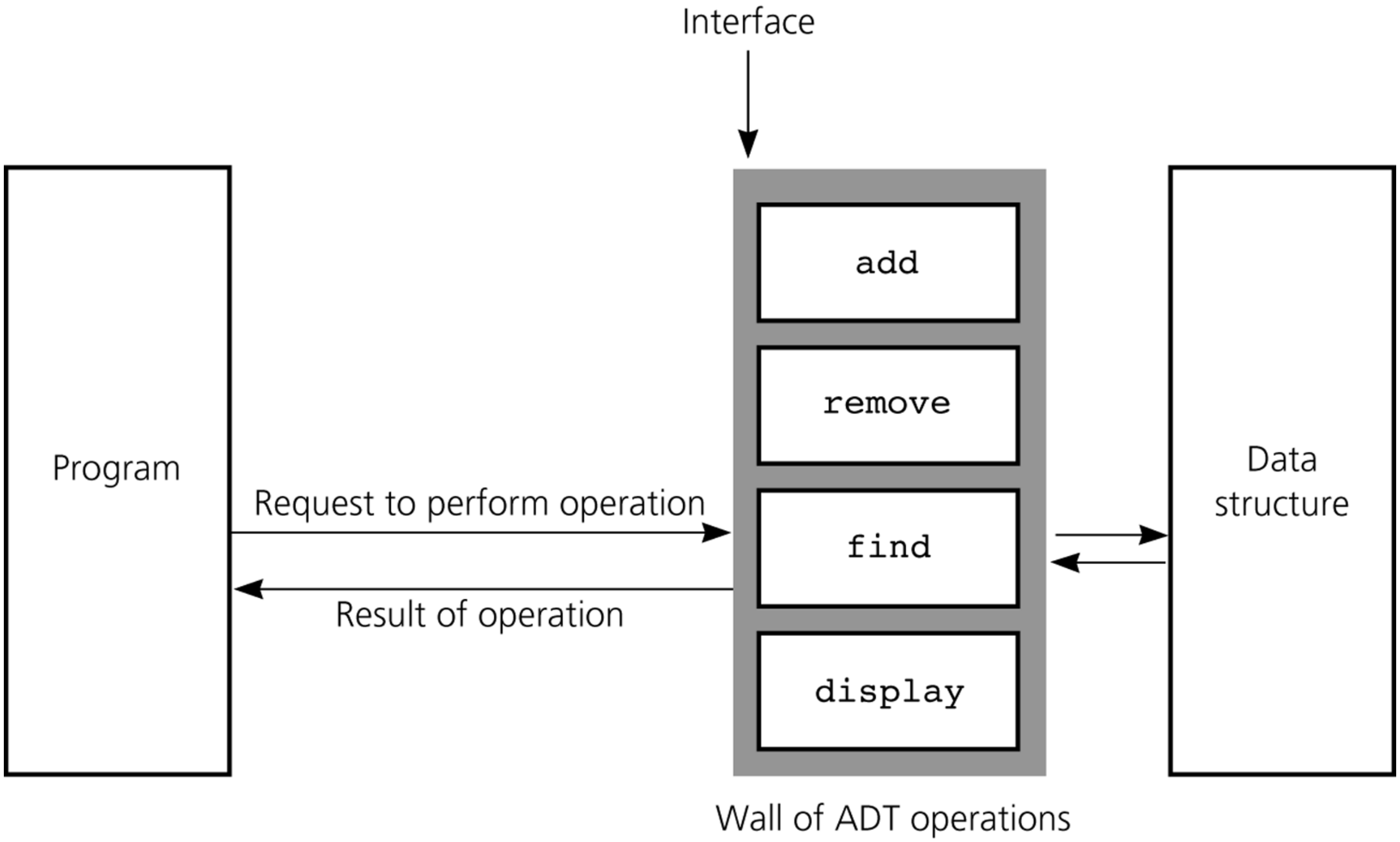
- The behaviors of the ADT are its operations or functions

Data structure Vs ADT

- An ADT is not the same with a Data Structure
- Data structure
  - A construct that is defined within a programming language to store a collection of data
  - Example: arrays
- ADT provides "data abstraction"
  - Results in a wall of ADT operations between data structures and the program that accesses the data within these data structures
  - Results in the data structure being hidden
  - Can access data using ADT operations
  - Can change data structure without changing functionality of methods accessing it

Abstract Data Types

- A wall of ADT operations isolates a data structure from the program that uses it



Disadvantages & Advantages

- Disadvantages of Using ADTs
  - Initially, there is more to consider
    - Design issues
    - Code to write and maintain
    - Overhead of calling a method to access ADT information
    - Greater initial time investment
- Advantages of Using ADTs
  - A client (the application using the ADT) doesn't need to know about the implementation
  - Maintenance of the application is easier
  - The programmer can focus on problem solving and not worry about the implementation

Designing an ADT

- An abstract data type (ADT) is a specification of a set of **data** and the set of **operations** that can be performed on the data
- Such a data type is **abstract** in the sense that it is **independent** of various concrete implementations
- Questions to ask when designing an ADT
  - What data does a problem require?
  - What operations does a problem require?
  - Examples:
    - Integers, floating-point
    - Sets
    - Lists
    - Stacks, Queues, Trees …

Integers

- Data
  - Containing the positive and negative whole numbers and 0
- Operations which manipulate the data:
  - Such as addition, subtraction, multiplication, equality comparison, and order comparison
  - Methods for data conversion, output etc.

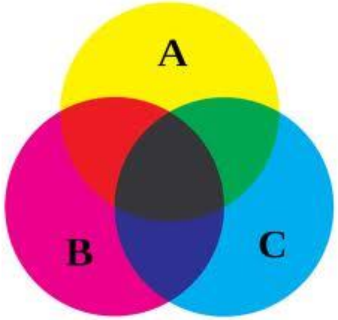
12.3 ADT Examples

Floating-point

- Data
  - Containing the whole possible positive and negative values
  - Precision is limited by number of digits
- Operations which manipulate the data:
  - Such as addition, subtraction, multiplication, equality comparison, and order comparison
  - Complex mathematical functions such as exponential, logarithm, triangular functions etc.
  - Roundings
  - Methods for data conversion, output etc.

Sets

- Data
  - An unordered collection of unique elements
- Operations which manipulate the data:
  - **add**
    - Add an element to a set
  - **remove**
    - Remove an element from the set,
  - Others
    - Get the union, intersection, complement of two Set objects

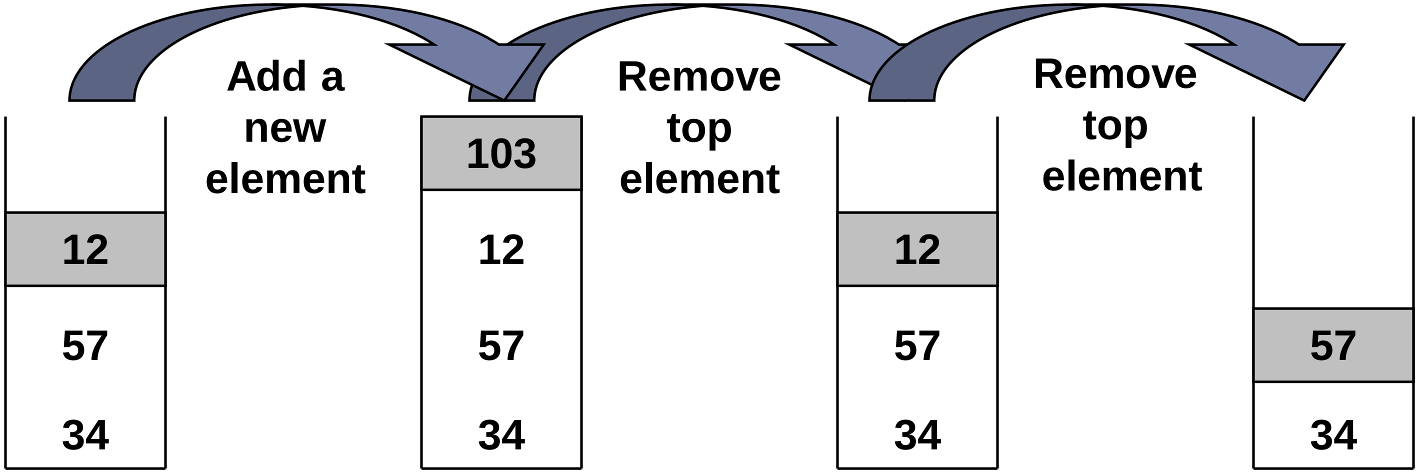


Linear Structures

- Linear structures are data collections whose items are **ordered** depending on how they are **added** or **removed** from the structure
- Once an item is **added**, it stays in that **position** relative to the other elements that came before and came after it
- Linear structures can be thought of as having two **ends**, **top** and **bottom**, (or front and end or front and back)
- What distinguishes one linear structure from another is the way in which items are added and removed, in particular the location where these additions and removals occur, e.g., add only to one end, add to both, etc.

Linear Structures

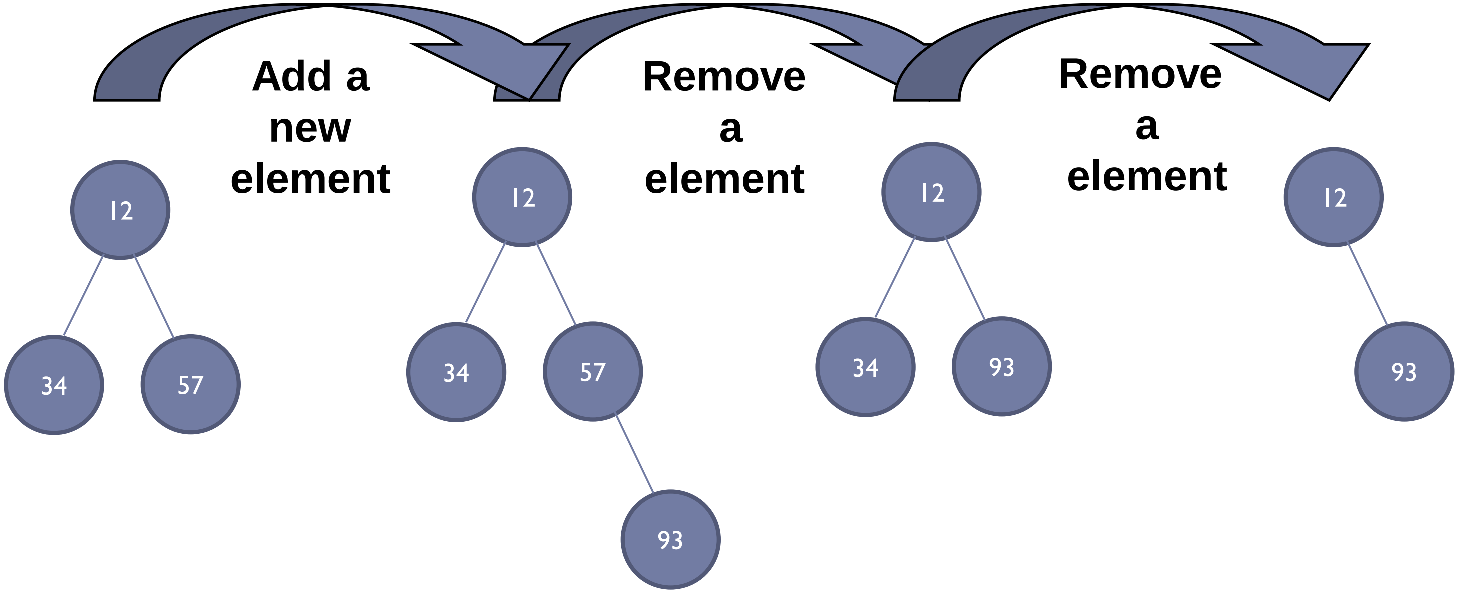
- Examples: Stack, Queue, Linked-list, etc.



12.3 ADT Examples

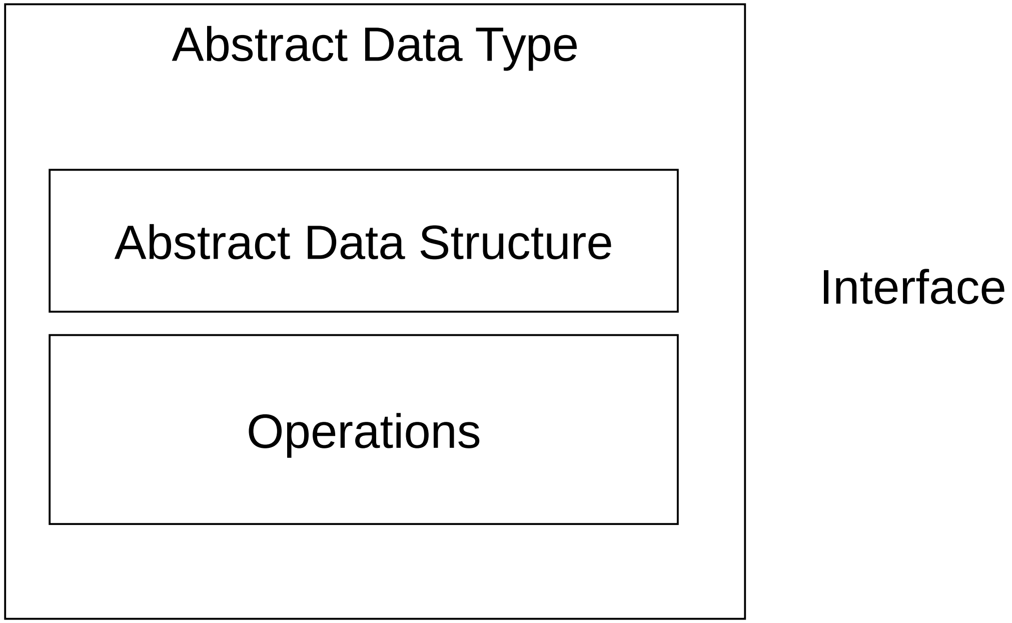
Non-Linear Structures

- Every data item is attached to several other data items in a way that is specific for reflecting relationships
- The data items are not arranged in a sequential structure
- Examples: Tree, Graph, Heap, etc.



Summary

- Solving a problem becomes easier by considering only relevant data and operations and ignoring the implementations (“abstraction”)



Summary

- Abstract Data Types (ADTs) enable you to think –
  - **What** you can do with the data independently of **how** you do it
- An abstract data type can be accessed by a limited number of **public methods** –
  - They are often defined by using an interface
- The implementation of the ADT (data structures and algorithms) can be changed without influencing the behavior of the ADT