Abstract Data Types

ADT

Haitham A. El-Ghareeb

October 8, 2019

Faculty of Computers and Information Sciences
Mansoura University
Egypt
helghareeb@mans.edu.eg

Contacts

- https://www.haitham.ws
- https://youtube.com/helghareeb
- https://www.github.com/helghareeb
- http://eg.linkedin.com/in/helghareeb
- helghareeb@mans.edu.eg

Introduction

Introduction

This course emphasizes three important concepts in computer science

- Algorithms
- Data Structures
- Abstractions

Introduction

Algorithm

Algorithm

- A sequence of clear and precise step-by-step instructions for solving a problem in a finite amount of time
- The foundation of computer science is based on the study of altorighms

Introduction

Programming

Programming

Algorithms are implemented by translating the steps into a computer program

- Computer Programming
  - the translation process
- Programming Language
  - used to construct a computer program
  - should be appropriate for the given problem

Introduction

Data Types

Data Types

Data is stored in a computer as a sequence of binary digits

- Type
  - a collection of values
  - ex: numeric
- Data Type
  - a given type along with a collection of operations
  - ex: integers

Data Types Groups

Data types can be divided into two groups:

- Simple
  - consists of single values
  - ex: integers, floating-points
- Complex (or composite)
  - multiple components
  - ex: lists, tuples, strings

Data Types - Definition

Data types can also be characterized by their definition

- Primitive Types
  - Provided by the language itself
  - ex: int, float, list, string
- User-Defined Types
  - defined by the programmer as needed
  - class defitnitions create new data types
  - ex: social security number, student record

Abstractions

Abstractions

Used by computer scientists to help manage complex problems

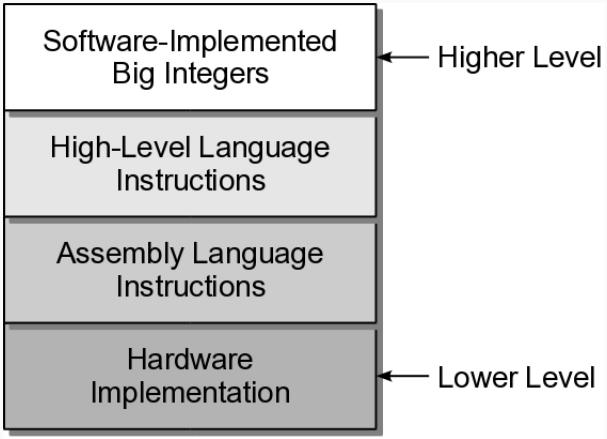
- Abstraction
  - a mechanism for separating the properties of an object
  - restricting the focus to those relevant in the current context
- Focus on the 'what' not the 'how'

Abstraction Types

Common types in Computer Science

- Procedural Abstraction
  - use of a function/method knowing what it does but not how it is done
  - ex: y = sqrt(x)
- Data Abstraction
  - separate the values/operations from the implementation of a data type
  - ex: big integers in Python

Multiple Layers of Abstractions



ADT Specifications

- A programmer-defined data type specified by
  - a domain or set of data values
  - a collection of well defined operations
- Defined independent of its implementation

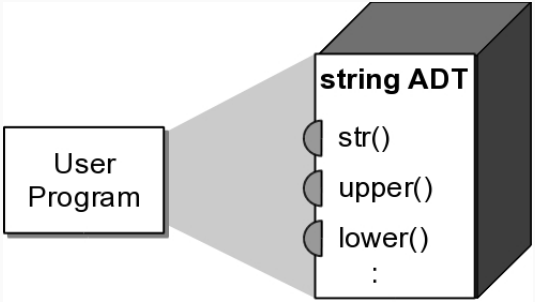
Abstractions

Information Hiding

Information Hiding

ADTs can be viewed as black boxes

- functionality is provided through an **interface**
- implementation details are hidden inside the box



Types of Operations

ADT operations can be grouped into four categories

- constructors
- accessors
- mutators
- iterators

ADT Implementation

- Abstractions make problem solving easier
- Programs require concrete implementations in order to execute
  - language library ADTs
    - implemented by the library maintainers
  - your own ADTs
    - implemented by you

ADT Categories based on Implementation

- Simple ADT
  - composed of simple individual parts
  - ex: Cartesian coordinates, dates, rational numbres
- Complex ADT
  - composed of a collection or group of values
  - ex: list, dictionary, set

Data Structure

Data Structure

Complex abstract data types are implemented using a **Data Structure**

- Physical representation of how data is stored, organized, and manipulated
- Store a collection of values

Common Data Structures

- Many common Data Structures
  - Arrays, Linked Lists, Stackes, Queues, Trees, Graphs
- Differ in
  - Data Organization
  - Available Operations
- Choice of Data Structure depends on
  - the specific ADT
  - the problem being solved

Advantages

Several advantages of working with ADTs

- Focus on solving the problem at hand
- Help to reduce logical errors from misuse of the data type
- Can easily change the implementation w/o affecting the use of the ADT
- Easier to manage and divide larger problems into smaller problems

Collection - Container - Element

Some terms used in Computer Science can have different meanings

Collection - Container - Element

Some terms used in Computer Science can have different meanings

- Collection: Group of values with no implied organization or relationship

Collection - Container - Element

Some terms used in Computer Science can have different meanings

- Collection: Group of values with no implied organization or relationship
- Container: Data Structure or ADT that stores a collection. Organization and operations can vary.

Collection - Container - Element

Some terms used in Computer Science can have different meanings

- Collection: Group of values with no implied organization or relationship
- Container: Data Structure or ADT that stores a collection. Organization and operations can vary.
- Element: an individual item or value stored in a container.

Sequence - Sorted Sequence

- Sequence
  - a container with elements arranged in linear order
  - elements can be accessed by index position
  - ex: list, tuple
- Sorted Sequence
  - a sequence in which the elemnts are arranged based on a relationship between the elements
  - ex: list sorted by name

List

- The term **List** commonly refers to any collection with linear ordering such that
  - every elemnt, but the first, has a unique successor
  - every element, but the last, has a unique predecessor
- A sequence is a list, but a list is not necessarily a sequence
- it depends on the list implementation and the method to access the elemnts

Python

Python List

- Python uses the term list for its primitive mutable sequence type
- To avoid confusion, we will use the terms
  - list
    - to refer to Python's list Structure
  - General List of List Structure
    - to refer to the more general list structure as defined before

Python Classes

- Defines and implement ADTs using Python Classes
- Advantages
  - Use Python's simple syntax
  - Easily translated to other languages
  - Provide an interface to hide the implementation
  - Allow for **Encapsulation**
    - Specification of the data and operations combined in a single definition and implementation

Using the ADT

Using the ADT

- Given the definition, we can use the ADT witout knowing how it is implemented
- Reinforces the use of Abstraction
  - by focusing on what functionality is provided
  - instead of how that functionality is implemented

Defining Operations

The ADT definition should specify

- required inputs and resulting outputs
- state of the ADT instance before and after the operation is performed

Precondition

Condition or state of the ADT instance and data inputs before the operation is performed

- Assumed to be true
- Error occurs if the condition list is not satisfied
  - ex: index out or range
- Implied Conditions
  - the ADT instance has been created and initialized
  - valid input types

Postcondition

Result or state of the ADT instance after the operation is performed

- Will be true if the preconditions are met
  - given: x.pop(i)
  - them i^{th} item will be removed if i is a valid index

Postcondition Depends on

Type of Operation

- Access methods and iterators
  - no Postcondition
- Constructors
  - create and initialize ADT instances
- Mutators
  - the ADT instance is modified in a specific way

Python

Exceptions

- OOP languages raise Exceptions when errors occur
  - An event that can be triggered by the program
  - Optionally handled during execution
- Example

```
myList = [ 12, 50, 5, 17 ]
print( myList[4] )

Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Assertions

- Used to state what we assume to be true

```
assert value != 0, 'Value can\'t be Zero'
```

- If condition is false, a special exception is automatically raised
  - Combines condition testing and raising an exception
  - Exception can be caught or let the program abort

Protected Members

Python does not provide for a technique to protect attributes and methods from direct access

- We use identifiers beginning with an underscore
- User shall not attempt direct access

```
self._no_faluts = 0
```

Helper Methods

- Methods used internally to implement the class
  - Allow for the subdivision of larger methods
  - Help to reduce code repetition
- Not meant to be accessed from the outside

```
            self.is_valid(a,b,c)
```

Overloading Operators

We can implement methods to define many of Python's standard Operators

- Allows for more natural use of objects
- Limit use of operator methods for meaningful purposes

```
# __eq__, __lt__, __le__ # 3 of 6 comparable Operators
# __add__, __sub__, __mul__, __div__
```

Bag ADT

Bags

- A bag is a basic container like a shopping bag that can be used to store collections
- There are several variations
  - simple bag
  - grab bag
  - counting bag

Bag ADT

- A simple bag is a container that stores a collection with duplicate values allowed. The elements
  - are stored individually
  - have no particular order
  - must be comparable
- Bag ADT
  - Bag()
  - length()
  - contains(item)
  - add(item)
  - remove(item)
  - iterator()

Bag: Example

```
# Creates a bag and fills it with values. The user is then
# prompted to guess a value contained in the bag.

myBag = Bag()
myBag.add( 19 )
myBag.add( 74 )
myBag.add( 23 )
myBag.add( 19 )
myBag.add( 12 )

value = int( input("Guess a value contained in the bag.") )
if value in myBag:
    print( "The bag contains the value", value )
else :
    print( "The bag does not contain the value", value )
```

Why a Bag ADT?

- Python's list can accomplish the same thing as a Bag ADT
- So, why do we need a new ADT?
  - For a small program, the use of a list may be appropriate
  - For large programs the use of new ADTs provide several advantages

Why a Bag ADT?

By working with the abstraction of a bag, we can

- Focus on solving the problem at hand instead of how the list will be used
- Reduce the cange of errors or misuse of the list
- Provide better coordination between different modules and programmers
- Easily swap out one Bag implementation for a possibly more efficient version

Implementing the Bag

- Implementation of a complex ADT typcially requires the use of a data structure
- There are many data structures (and other ADTs) from which to choose
- Which should we use?

Evaluating a Data Structure

- Evaluate the Data Structure based on certain criteria
- Does the data structure
  - Provide for the storage requirements of the ADT?
  - Provide the necessary functionality to fully implement the ADT?
  - Lend itself to an efficient implementation of the operations?

Selecting a Data Structure

Multiple Data Structures may be suitable for a given ADT

- Select the best possible based on the context in which the ADT will be used
- Common for language libraries to provide multiple implementations of a single ADT

Bag ADT Data Structure

Evaluate each DS/ADT option to determine if it can be used for the Bag

- Dictionary
- List

Evaluating Candidate Data Structure - Storage

Provide for the storage requirements of the ADT?

- Dictionary
  - stores key/value pairs; key must be unique
  - can store duplicates (using a counter as the value)
  - can not store each item individually
- List
  - can store any type of comparable object
  - can store duplicates
  - can store each item individually

Evaluating Candidate Data Structure - Functionality

Does the List provide the necessary functionality to fully implement the ADT?

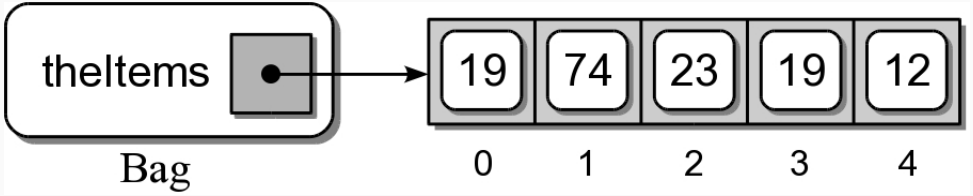
- Empty bag - Empty list
- Bag size - List size
- Contains item - use in operator on list
- Add item - `append()` to the list
- Remove item - `remove()` from the list
- Traverse items - can access each list elements

Selecting the List

The Python list can be used to implement the Bag

- Provides for the storage requirements
- Provides the necessary functionality

Sample Bag Instance



Traversals and Iterators

Traversals are very common operations performed on containers

- Iterates over the entire collection
- Provides access to each individual element
- Examples
  - find an item
  - print the entire collection

Bag Implementation

bag.py

Summary

Summary

- ADT
- Choose the appropriate Data Structure
- Iterator

https://www.haitham.ws