Abstraction

one definition:
Generalization; ignoring or hiding details to capture some kind of commonality between different instances.

Source:

Levels of Abstraction: Computer

- Programs
- Code libraries/operating system
- High-level language
- Virtual machine/compiler
- Assembly language
- Computer architecture

C++ Standard Libraries

- Provide:
  - Functional abstractions (e.g., math functions)
  - Structural abstractions (data types)
  - Operating system/computer resources (storage, network, I/O)
- Two facets of a library:
  - Interface
  - Implementation

Interfaces

Interface:
- The user-facing part of the library
- The templates/classes/functions available
  - Public parts only of classes and templates
  - Implicitly includes documentation – how do I use it?

The interface hides the complexity of the underlying implementation (how does sqrt work?)

Interface Illustrated

Consider a generic car:
- Steering wheel
- Accelerator
- Brake pedal
- Gear shift (and maybe clutch)
- Mirrors
These form the car’s interface.

Implementation: varies by make, model, year

If you know how to drive, you can probably drive any car (ignoring automatic vs. manual) because you know how to use the standard interface.
1/14/2018

Levels of Abstraction: Data

- Abstract Data Types
- Concrete Data Types
- Storage (bits & bytes)

Example: Integer

- Abstract Data Type
  - Domain: positive and negative integers
    - Max, min values may be bounded
  - Interface: +, -, *, /, =, etc.
- Concrete data type:
  - E.g., int
  - Implementation: in compiler
- Storage:
  - 1 word = 4 bytes = 32 bits
  - 2’s complement representation (CSCI 341, others)

Abstract Data Type

- Defines a domain of values for the type
- Specifies a general interface for a type
  - Primarily specifies behaviors
  - Can also specify properties
  - May specify performance characteristics
- Implementations allowed to vary
  - Generally hidden
  - Generally irrelevant (except when not – RTFM*)

*Read The Fine Manual

Containers

Structures which contain collections of objects:
- Vectors/Lists
- Stacks
- Queues
- Sets
- Maps

We will study all of these container types!

Why Study Containers

- They are incredibly useful:
  - Data naturally occur in collections
  - Key to many if not most important applications
    - Spreadsheets, databases
    - Signal processing/compression/cryptography
    - MapReduce (Google)
  - ...
- They are instructive:
  - Good examples of ADTs
  - (Relatively) easy to understand and program
  - Good models for complexity analysis
Example: Vector

- Generalization of an array
- Sequential collection of data
- Random access
  - Access items by index
  - Access operations are constant time
- Principal operations
  - Add, insert, remove
  - Get, set at a particular index
  - Get size

Standard Template Library Vector

```cpp
#include <vector>

template <class T> class vector

Operations:
- push_back(value) // add value to end
- insert(position, value) // insert value before the specified iterator
- erase(position) // remove value at specified iterator
- at(index) // access (get/ set) value at specified index
- operator[](index) // access (get/ set) value at specified index
- size() // get size
- empty() // true if no elements
- clear() // remove all elements
- ...
```

Up Next

- Read Sections 11.1 – 11.6
- Friday, January 19
  - Lab 2 – I/O
  - APT 1 due
  - Project 1: Image Editor assigned
- Monday, January 22
  - Stacks & Queues