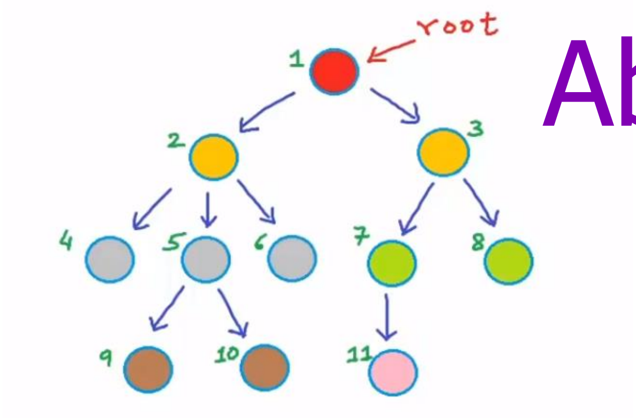
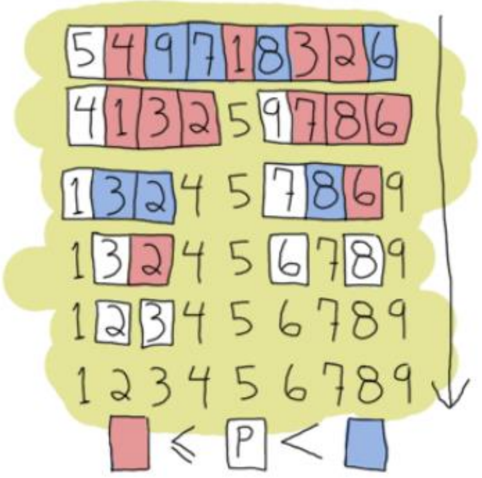
ECE 250 Data Structures & Algorithms



Abstract Data Types



Ziqiang Patrick Huang

Electrical and Computer Engineering

University of Waterloo

WATERLOO | ENGINEERING

Admin

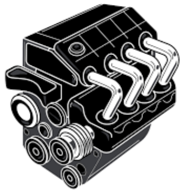
- Makeup Lecture next week
  - Friday 11:30 (section 1), Monday 3:30 (section 2)
  - One of the TA Ahmad will give a guest lecture on "How the contents of this course can be used to create an underwater positioning system. We will create an algorithm to detect and identify signals from acoustic waves that is efficient enough to run on a microprocessor. We'll discuss trade offs between speed (time), space (memory), and accuracy."

- Lab Sessions next week
  - Come ready to code and ask questions
  - Aim to make significant progress (like ~80%) during the lab sessions

Big Idea: Abstraction

- Abstraction: To take a relative complex system and simply it for use

Example 1: Driving a car



Example 2: Software-hardware stack (abstraction layers)

Application
Algorithm
Programming Language
Operating System
Compiler
Instruction Set Architecture
Microarchitecture
Register Transfer Level
Circuits
Devices
Physics

Abstraction in Data Structures

- Abstraction: separation of interface from implementation

- Interface = Abstract Data Type (ADT)
  - What operations can we perform on the structure?
  - Says nothing about how we perform those operations

- Implementation = Specific Data Structure
  - How do we make those operations happen?
  - Could have many implementations
  - Will determine the efficiency of the operations

Similarly …

What instructions the computer can execute

Instruction Set Architecture
Microarchitecture

How the computer execute those instructions

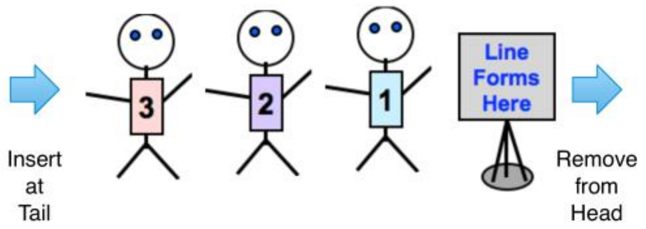
More Reasons for Abstraction

- Makes it easy to divide work
  - One person write algorithms that use the data structures
  - Another person write the data structures themselves
  - Only need to know other people's interface not implementation

- Makes it easy to change implementations
  - Start with a simple implementation that is correct but slow
  - While debugging, could have a "debug implementation"
  - Finalize with a more sophisticated and efficient implementation
  - Can switch back and forth easily

ADT Example 1: Queues

- Queue: first-in first-out (FIFO) sequence of items
  - Primary operations: *enqueue* and *dequeue*
  - FIFO → items returned by *dequeue* in the same order as placed by *enqueue*
  - May support other operations:
    - Testing if the queue is empty
    - Obtaining a count of how many items are in the queue
    - Peeking at the next item (seeing what it is without removing it)



- Uses of queues
  - Hardware: instruction queues in processors, packet queues in networks, etc.
  - Software: task scheduling in OS, Breadth-First Search in Graphs, etc
  - A natural choice to ensure fairness

Queues: ADT Definition

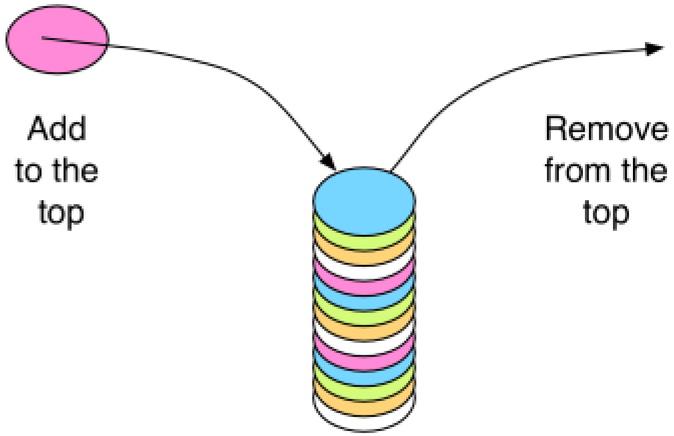
```
template <typename T>
class Queue {
        void enqueue(const T & item);
        T dequeue();  //might choose to return void instead
        T & peek();
        const T & peek() const;
        int numItems() const;
};
```

C++'s STL has a std::queue class, with different function names (e.g., push/pop)

WATERLOO | ENGINEERING

ADT Example 2: Stacks

- Stack: last-in first-out (LIFO) sequence of items
  - Primary operations: *push* and *pop*
  - LIFO → *pop* returns the most recently *pushed* item
    - E.g., push 1, then 2, then 3, and then pop → get 3
  - May support other operations as well



- Use of Stacks
  - Function call stack
  - Reverse any sequence of items
  - Nested matching
    - E.g., (4 + (3 * 2) – (8 * 9) + 1)

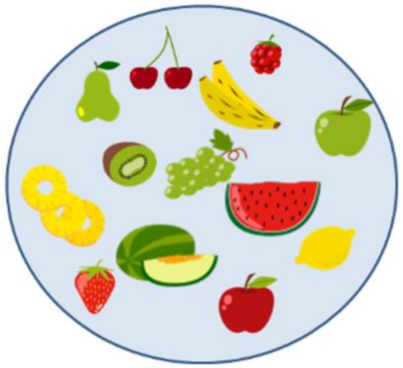
Stacks: ADT Definition

```
template <typename T>
class Stack {
        void push(const T & item);
        T pop();  //might choose to return void instead
        T & peek();
        const T & peek() const;
        int numItems() const;
};
```

C++'s STL has a std::stack class

ADT Example 3: Sets

- Set: a collection of elements
  - Support operations similar to those found on a mathematical set:
    - Adding items, testing if an item is in the set, checking if the set is empty
    - Union, intersection
  - Variant: multiset(bag), allows same element to appear multiple times
  - Limit to finite set → do not allow certain operations (e.g., complement)



- Use of Sets
  - Any time we want to track what items belong to a particular group
    - In a word-based game: track set of valid words
    - In task scheduling: track whether sets of hardware resources intersect

Sets: ADT Definition

```
template <typename T>
class Set {
        void add(const T & item);
        bool contains(const T & item) const;
        int numItems() const;
        Set<T> intersect(const Set<T> & s) const;
        Set<T> unionSets(const Set<T> & s) const;
};
```

C++’s STL has a std::set class

ADT Example 4: Maps

- Map: tracks a mapping from *keys* to *values*
  - Primary operations: add, update, lookup, etc.
  - Multiple items with the same key not allowed
  - Variant: multimap, allows items with duplicate keys
    - E.g., "Avenger" → "Iron Man"; "Avenger" → "Captain America"
    - Now what happens if we do: lookup("Avenger")?
      - Option 1: return a list of items
      - Option 2: return an iterator to a single item

- Use of maps
  - Any time we want to associate one piece of info with some other info
  - E.g., social networking site: user ID → a list of IDs of friends

Map: ADT Definition

```
template <typename K, typename V>
class Map {
        void add(const K & key, const V & value);
        const V& lookup(const K & key) const;
        V & lookup(const K & key);
        int numItems() const;
        void remove(const K & key);
};
```

If we want to have an iterator for our Map class, what (type) should the iterator return?

C++'s STL has a std::map class

Exercise For You: Manage Browser History

- How would you devise an algorithm to manage browser history, allowing users to navigate backward and forward?
  - Hint: one of the ADTs we introduced earlier is a good fit for this

Wrap Up

- In this lecture, we talked about:
  - Idea of abstraction why it is useful
  - Abstraction in the context of data structure: abstract data types
  - Four ADT examples: queues, stacks, sets, maps
    - Primary operations
    - Use cases
    - Typical definition in C++
- Next up
  - Review of Linked List
  - Implementations of some of the ADTs

Suggested Complimentary Readings

- Data Structure and Algorithms in C++: Chapter 3 (Leave the implementation parts for now)

Acknowledgement

- This slide builds on the hard work of the following amazing instructors:
  - Andrew Hilton (Duke)