

# For Wednesday

- Read Weiss, chapter 4, sections 1, 2, 6
- Homework
  - Complete the List Practice described on Blackboard under Notes and Homework. This is written work. Only turn in the requested pieces.

# Programming Assignment 1

- Any questions?

# Abstract Data Type

- A specification of a data type, including the operations of the data type
- Describes data items and operations in an **implementation-independent** way
- Can be implemented as a C++ class
- Could also be implemented as a set of variables and associated functions in C or another language

# Linear List ADT

- AbstractDataType *LinearList*{  
    instances (or data)  
        ordered finite collection of zero or more elements  
    operations  
        Create()  
        Destroy()  
        IsEmpty()  
        Length()  
        Find(index) // returns an element  
        Search(key) // returns an element  
        DeleteAt(index)  
        DeleteValue(key)  
        Insert(index,element)  
        Output()  
}

# Using an ADT

- ADTs are not directly usable
- They must be implemented
- Most ADTs can be implemented in more than one way
  - What would be different ways to represent the linear list ADT?

# Linked Lists

- What's the basic concept?

# What Is a Node?

- Two parts
  - Data
  - Pointer to the next one
- Why might you use a class instead of a struct to represent a node?

# Creating Nodes

- We'll always create nodes dynamically.
- Note that we almost never actually work with a node itself; we use pointers to the nodes.
- Important:
  - ALWAYS initialize next to NULL when a node is created unless you are immediately assigning it another meaningful value.

# C++ Details

- Deleting nodes . . .

# Linked List Variations

- Doubly-linked lists
- Empty head node

# Stacks

- What is a stack?
- What would a stack ADT look like?
- How could you implement a stack?
- What are stacks used for?

# Stack ADT

- **AbstractDataType** *Stack* {  
    **instances**  
        linear list of elements; one end is  
        the *top*  
    **operations**  
        Create()  
        Destroy()  
        IsEmpty()  
        IsFull() (not always needed)  
        Top()  
        Push(element)  
        Pop()  
}

# Queues

- What is a queue?
- What would a queue ADT look like?
- How could you implement a queue?
- What are queues used for?

# Queue ADT

- **AbstractDataType** *Queue* {  
    **instances**  
        linear list of elements; one end is  
            the *front*, the other the *rear*  
    **operations**  
        Create()  
        Destroy()  
        IsEmpty()  
        IsFull() (not always needed)  
        Front()  
        Add(elem) (or Put(elem) or Enqueue(elem))  
        Delete() (or Get() or Dequeue())  
}

# A Note on Push and Pop

- In this class, do **not** use push and pop to refer to queue operations.
- They should **only** apply to stack operations.

# Queues and Stacks

- Often used in similar contexts to achieve different desired results.
- Often used in conjunction with other data structures.