Overview

What is the Course About?
- Data Structures Intro
- Classes, Objects, and Pointers
- Linked Lists
Learning Objectives

• Improve your programming skills
  - Including complex data structures
  - Object oriented programming
  - Ability to conceptualize programming problems
• Know how to analyze the efficiency of an algorithm
  - How much time does it take as the size of the input grows
• Know how to prove the correctness of an algorithm

WebEx

• Please share your video, unless bandwidth is too slow
• Mute microphone except when talking
• Ask questions:
  - Can ask in the chat window
  - Or using the 'raise' hand feature
• Download the slides from website, find a tool to mark them up
  - Tablet might work well
Background Needed

• Intended to be taken earlier in your education
• Anyone admitted to the program
  - At a minimum, expected to have 1st year undergraduate CS major courses
• A course in discrete math also useful

Logistics

• Grading
  50% homework
  25% midterm
  25% final

• Course website: cslu.ohsu.edu/~heeman/cs532
Critique

- Answer key will be available to you in Sakai after you submit your answers.
  - You will have until Sunday at 11:59pm to submit a critique.
  - Explain what you did wrong, and why you made that mistake.
  - Worth up to half the marks that you lost.
    + Really good explanations might even get more.
  - Should show that you reviewed and understood answer key and understood whether your answer was correct.
  - See sample homework for how to format this.
C versus Python

- Could have taught this course in C
  - Pointers are built into the language
  - C does not have a built-in dictionary datatype
  - But few people know C, so would spend a lot of time teaching this
- Instead using Python
  - You should not use dictionaries or advanced packages
  - Use classes/objects to get pointers
  - Plus: get more in-depth experience with a programming language
  - Plus: get a solid foundation in object-oriented programming
  + Not in textbook

Academic Integrity

- You can do the homeworks with your colleagues
  - Document how you collaborated
  - e.g., worked on entire question together, versus clarified what the question meant
- After you have come up with a solution, redo it on your own
  - Nothing in, nothing out, wait at least an hour between
Overview

• What is the Course About?

⇒ Data Structures Intro

• Classes, Objects, and Pointers

• Linked Lists

Dynamic Sets: Overview

• From: III Data Structures of textbook

• Set: collection of elements (like integers)

• Dynamic sets
  - Can be manipulated by algorithms
  - Can grow, shrink, or otherwise change over time

• Different algorithms might require different operations
  - Many algorithms just need to insert, delete, and test membership
    + Dynamic set that supports this is called a dictionary
  - Some algorithms need to insert, and extract the smallest element
    + Min-priority queue
  - What operations are needed by an algorithm and how efficiently they need to be done determines what data structure type to use
Chapter 10: Elementary Data Structures

- Arrays
  - Elements are numbered starting at 0 (or 1)

- Stacks and Queues
  - For some tasks, you want to restrict how elements are accessed
  - Stack: last-in, first-out
    - Write an element to the stack, can read/pop most recent element added
    - When it is read/popped, no longer on stack
  - Queue: first-in, first-out
    - Can write an element to queue, can read/dequeue the oldest element
    - When it is read/dequeued, no longer in queue

Implementation for Stack

- Stack
  - Assume a maximum size of stack n
  - Use an array of size n S[0...n-1]
  - S[top] keeps track of the most recently added element
  - Start pushing elements starting at index 0
# Implementation for Queues

- **Queues**
  - Assume a maximum size of \( n \)
  - Use an array of size \( n S[0...n-1] \)
  - Enqueue: put on queue
  - Dequeue: take off of queue
  - Need two variables:
    + \( Q\text{-head} \) (next to be dequeued)
    + \( Q\text{-tail} \) (most recently added)

```python
def Stack-Empty(S):
    if S_top == -1:
        return True
    else:
        return False
def Push(S, x):
    S_top = S_top + 1
    S[S_top] = x
def Pop(S):
    if Stack-Empty(S):
        error "underflow"
    S_top = S_top - 1
    return S[S_top+1]
```

*pseudocode that looks like python*
Continued

• Does the amount of time to enqueue and dequeue take depend on the size of the queue?
• What is the empty queue?
• What is maximum number of elements that can be stored?
• Why not have Q.tail point at last element, rather than next space?

Code for Queues

def Enqueue(Q, x):
    Q[Q.tail] = x
    if Q.tail == Q.length:
        Q.tail = 1
    else:
        Q.tail = Q.tail + 1

def Dequeue(Q):
    x = Q[Q.head]
    if Q.head == Q.length:
        Q.head = 1
    else:
        Q.head = Q.head + 1
    return x

• Code assumes queue array starts at 1
Overview

- What is the Course About?
- Data Structures Intro
  ⇒ Classes, Objects, and Pointers
- Linked Lists

Groups of Information

- Will need more complex data structures
- Need data structure that can encapsulate pieces of information
  - like an street address, city, zipcode, phone number
- Elements of a dynamic set
  - Usually consist of a number of types of information
  - Usually one type is viewed as the key, used for searching, or sorting
  - Rest is viewed as satellite data
Reference other set elements

- Also, need a way to refer to other pieces of data
- Example: family tree
  - For each person, want to store name, date of birth, gender
  - Want to list who their parents are
    + Not just their names, but a way that we can access all the data about them

Python

- How can we do this in python?
  - Have an array for name, date of birth, gender
  - Each person has an unique index into arrays
  - Can have an array for father and mother to, which has the index for the
    father and mother of each person
- Kind of ugly
Python with Classes

- Python has classes
- Classes allow you to
  - Group data fields together as an object
  - Refer to an object with a constant or a variable
  - Include a var in the object that refers to another object
  - Associate functions on the group
- An instance of a group (or class) is called an object
- Will use classes in constructing complex data types

A Simple Class

class Person:
    def __init__(self, name, gender, birthday):
        self.birthday = birthday
        self.gender = gender
        self.name = name

d = Person("donald","male","1946-6-14")
m = Person("melania","female","1970-4-26")
b = Person("baron","male","2006-3-20")

print b.name

- class specifies how the class works
- Function __init__ specifies how to create a new object
  - Called by using the name of the class
  - Should initialize all of the object fields
- Object has 3 fields in it: birthday, gender, name
Referencing variables inside of an object

```python
class Person:
    def __init__(self, name, gender, birthday):
        self.birthday = birthday
        self.gender = gender
        self.name = name
        self.father = None
        self.mother = None
d = Person("donald","male","1946-6-14")
m = Person("melania","female","1970-4-26")
b = Person("baron","male","2006-3-20")
b.father = d
b.mother = m
print b.father.name
```

Pointers

```python
d = Person("donald","male","1946-6-14")
m = Person("melania","female","1970-4-26")
b = Person("baron","male","2006-3-20")
b.father = d
b.mother = m
e = d
e.name = "donnie"
print b.father.name
```

• d, m, b are not the actual objects, but pointers to them
Overview

• What is the Course About?
• Data Structures Intro
• Classes, Objects, and Pointers
  ⇒ Linked Lists

Operations on Dynamic Sets

S is a set, k is a key, x is a pointer to an element

• For any dynamic set:
  - Search(S,k) returns pointer to an element or None
  - Insert(S,x)
  - Delete(S,x)

• For totally ordered sets:
  - Minimum(S)
  - Maximum(S)
  - Successor(S,x)
  - Predecessor(S,x)
Linked List (Chapter 10.2)

- Linked list is a data structure
  - objects are arranged in a linear order
  - More flexible than an array as you can insert/delete objects faster
- Singly or doubly linked
  - Singly: elements point to next element
  - Doubly: also point to prev element
- Sorted or not
  - Sorted: min element is at head of list, next pointer points to next largest element
- Circular or not:
  - Circular: prev pointer of head element points to tail element, and next pointer of tail points to head
- Remainder of section: assume unsorted doubly linked

Operation

(a) $L.\text{head}$

(b) $L.\text{head}$

(c) $L.\text{head}$
Methods versus Functions

- Previous example had ListSearch as a function
  - Can work on any type of structure that has a head field, and the head points to something that has a next, and that thing has a next etc.
  - Works because Python is not tightly typed

- Alternative: make ListSearch a method
  - Method is defined in a class and operates on an object
  - Works because Python is not tightly typed
  - Method is defined in a class and operates on an object

```
class Element:
    def __init__(self, k):
        self.key = k
        self.next = None
        self.prev = None

class LinkedList:
    def __init__(self):
        self.head = None

    def ListSearch(L, k):
        x = L.head
        while x is not None and x.key != k:
            x = x.next
        return x

l = LinkedList()
a = Element(1); l.head = a
b = Element(4); a.next = b; b.prev = a
c = Element(16); b.next = c; c.prev = b
print ListSearch(l, 4).key
```
Methods allow you to associate functions on class objects directly with the definition of the class.

Makes it so that you can encapsulate code better so that only code inside of the class needs to know how the class works.

Forces you to think about what data you want, and what operations will work on it.

```python
class LinkedList:
    def __init__(self):
        self.head = None

    def Search(self, k):
        x = self.head
        while x is not None and x.key != k:
            x = x.next
        return x

lst = LinkedList()
...
print(lst.Search(4).key)
```
Inserting into a List

- Inserting at end of linkedlist: simpler, but need pointer to last element
  
  ```python
  l = LinkedList()
  a = Element(1); l.head = a
  b = Element(4); a.next = b; b.prev = a
  c = Element(16); b.next = c; c.prev = b
  ```

- Inserting at start of list: more complex but just need pointer to start of list
  
  ```python
  l = LinkedList()
  a = Element(1); l.head = a
  b = Element(4); a.prev = b; b.next = a; l.next = b
  c = Element(16); b.prev = c; c.next = b; l.next = c
  ```

- List Method: insert at start
  
  ```python
  def Insert(self,new):
      new.next = self.head
      if self.head is not None:
          self.head.prev = new
      self.head = new
      new.prev = None
  ```

Deleting from a linked List

- List Method:
  - This has to be method for linked list, not for element
  - It might have to update the head pointer
  - Boundary conditions force additional checks
  
  ```python
  def Delete(self,del):
      if del.prev is not None:
          del.prev.next = del.next
      else:
          self.head = del.next
      if del.next is not None:
          del.next.prev = del.prev
  ```
Sentinels

- To get rid of boundary case, have a sentinel
  - Makes this into a circular linked list
- Sentinel is a dummy object always in the linked list

```
(a) L.nil → L
       ↓    ↓
        9 16 4 1
     ↓    ↓    ↓
    L 16 4 1
```

```
(b) L.nil → L
       ↓    ↓
        9 16 4 1
     ↓    ↓    ↓
    L 16 4 1
```

```
(c) L.nil → 25 9 16 4 1
       ↓    ↓    ↓
    L 16 4 1
```

```
(d) L.nil → 25 9 16 4 1
       ↓    ↓    ↓
    L 16 4 1
```

Code with Sentinels

```python
def Delete(self, del):
    del.prev.next = del.next
    del.next.prev = del.prev

def Insert(self, new):
    new.next = self.nil.next
    self.nil.next.prev = new
    self.nil.next = new
    new.prev = self.nil
```

- Main point: logic is simpler rather than the slight gain in speed
Questions about Code

- Code for search is from textbook.
  - What does search return if item is not in the list?
    + How would you check for this?
  - What is a better option?
    + How would the code be changed?
    + How would you now check for not-found?

- How do we change code to store satellite data?
  - Where should satellite data be stored?
  - We do not want to make a new instance of List for each different type of satellite data we want to store
Class Inheritance

class Element:
    def __init__(self, k):
        self.key = k
        self.next = None
        self.prev = None

class MyEle(Element):
    def __init__(self, k, name, age):
        Element.__init__(self, k)
        self.name = name
        self.age = age

l = LinkedList(); l.Insert(MyEle(4, "chris", 35)); l.Insert(MyEle(8, "paul", 35))
print l.Search(8).name
print l.Search(5).name

• List uses Element class to define sentinel
  - Add entries using MyEle, which saves satellite data
  - List doesn’t care as long as entry has next, prev, and key fields

Unbounded Branching

• Parent cannot reference every child, like in a binary tree
• Define a class for nodes
  - left-child: points to left-most child (like a single linked list)
    + If no children, pointer is None
  - right-sibling: points to next sibling
    + If last sibling, pointer is None
  - parent

[Diagram of unbounded branching]