Overview of Computer Science

CSC 101 — Summer 2011
Programming Concepts
Data Structures and Algorithms
Lecture 10 — July 19, 2011

Announcements
• Lab #3 today
• Writing Assignment #4 due Thursday
• Midterm Friday (Lectures 1-11)
• Midterm review on Thursday
  – Come with questions

Objectives
• Compilation vs. interpretation
• General programming concepts
• Abstractions
  – A way to refer to something’s essence instead of its detail
• Data Structures
  – Ways to organize information in main memory
• How choice of algorithm can affect program performance
  – Comparison of some algorithms for searching and sorting data
Programming Concepts

- Programming:
  - Creating a *program* – a sequence of instructions to accomplish some task
- Some basic ideas have evolved to simplify creating programs

An Example Script

```html
<html>
<head>
<title>image script</title>
<script language="JavaScript">
var counter = 0
function changer() {
  counter = counter + 1
  if (counter > 9) { counter = 0 }
  document.images[0].src = counter + ".jpg"
}
</script>
</head>
<body>
<h1>Click on the image to change it.</h1>
<p>
<a onClick="changer()"> <img src="0.jpg"> </a>
</p>
</body>
</html>
```

- This is a short JavaScript program
- A program consists of *statements* which are *executed* sequentially
- A program usually needs some method to keep track of information – *variables*
An Example Script

```javascript
var counter = 0;
function changer() {
    counter = counter + 1;
    if (counter > 9) {
        counter = 0;
    }
    document.images[0].src = counter + " .jpg";
}
```

- In this script, `counter` is a variable
  - It is a name that refers to a place
    (an address) in main memory that holds a value

<table>
<thead>
<tr>
<th>Name</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter</td>
<td>0</td>
</tr>
</tbody>
</table>

Names, Values, And Variables

- We refer to types of things by generic names
- Names may have specific values
  - This kind of name refers a place to hold a value
  - In programming, we call this a variable

<table>
<thead>
<tr>
<th>Name</th>
<th>Current Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outside Temp</td>
<td>65.3°F</td>
</tr>
<tr>
<td>Fuel Remaining</td>
<td>9,231 kg</td>
</tr>
<tr>
<td>US President</td>
<td>Barack Obama</td>
</tr>
</tbody>
</table>

Layers of Abstraction

- An abstraction is a kind of simplification
  - A way to think of something without worrying about specific details
  - Consider an automobile
- Abstractions are an important part of computing
  - When you send an email, you don’t need to think about all the details of how it actually gets sent
- We will look at data types as abstractions
Data Abstractions

- Remember: everything in main memory is just bits – 1’s and 0’s
  - But, we can refer to many kinds of information
  - Text, pictures, numbers, music, etc.
- Software often must deal with large, changing data sets
  - Need to be able to handle changes quickly and easily
  - Need efficient ways of organizing and managing lots of data
- Data can be organized into data structures to help with that organization and management
  - Data structures deal with the arrangement of bytes of information in main memory
- Data structures are sometimes called abstract data types
  - Provide methods that applications use to deal with information without worrying about the details of where all the bits and bytes are in memory

Arrays

- Array – the simplest form of data structure
  - A series of the same type of items, stored in consecutive locations in memory
  - A one-dimensional array is a single series of items
  - A two-dimensional array is like a table
  - Higher dimensions are also possible

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$500</td>
<td>$550</td>
<td>$750</td>
</tr>
<tr>
<td>2nd Floor</td>
<td>$430</td>
<td>$400</td>
<td>$800</td>
</tr>
<tr>
<td>1st Floor</td>
<td>$550</td>
<td>$300</td>
<td>$300</td>
</tr>
</tbody>
</table>

- Arrays are static objects
  - Simple implementation
  - Can change the contents of any item in an array
  - But it’s difficult to add or delete array items
    - Would need to shift items up or down
    - The size of the array is fixed
  - All items must be the same kind of data
  - All numbers, or all text, etc.
Lists

- Lists are similar to one-dimensional arrays, but they are dynamic data structures
  - Can adapt to changing information needs
- Three main types of lists:
  - Linked list
  - Stack
  - Queue

Linked Lists

- A linked list
  - Used for a variable data set
  - Data need not be stored in consecutive memory addresses
  - Each data item in the list includes a pointer to the address of the next item in the list
  - Data can be easily added to or deleted from the list by changing pointers
  - A head pointer points to the first element in the list
  - The pointer in the last element is a special value, “NIL”, which doesn’t point to anything

Linked Lists

- The structure of a linked list
  - This list has three elements
Linked Lists

- Deleting an entry from a linked list

- Inserting an entry in a linked list

Stacks

- A stack is a list in which items are only added or removed from the head (top) of the list
  - Like a stack of dishes in a cafeteria
  - To add an item to the stack, push it onto the stack
  - To remove an item, pop it off the top of the stack
  - Stacks are also implemented through the use of pointers
  - A stack is a LIFO (last-in, first-out) data structure
Queues

- A queue is a list in which items are only added at the tail and removed from the head of the list
  - Like a line of people (a queue) waiting to buy concert tickets
  - When an item is dequeued
    - It’s removed from the front of the list by changing the head pointer to point to the next item in the queue
  - When an item is enqueued
    - The last item’s pointer is changed from NIL to point to the new item, and that new item gets a NIL pointer
  - A queue is a FIFO (first-in, first-out) data structure

Trees

- While linked lists, stacks and queues are all linear data structures, a tree is a data structure with a hierarchy
  - The single item at top is called the root node
  - A node with no children is called a leaf node
- A tree in which no parent has more than two children is called a binary tree

Binary Trees

- In a binary tree, each node has zero, one, or two children
  - Can be implemented using pointers
  - A binary tree can be used as a very efficient searching mechanism
    - Make sure the left child is always ‘before’ the parent and the right child is always ‘after’ the parent in the search order (an ordered tree)
Binary Trees

- An ordered tree

Binary Trees

- How a binary tree is created using pointers

Data Objects

- So, why would we want to search a tree for the letters of the alphabet?
  - Each node of the tree may have much more data associated with it
  - Each node of a binary tree can represent any type of data, as long as it can be sorted:
    - For an online catalog of books:
      - Author(s)
      - Title
      - ISBN
      - Publication date
      - Picture of the front cover
      - Reviews
      - Categories
      - Popularity ranking
      - Etc., etc., etc.
Application Design

- The five phases of software development:
  1. Analyze the task
  2. Develop a plan (algorithm) to accomplish the task
  3. Implement the plan
  4. Test and correct the implementation
  5. Maintenance
- The choice of algorithm can have a huge affect on how well the program performs

Algorithms

The English word algorithm is derived from the name 

محمد بن موسى الخوارزمی

Muḥammad ibn Mūsā al-Khwārizmī
- (c. 780–850 AD) Considered by some to be the “father of algebra”
- In about 820 AD, he wrote “Al-Kitāb al-mukhtasar fi hisab al-jabr wa’l-muqabala”
- “The Compendious Book on Calculation by Completion and Balancing”
- The word algebra comes from mispronouncing the title of his book

- An algorithm is a plan for accomplishing a task
- Example of an algorithm:
  Interest, compounded daily
  \[\text{amount} = \text{principal} \times (1 + \text{rate})^{\text{time}}\]
  where:
  - \text{amount} is the final value of the account
  - \text{principal} is the initial value of the account
  - \text{rate} is the daily percentage rate (\text{APR ÷ 365})
  - \text{time} is the number of \text{days} (\text{years} \times 365)
### Algorithms

- We can represent the compound interest problem in different ways:
  - **Pseudocode**

  ```plaintext
  begin
  input principal, APR, years
  rate = APR / 365
  time = years * 365
  amount = principal * (1 + rate)^time
  output amount
  end
  ```

- **Flowchart**

### Large Amounts of Data

- Many computer systems deal with large amounts of data
  - Retail systems
  - Banking
  - Government (taxes, social security, etc.)
  - Library catalogs
  - Music players
  - Search engines
  - Etc., etc., etc.
- To use the data, there must be fast and efficient ways to
  - Sort the data (alphabetize, etc.)
  - Search through the data

### Searching for an Item

- Consider an online bookstore with 1,000,000 titles available
  - A user types in the name of a book
  - The server needs to find that 1 book out of 1,000,000 – quickly
- The ‘obvious’ approach:
  - **sequential search** (also called linear search)
  - The server checks each book in the catalog to see if it’s the desired book
  - Consider performance
    - For 1,000,000 books this takes an average of 500,000 comparisons
    - Each comparison takes computer time – time that the user doesn’t want to wait for
Searching for an Item

• A ‘smarter’ method: binary search
  – Uses a divide and conquer approach
  – Break a big problem into smaller, easier problems

Pseudocode:
  • Start with an ordered (alphabetized) list
  • Look at the middle item in the list
    If it’s the right item, we’re done
  • Otherwise, is the item in the first half or second half?
    (i.e. is it alphabetically before or after the middle item?)
    • Repeat the search on the half of the list that contains the item

• Binary search on 1,000,000 items takes on average only 20 comparisons
  – Much better than 500,000 comparisons!

“Big-O” Notation

• Different algorithms to solve a problem may have very different performance
  – Different algorithms have different complexities
  – “Big-O” notation is sometimes used to describe the complexity of an algorithm

• The sequential search algorithm is $O(n)$
  – If we have 1000 times as many books, the algorithm will take about 1000 times as long

• The binary search algorithm is $O(\log n)$
  – If we have 1000 times as many books, the algorithm will take only about 10 times as long
  – If we have 1,000,000 times as many books, the algorithm will take only about 20 times as long
  – Much better for big jobs!

Sorting a List of Data

• For an algorithm like binary search to work, the data must already be in order (sorted)

• Sorting is a harder job than searching
  – Need a good sorting algorithm
    • We can compare sorting algorithms based on speed and on efficiency of data storage space
      • Several sorting algorithms exist, we’ll consider two
  – Assume you had to sort a pile of papers
    • How would you accomplish this task?
Sorting a List of Data

• An easy sorting method: selection sort
  – To alphabetize a list of names (or a stack of papers):
    • Find the name that comes first alphabetically
    • Exchange that name with the name at the top of the list
    • Repeat for the remainder of the list

• Example of selection sort
  (Sorted items are shaded in pink)

• Can take up to 1,000,000 swaps for 1,000,000 items
  • And each swap requires a sequential search of up to 1,000,000 comparisons!
  • This is an \( \Theta(n^2) \) algorithm \( \sim 1,000,000,000,000 \) operations required

Sorting a List of Data

• A faster method: quicksort
  – Uses a divide and conquer approach
  – To alphabetize a list of names (or a pile of papers):
    • Divide the list into two piles
      • Choose a “split” value such as “L”
      • The two piles will then be A-L and M-Z
    • Place each paper into the appropriate pile
    • Divide each pile again based on a new split value
    • Keep doing this until each pile only contains 1 or 2 items
      • Sorting 1 or 2 items is trivial
    • Put the piles together in order, and you are done
Sorting a List of Data

• Example of quicksort

Quicksort Performance

• Performance relies on finding a good split value
  – Optimal split value gives sub-lists about the same size
    • $O(n \log n)$:
      ~20,000,000 operations required for 1,000,000 items
    • A poor choice of split could lead to very different-sized lists, and thus poor performance
      – Degenerates to selection sort, $O(n^2)$, if every choice of split is a terrible choice

Sorting Algorithms

• Development and analysis of sorting algorithms is an active area of research in computer science
  • [www.cs.ubc.ca/spider/harrison/Java/](http://www.cs.ubc.ca/spider/harrison/Java/) has a nice visual comparison of the performance of some common sorting algorithms
Algorithms and Application Design

• We have only covered two types of problems
  – Searching and sorting
  – Many different algorithms can solve the same problem, some better than others
  – Usually there is not a best algorithm for all situations
• Thousands of such fundamental problems exist
  – Finding the shortest path (the ‘traveling salesman problem’)
  – Word matching
  – Job scheduling
  – Beating the stock market
  – Etc., etc…
• Notice we didn’t even talk about coding
  – There’s more to computer science than just programming!