

## ILM's for CS5070 Fall 2009

In some cases, the specified ILM's may require that a student have completed a specific upper division CS class. For example, in general, a student should have completed or possibly be taking CS5460 in order to implement any of the security related ILM's

### 1. Injection Attacks (Security Related)

Injection attacks are a class of malicious activities that targets vulnerabilities in software applications for the purpose of stealing information or compromising system resources. This ILM will introduce the software vulnerabilities that allow attacks, common types of attacks, and techniques to defend against attacks. The following are examples of common types of injection attacks.

- a. Buffer Overflow
- b. Code Injection
- c. SQL Injection
- d. XSS

### 2. Wireless Attacks & Protections (Security Related)

Wireless networking provides exceptional convenience, but provides new avenues for an attacker to compromise resources and data. This ILM will explore common wireless attacks, and the associated defenses. It involves teaching about wireless protocols, including those parts which enable an attacker to compromise wireless systems and communication. The following are examples of topics that the ILM will include.

- a. Sniffing
- b. MITM
- c. Rogue WAP
- d. Monkey Jack
- e. WEP
- f. WPA

### 3. Network Protocols & Security (Security Related)

Communication protocols are the glue that holds a network together logically. A solid understanding of network protocols is essential to fully grasp the process and complexity of network-based computer attacks. This ILM will introduce important network protocols, including instruction on how the protocols can be used to facilitate an attack. In addition, the ILM will introduce protocols that are designed to defend against malicious activities. The following are examples of protocols that may be included.

- a. TCP
- b. IP

- c. UDP
- d. SSL/TLS
- e. VPN
- f. IPSec

#### 4. Parameter Passing

Create an ILM which demonstrates both the effect and implementation of the following types of parameter passing:

- a. Pass by value
- b. Pass by reference
- c. Pass by value result
- d. Pass by name
- e. Pass by need
- f. Pass by constant value - if a parameter is transmitted by constant value, then no change in the value of the formal parameter is allowed during program execution.

*Log on to "csilm.usu.edu". Click "Browse resources". See "Parameter Passing" under "Programming Languages" for some ideas as to how this might be done.*

#### 5. Polymorphism The purpose is to illustrate the difference between static and dynamic binding when you have inheritance. Demonstrate how they are implemented.

- a. Allow the user to predict what the results will be under various binding criteria. You'll need a half dozen code segments to show various situations and allow testing of knowledge.
- b. Show physically how the polymorphism is accomplished. How is it that the system can find the correct inherited routine?
- c. Allow the user to create abstract classes and show how the compiler can tell whether or not an instance is allowed.
- d. Maybe the user can click on methods within a class to decide whether or not the method will be present. If a method was missing, the parent would have to be called even in late binding.

*Log on to "csilm.usu.edu". Click "Browse resources". See "Polymorphism" under "Programming Languages" for some ideas as to how this might be done.*

#### 6. Array Accessing

Demonstrate how array accessing formulas work. Array accessing formulas depend on a few things:

- a. row major or column major storage

- b. size of individual elements
- c. number of values in each dimension
- d. where each dimension begins. In C++/Java, each dimension begins with 0, but if they can begin somewhere else, the formulas change.

*Log on to "csilm.usu.edu". Click "Browse resources". See "Array Access" under "Programming Languages" for some ideas as to how this might be done.*

## 7. Good programming style

Show the importance of good variable names and documentation. Show code snippets and have user describe what the code does. Time the results for various levels of documentation to show how documentation makes a huge difference in understanding. Allow the user to select various levels of "documentation" for the same piece of code to see how they make a difference. In order to tell if they understood the code, we would have to have multiple choice answers. Keep track of how long each piece of code takes to understand correctly and post the results. Or, have the users evaluate the quality of the code and post the results as a histogram of results.

*Log on to "csilm.usu.edu". Click "Browse resources". See "Good Programming Style" under "Programming Languages" for some ideas as to how this might be done.*

## 8. Number Representation

Everyone is aware that computers use binary representations for numbers whereas most people use decimal. Every computer systems has a specific scheme for representing numbers, e.g. integers, floating point, exponential, ASCII, etc. The purpose of this ILM would be to teach (and assess) students to be able to take a given decimal string and convert it to various internal computer binary or hexadecimal representations. They should also be able to do the reverse, i.e. binary string (or hexadecimal) to decimal. The general formats to consider are:

- a. Integer decimal to: (word size to be determined)
  - i. Sign magnitude binary
  - ii. 1's complement binary
  - iii. 2's complement binary
- b. Reverse of (a), e.g. 2's complement binary to integer decimal
- c. Floating point decimal, including exponential to: (word size to be determined)
  - i. IEEE 754 format
    - 1. Single precision (32-bit)
    - 2. Double precision (64-bit)
  - ii. Some other floating point format – need 2 to 3 of these
- d. Reverse of (c), e.g. IEEE 754 in binary or hex to decimal equivalent

- e. Errors in various representations
- f. Limits and limitations

9. A tutorial on image (affine) transformations

Develop a tutorial to show students how (by graphics/video) the various types of transformations (scale, translate, rotate, etc. ) affect how one sees an image.

10. Genetic algorithms (swarm vs standard GA operations on a population of chromosomes)

A chromosome is a binary string. For a variety of 2-variable functions show how standard GA's and swarm techniques evolve to the optimal solution.

11. Boolean Algebra

Develop an ILM to show the basics of Boolean Algebra, such as:

- a. Define a minimal set of Boolean "laws", then show how those laws can be used to derive other basic Boolean equations.
- b. Given a truth table, what are the canonical SOP and POS forms?
- c. Given a Boolean function, what is the equivalent truth table?
- d. Show how don't cares arise and how they can be used in developing a Boolean expression.
- e. Show the basics of algebraic minimization

12. Sorting

Show in video, with user control for speed, how the various sorting algorithms work.

This should be an improvement on the existing ILM.

*Log on to "csilm.usu.edu". Click "Browse resources". See "Sorting" under "Algorithms and data structures".*

13. T9 Algorithm(<http://www.t9.com/t9trainer/Web/trainer.html>) Implement the T9Trainer game.

*Log on to "csilm.usu.edu". Click "Browse resources". Select "Teacher Published Lessons". See "Computer Programming" under "Logan High School". Now click "Cell Phones: Text on 9 keys" for some ideas as to how this might be done.*

14. Iteration. Have beginning programmers experiment with for loops to create graphical patterns.

Log on to "csilm.usu.edu". Click "Browse resources". Select "Teacher Published Lessons". See "Computer Programming" under "Logan High School". Now click "Iteration" for some ideas as to how this might be done.

15. Code snippets. Rather than have new students write complete programs, allow them to work within a "sandbox" where variables have been declared, classes have been defined, etc. They will need to be able to enter code, have it compiled (along with the provided code), and see error messages.
16. Dynamic programming. Dynamic programming is always a tricky thing for students to understand. Students need to:
  - a. compare with the greedy method
  - b. understand why recursion is slower (due to duplication)
  - c. Be able to fill in the tables by themselves

A student is already exploring one specific dynamic programming problem (making change). Lots of others could be implemented:

- i. Maximum Value Contiguous Subsequence. Given a sequence of  $n$  real numbers  $A(1) \dots A(n)$ , determine a contiguous subsequence  $A(i) \dots A(j)$  for which the sum of elements in the subsequence is maximized.
- ii. Making Change. You are given  $n$  types of coin denominations of values  $v(1) < v(2) < \dots < v(n)$  (all integers). Assume  $v(1) = 1$ , so you can always make change for any amount of money  $C$ . Give an algorithm which makes change for an amount of money  $C$  with as few coins as possible. [on problem set 4]
- iii. Longest Increasing Subsequence. Given a sequence of  $n$  real numbers  $A(1) \dots A(n)$ , determine a subsequence (not necessarily contiguous) of maximum length in which the values in the subsequence form a strictly increasing sequence. [on problem set 4]
- iv. Box Stacking. You are given a set of  $n$  types of rectangular 3-D boxes, where the  $i^{\text{th}}$  box has height  $h(i)$ , width  $w(i)$  and depth  $d(i)$  (all real numbers). You want to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2-D base of the lower box are each strictly larger than those of the 2-D base of the higher box. Of course, you can rotate a box so that any side functions as its base. It is also allowable to use multiple instances of the same type of box.
- v. Building Bridges. Consider a 2-D map with a horizontal river passing through its center. There are  $n$  cities on the southern bank with  $x$ -coordinates  $a(1) \dots a(n)$  and  $n$  cities on the northern bank with  $x$ -coordinates  $b(1) \dots b(n)$ . You want to connect as many north-south pairs of cities as possible with

bridges such that no two bridges cross. When connecting cities, you can only connect city  $i$  on the northern bank to city  $i$  on the southern bank. (Note: this problem was incorrectly stated on the paper copies of the handout given in recitation.)

- vi. Integer Knapsack Problem (Duplicate Items Forbidden). This is the same problem as the example above, except here it is forbidden to use more than one instance of each type of item.
- vii. Balanced Partition. You have a set of  $n$  integers each in the range  $0 \dots K$ . Partition these integers into two subsets such that you minimize  $|S_1 - S_2|$ , where  $S_1$  and  $S_2$  denote the sums of the elements in each of the two subsets.
- viii. Edit Distance. Given two text strings  $A$  of length  $n$  and  $B$  of length  $m$ , you want to transform  $A$  into  $B$  with a minimum number of operations of the following types: delete a character from  $A$ , insert a character into  $A$ , or change some character in  $A$  into a new character. The minimal number of such operations required to transform  $A$  into  $B$  is called the edit distance between  $A$  and  $B$ .
- ix. Counting Boolean Parenthesizations. You are given a boolean expression consisting of a string of the symbols 'true', 'false', 'and', 'or', and 'xor'. Count the number of ways to parenthesize the expression such that it will evaluate to true. For example, there is only 1 way to parenthesize 'true and false xor true' such that it evaluates to true.
- x. Optimal Strategy for a Game. Consider a row of  $n$  coins of values  $v(1) \dots v(n)$ , where  $n$  is even. We play a game against an opponent by alternating turns. In each turn, a player selects either the first or last coin from the row, removes it from the row permanently, and receives the value of the coin. Determine the maximum possible amount of money we can definitely win if we move first.
- xi. Two-Person Traversal of a Sequence of Cities. You are given an ordered sequence of  $n$  cities, and the distances between every pair of cities. You must partition the cities into two subsequences (not necessarily contiguous) such that person  $A$  visits all cities in the first subsequence (in order), person  $B$  visits all cities in the second subsequence (in order), and such that the sum of the total distances travelled by  $A$  and  $B$  is minimized. Assume that person  $A$  and person  $B$  start initially at the first city in their respective subsequences.
- xii. Bin Packing (Simplified Version). You have  $n_1$  items of size  $s_1$ ,  $n_2$  items of size  $s_2$ , and  $n_3$  items of size  $s_3$ . You'd like to pack all of these items into bins each of capacity  $C$ , such that the total number of bins used is minimized.

Log on to "csilm.usu.edu". Click "Browse resources". See "Dynamic Programming" under "Data Structures: CS2420" for some ideas as to how this might be done.

## 17. Matching

In the mathematical discipline of graph theory a matching or edge-independent set in a graph is a set of edges without common vertices. It may also be an entire graph consisting of edges without common vertices.  
Maximum Matching: A matching  $M$  is said to be Maximum if we cannot find another matching which has more matches. Allow the user to discover algorithms for maximum matching.

Log on to "csilm.usu.edu". Click "Browse resources". See "Matching Problems" under "Algorithms and Data Structures" for some ideas as to how this might be done.

## 18. Huffman Coding

Huffman coding uses a specific method for choosing the representation for each symbol, resulting in a prefix code (sometimes called "prefix-free codes") (that is, the bit string representing some particular symbol is never a prefix of the bit string representing any other symbol) that expresses the most common characters using shorter strings of bits than are used for less common source symbols. Huffman was able to design the most efficient compression method *of this type*: no other mapping of individual source symbols to unique strings of bits will produce a smaller average output size when the actual symbol frequencies agree with those used to create the code. A method was later found to do this in linear time if input probabilities (also known as *weights*) are sorted.

Notes to Programmer:

- Allow the user to input a selection of text or use a file of text already in the system.
- From the text, show the user the frequency of each character.
- Show the user how the huffman code is built. Have a version where the user is able to select the two trees to combine.
- Ask the user to use the huffman tree to code or decode a message
- Illustrate the savings in storage space.

## 19. Pointers

Devise an exercise which helps students use pointers correctly. Some possibilities of properties to enforce are found in:

[http://mudge.net/cppclass/class\\_5/class\\_5\\_pointers.htm](http://mudge.net/cppclass/class_5/class_5_pointers.htm)

## 20. NetBeans IDE Modification for future use.

The overall goal is to modify the NetBeans environment/software so that it can be better used to demonstrate programming or computational science concepts. This is a proof of concept and demonstration part. Once this is done, more will be added. It is not so much a demonstration of the utility of the idea, but proof that it can be done.

## 21. YOUR IDEA FOR AN ILM