ECE 220 Learning Objectives

Week 1 (Lectures 1 & 2):

1. Write LC-3 programs using memory operations, arithmetic/logical operations, and control instructions.

2. Understand memory-mapped I/O in LC-3 and write programs that can read input from keyboard and print output to monitor.

3. Understand how TRAPs are invoked in LC-3 and use basic TRAPs such as GETC and OUT.

4. Differentiate between callee-saved and caller-saved conventions for saving and restoring registers and apply these concepts when writing or using subroutines in LC-3.

Week 2 (Lectures 3 & 4):

1. Explain the stack abstract data type (ADT), including its order of access (LIFO) and common implementation methods

2. Implement and use stack operations (PUSH & POP) in LC-3

3. Identify problems suitable to be solved using stacks and implement the solutions in LC-3

Week 3 (Lectures 5 & 6):

1. Identify the scope of a variable (global vs. local) and use it within its scope.

2. Understand how a C program utilizes different regions of memory during execution.

3. Write code using C operators, control structures, and I/O operations such as scanf and printf.

Week 4 (Lectures 7 & 8):

1. Explain the advantages of using functions in C for modularity, readability, and code reuse.

2. Write and implement functions in C, including parameter passing and return value.

3. Translate C code containing function calls into LC-3, applying proper stack set-up and tear-down conventions.

Week 5 (Lectures 9 & 10):

1. Create and dereference pointers in C.

2. Distinguish between pass-by-value and pass-by-reference in C function calls.

3. Apply correct array indexing and recognize the importance of array bounds.

4. Use the relationship between arrays and pointers (pointer-array-duality) to simplify code syntax.

5. Explain the importance of array bounds checking in C.

6. Use multi-dimensional arrays in C and understand how they are stored in memory.

Week 6 (Lecture 11):

1. Convert between 2-D array representations and equivalent 1-D representation in C.
2. Implement and analyze simple searching algorithms (linear search, binary search) and sorting algorithms (bubble sort, insertion sort) in C.

Week 7 (Lectures 12 & 13):

1. Understand the differences between recursion and recurrence.

2. Implement the base base(s) and recursive case(s) of a recursive function in C.

3. Identify problems that can be solved by recursion with backtracking.

4. Implement calling conventions (stack built-up and tear-down) in LC-3 for a recursive function in C.

Week 8 (Lectures 14 & 15):

1. Use built-in file I/O functions in C to read from and write to files.

2. Differentiate between enums, structs, and unions in C and identify use cases for each user-defined data type.

3. Declare structs in C, describe the impact of padding on their size, and correctly access members in a struct, struct arrays, and pointers to struct.

Week 9 (Lectures 16 & 17):

1. Understand the difference between "static" and dynamic memory allocation in terms of allocation mechanism, lifetime, location, and size.

2. Dynamically allocate and deallocate memory using malloc(), free(), calloc(), and realloc().

3. Understand the difference between arrays and linked lists in terms of memory allocation, memory structure, overhead, order of access, and data insertion/deletion.

4. Implement functions in C for linked list operations such as traversal, insertion, and deletion.

Week 10 (Lecture 18):

1. Understand the difference between singly linked list and doubly linked list.

2. Implement stack and queue abstract data types using linked lists in C.

3. Differentiate between using head and tail points for enqueue and dequeue operations in a singly linked list, and explain the impact on efficiency.

Week 11 (Lectures 19 & 20):

1. Implement calling conventions (stack built-up and tear-down) in LC-3 for a recursive function in C that performs linked list operations.

2. Differentiate between a struct in C and a class in C++.

3. Apply cin and cout to perform I/O operations in C++, and new and delete for dynamic memory allocation and deallocation.

4. Understand and implement operator overloading and function overloading in C++.

Week 12 (Lectures 21 & 22):

1. Differentiate pass by value, pass by pointer (address), and pass by reference in C++.

2. Explain the purpose and advantages of passing objects by constant reference in C++ and apply this concept in implementing functions such as copy constructors.

3. Distinguish between base and derived classes in C++ and explain how inheritance supports code reuse and hierarchical class organization.

4. Explain the concepts of virtual functions and virtual function tables in C++ and apply them to enable run-time polymorphism where appropriate.

5. Differentiate between shallow and deep copies in C++ and correctly implement the Big Three to manage dynamic resources.

6. Understand the concept of templates in C++ and apply it to effectively use vector and list containers from the C++ Standard Template Library (STL).

Week 13 (Lectures 23 & 24):

1. Define and correctly use tree terminology, such as root, branch/edge, leaf, parent, child, height, when describing tree structures.

2. Differentiate between a general binary tree and a binary search tree.

3. Perform pre-order, in-order, and post-order traversals on a given binary (search) tree.

4. Implement essential tree operations in C/C++, including node insertion, tree traversal, and computing the height of a tree.

Week 14 (Lecture 25):

1. Understand the difference between Polling and Interrupt-Driven I/O in LC-3.

2. Explain the LC-3 interrupt mechanism and describe how processor state is saved and restored during interrupt.

3. Determine whether a given LC-3 program is executing in user mode or supervisor mode by analyzing its instructions, memory usage, and contents on the supervisor stack.