Programming Projects

NOTE: the programs listed here do not represent all the work I have done in my classes or outside of class. I'm just listing some examples.

C Programming

Many interfaces here were done using the C opaque object design -- using void pointers as handles to objects.

Unbeatable Hangman Game

- Used a dictionary file of over 127,000 words to create a virtually unbeatable hangman game. It is not beatable unless the user selects a very high amount of guesses.
- Developed a string interface similar to the string classes in modern object-oriented languages, a generic vector interface for objects, and an AVL tree interface customized for the game. Implemented C opaque object design for the interfaces.
- Developed a more efficient method of extracting a vector from the AVL tree which led to the removal of some lag in early rounds of the game.
- Completed the game with zero memory leaks.
- Gained experience in unit testing.

String Interface

- Used in the game as described above. This string interface behaves like the "string" or "String" classes in C++, Java, C# etc. and can dynamically change in size unlike the c-strings traditionally used in C.

Generic Vector Interface

- Used in the game as described above. This is a generic vector because it is a vector that can hold objects of any type. In this game it happened to hold the string objects from the string interface.

AVL Tree Interface

- Used in the game as described above. Created a customized AVL tree interface for the game. Each node in the tree contained a string key and a generic vector of string objects corresponding to the key. For example, if the key in the node was "b---b---" then a string object in the generic vector could be "baseball".

Integer Bit Flags Interface

- Created an integer vector interface that can much more efficiently keep track of numbers. Instead of using an integer for each number the bits in each integer represent a single number. So, for example, to mark that 5 has been entered, the bit at position 5 in the first integer in the vector is set. To mark that 40 has been set, the bit at position 40 (which is in the second integer in the vector) is set. This means that each integer can be used to represent many numbers (i.e. a single 32 bit integer could be used to represent 32 numbers) thereby saving memory usage during the runtime of a program.

Priority Queue Interface

- Created a priority queue interface for data items that uses a heap data structure.

Average/Standard Deviation Interface

- Created an interface to calculate the average and standard deviation of a set of N random numbers up to a given size.

Points on a Plane Interface

- Created an interface to store a set of x-y coordinate points in an object, determine the distance between two points, swap points, and determine if 3 points are collinear up to a certain error tolerance.

Complex Numbers Interface

- Created an interface to represent complex numbers and do mathematical calculations on them.

Polynomial Interface

- Created an interface to represent a polynomial of any size. Used arrays and linked lists. Can add, subtract, multiply, evaluate the polynomial at a given number, differentiate, and integrate the polynomial.

Linked List Interfaces

- Created an interface for data items in a singly linked list, doubly linked list, and circular linked list. Created separate interfaces for including duplicates and ignoring duplicates (can choose between ignore the new item policy vs. replace the old item policy).
- Can do things such as print the whole list, move the largest item to the end, move the smallest item to the front, swap all the nodes in even positions with nodes in odd positions (and vice versa), exchange two nodes in the list, create a copy of the list, and reverse the list among others.

Stack Interfaces/Calculator Program

- Created a stack interface for data items using both a linked list or an array. Created separate interfaces for including duplicates and ignoring duplicates (can choose between ignore the new item policy vs. replace the old item policy).
- Using a stack created a calculator program that can do calculations using both prefix and postfix expressions.

Queue Interfaces

- Created queue interfaces using both arrays and linked lists. Created different interfaces for regular queues and double ended queues as well as including duplicates or ignoring duplicates (can choose between ignore the new item policy vs. replace the old item policy).

Binary Tree Interface

- Created a binary tree interface for data items. Can do things like insert into the tree, traverse the tree in various traversal orders, calculate the tree's height, and calculate the number of nodes in the tree among others.

Grader Repository

- Created a repository of comprehensive tests for all the assignments in the data structures course for which I grade. There are 12 assignments in total and all are in C.

C++ Programming

Image Encryption/Decryption Program (C++ and SFML APIs)

- Created a program that can be used to encrypt or decrypt an image.
- Used a 16 bit seed to initialize a custom linear feedback shift register which then manipulates the red, green, and blue components of each pixel of the image thereby encrypting it. If the same 16 bit seed is used on the image after encryption then the image will be decrypted back to its original form.

Solar System Simulation (C++ and SFML APIs)

- Created a solar system model that displays a space background, images of planets, and then accurately models their orbits around the sun. A timer is also displayed to show the time that would have elapsed and the theme song to the movie Interstellar plays in the background.

Guitar String Simulation (C++ and SFML APIs)

- Created a program that allows the user to play 37 different guitar notes from the keyboard. Generated the guitar simulation by using SFML's audio features as well as using multiple class implementations I created myself. This included a circular buffer class which was used as a queue to simulate the Karplus-Strong algorithm to help create the correct frequencies for each key on the keyboard. I was able to play my favorite song, Moonlight Sonata, on my own keyboard.

Other Assignments

These assignments came from my computer architecture course and were some of the most challenging assignments I've encountered so far so I thought I would include them here.

Mic-1 Assembly Language Multiplication and Division

- Wrote the multiplication and division operations in microcode for the Mic-1 hardware system. By writing the microcode the Mic-1 Assembly language could then support multiplication and division with the commands MULT and DIV.

Mic-1 Assembly Language Linker

- Wrote a linker that can link together multiple Mic-1 files into a single program.