# Programming Assignment 2

**Problem 1:**

I solved this problem by using 2 stacks. The program begins by prompting the user to enter some integers. I used 6 integers while testing the program. My function sort sorts the stack in order so that the smallest number entered is on the top of the stack. It does this by using 2 while loops; the first checks that the original stack (called input) isn't empty and the second checks that the new temporary stack isn't empty and the value of the integer temp (which is initialized to the first element in the original stack) is greater than the first element in the temporary stack. The second while loop pushes the first element in the temporary stack into the input stack (the original). The first while loop then pushes the value of temp into the temporary stack. Since this continues until the input stack no longer has any digits, and since the second while loop will only function if the first element in the temporary stack is less than the temp value, the returned stack will be in order from least to greatest.

**Problem 2:**

Similar to the first problem, this program functions using 2 stacks to mimic a queue. It again starts by prompting the user for input, this time testing was done using only 5 integers. I used a templated class so values of any type (integers, char, string, etc) could work for the MyQueue class. It again uses the main stack and a temporary stack for holding values. The function front returns the value at the top of the main stack. The function push will move a new value into the bottom of the stack, then moves all elements from the main stack to the temp stack. By doing this the elements are first pushed in reverse order (first stack) and then in the original order again (mimicking a queue FIFO). The function pop is used to pop the top of the stack, and the function bool checks whether the main stack is empty or not.

**Problem 3:**

This program loops through a linked list and determines whether it is circular (whether a node is found more than once). I tested the function using the sequence that was given in the problem ABCDEC and used the main function for testing. I created 2 functions and a struct for this problem. The struct contained the data (or value) of the current node and the struct node link served to point to the next node. The function append served to just build the linked list based on the string that was used (it appended each character to the linked list one at a time). The function FindStartOfNode was used to determine whether there was a loop in the linked list, and if so, at which letter it occurred.

**Problem 4:**

For this program, I just wrote the function that would find the sum of 2 linked lists. Thus function utilized 2 while loops and several if/else statements. The while loop continued for the 2 linked lists until one or both of them were null (until it reached the end of the list). If l1/l2 were not null, their value was added to value of sum. If the sum was greater than 10, then a carry value of 1 was initialized, otherwise it was 0 (just as in basic arithmetic). Once this first value was calculated, it moved on to the next node and computed their sum as well until there were no more sums and carries. The output was printed in reverse order (for example, the output [2, 1, 9] demonstrated by my code output really signified 912.)

**Programming Assignment 3**

**Problem 1:**

For this program, I want to preface by stating that I created this program in C++11. I use C++11 at my internship and thought some of its features would better suit this program. The maze the robot had to traverse through was created using a 2D vector. I created a class for the robot called Robot and used the main function for testing purposes. The first function was a constructor called Robot that used the values for row and column size to initialize the grid. The function show_grid() outputted a display of the grid the robot would be travelling through. The function isFree was used to determine whether a position on the grid was an acceptable position for the robot to move to. The function showOffLimit was used to determine where the robot couldn't move. The location was marked by a 1 if the robot couldn't move there, and a 0 if it could. The function findPath used a list of strings to determine the different paths the robot could take to travel through the graph. Moving down was added to a list of strings by the letter "D" and moving right was added to the list of strings using "R". The final paths were outputted by the main function.

**Problem 2:**

This program was solved using depth first search. The program calls DFS twice. DFS was used to check if the current node was visited, and then check if the adjacent vertices were visited. The function getTranspose was used to iterate through each vertex in the 2D array and transpose it (so the columns now become the rows) and thus reverse the graph. fillOrder recursively checks that each vertex was visited and then pushes these vertices to the stack Stack. The SCC function actually gets the strongly connected components. It marks each vertex as unvisited, and then fills the vertices based on the finishing times. It calls getTranspose to reverse the graph and then iterates through the graph and marks the vertices as unvisited. While the Stack is not empty, it initializes the vertex to the be value of the top of the Stack and then pops the element from the Stack. If the vertex is not visited, it calls DFS (the second DFS call) and then prints the strongly connected components of the removed vertex. The main function was used for testing purposes.

**Problem 3:**

For this problem, Dijkstra's algorithm was used to determine the minimum weight cycle in an undirected weighted graph. The main function was again used for testing. The class udGraph was for the undirected graph. It's function insertEdge inserted and edge into the list of edge to start building the graph. The function delete edge removed the edges that did not contribute to creating the minimum weight cycle. The function computeMinimumPath iterated through a set and determined if there was a minimum weight between the source and vertex and then added that weight to the set. The function then returned the shortest path. The function computeMinimumCycle computed the minimum distance between two vertices and then based on this distance computed the minimum cycle and returned the value of the minimum cycle.

**Problem 4:**

This program utilized Prim's MST algorithm. A class for Prim's algorithm was created. The function initmat was used to initialize all the starting values to the infinity value initially. The function create took in user input to populate the graph with the number of vertices and edges, as well as the values for u, v, and w. The function display was simply used to print the output in a way that made it easier to view. The function primsmat did the actual calculations. It started by taking in user input to initialize the source vertex. Then in a loop, it marked the vertices as visited and stored the distance from each vertex. It initially set the minimum value to the infinity value and then in a loop compared the weight of that edge to the infinity value for each edge not yet visited and then updated the minimum value. It then determined the minimum edge weights through another loop. Finally, the function printed the path and distances of the edges in the MST.