CS 2101 Machine Problem # 3
Graph and Others – May 20,2023

Machine Problem RUBRICS					
Note: At minimum, the program should run. No compilation errors.					
Criteria	**Percentage**	**Scale**			
		3	2	1	0
Meets program specifications	70%	All of the function modules are implemented correctly. (All 3 Problems are answer correctly)	No. of Problems answered correctly : 2	No. of Problems answered correctly : 1	No. of Problems answered correctly : 0
Readability	15%	Code is organized and easy to follow and 100% of the agreed coding conventions are followed	Code is fairly easy to read and 80% of the agreed coding conventions are followed	Code is readable only by somehow who knows what the code does and 60% of the agreed coding conventions are followed.	Code is poorly organized and less than 60% of the agreed coding conventions are followed
Efficiency	15%	Code is efficient without sacrificing readability. No unnecessary variables are used and no unnecessary and redundant statements. Code is at its optimum.	Code is 80% efficient without sacrificing readability. At most 20% of the code can be improved in terms of running time, storage, and lines of code	Code is 60% efficient and somehow unnecessarily long. 40% of the code can be improved in terms of running time and storage, and lines of codes.	Code is done in brute force manner.

There are 3 problems which are independent of each other and stored in 3 different files.

Problem 1: Populates and displays the Labeled Adjacency List .

Problem 2: Finds the edges of the minimum cost-spanning tree using Prim's Algorithm starting at the vertex which is inputted by the user.

Problem 3: Converts a BST to a List by removing the elements from the smallest to the biggest until BST is empty

Names of files to be submitted:

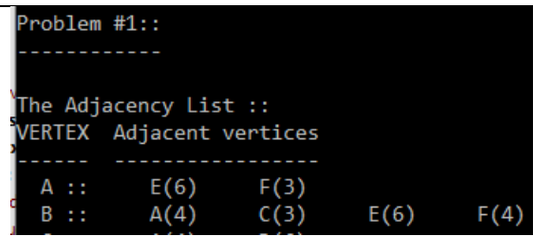
Problem 1: Final_Probl_01_lastnameXX.c //Lastname and XX are the 1st two letters of firstname

Problem 2: Final_Probl_02_lastnameXX.c

Problem 3: Final_Probl_03_lastnameXX.c

The following functions prototypes have to be implemented.

Function Prototypes	Description
Problem # 1: Creates and displays a Labeled Adjacency List by calling populateDirectedAdjList()	
`void initDirectedAdjList(directedAdjList * L);`	**The function initializes the given adjacency list representing a directed graph to be empty. .**
`directedAdjList populateDirectedAdjList();`	**The function creates the labeled adjacency list of a DIRECTED graph. The adjacency list is sorted in ascending order according to adjacent vertices.** **The edges of the graph, which are of type edgeType are given in the program. The edge is uniquely identified through the tail and head vertices. If two or more edges are the same, record the edge with the smallest weight.**
`void displayAdjList(DirectedAdjList L);`	**Partial Code is provided. The function displays the contents of the adjacency list. It displays the vertex V and the vertices adjacency to V including their weights. Weights are enclosed in parenthesis. The display includes the number of edges in the adjacency list.** **Given below is the Partial Display:**

	 . . .
Problem # 2: Finds the edges of the minimum cost-spanning tree using Prim's Algorithm starting at the vertex which is inputted by the user.	
`void populateGraph(graphType G);`	**Complete Code.**
`primMST primAlgo(graphType graph, int Vertex);`	**Given a graph and the start vertex Vertex, the function will return a list of edges in the MST.**
`void displayPrimMST(primMST tree);`	**Partial Code is given.**
	Sample output at a starting vertex 3. Start Vertex is 3 Edge Cost (3,4) 1 (3,2) 2 (4,5) 3 (3,0) 4 (0,1) 1 Minimum Cost : 11
Problem # 3: Converts a BST to a List by removing the elements from the smallest to the biggest until BST is empty	
BST createMagicalBSTvheap(VHeap *VH);	**Complete Code is provided. The function creates and populates a BST implemented using cursor-based representation in memory. In effect, the function organizes the Virtual heap. The RC field in the node of the virtual heap is used to link the nodes, and a -1 means the node is the last node in the list of nodes.**
void displayVHeap(VHeap V);	**Partial Code is provided. The function displays the indexes of the nodes in the virtual heap, together with the product ID, LC and RC fields. Included in the display is the value of the available cell.** **Given below is the partial expected output.**
void displayProduct(product P);	**Complete Code is provided. The function displays the information about the a given product.**
ArrayList convertBstToList(BST *B, VHeap *VH);	**Given the BST and the virtual heap, the function will convert the BST into an array list by removing the elements from the BST one at a time from the smallest to the biggest element until the BST becomes empty. The newly created array list will be returned to the calling function.**
void displayList(ArrayList A);	**The function displays all products in the List. This function calls displayProduct().**