CS163 Feedback for Lab #: 8 Page 1 **Name:**_____

Getting Set Up – Check solutions for the first page of the lab before logging on

Step	Measurement	Expected	Assessment
Group Step 1	• Creates the vertex structure • Having the journal entry be a pointer allows us to represent a NULL (or not stored) vertex • Could also use a visit flag!	**struct vertex** **{** **journal * entry;** **struct node * head; //"incomplete" declaration** **};**	

Group Step 2	• Creates the node structure for edge list	**struct node**
		{
		vertex * adjacent; //could be an index or a key
		node * next;
		};	

Group Step 3	• Create an array of vertices, dynamically allocated of size N	**vertex * adjacency_list; //data member** Constructor: **adjacency_list = new vertex[N];**	
Group Step 4	• Underlying type is…	• A vertex. This vertex's head and entry pointer should be NULL **for (int i = 0; i < N; ++i)** **{** **adjacency_list[i].entry = NULL;** **adjacency_list[i].head = NULL;** **}**	
Group Step 5	• Indices range…	From 0 (zero) through N-1 (size-1)	

Step 6a	• Constructor • Allocate memory for the adjacency list and initializes each element's pointers to NULL • Save the size sent in as an argument to the list_size data member:	**adjacency_list = new vertex [size];** **for (int i = 0; i < size; ++i)** **{** **adjacency_list[i].head = NULL;** **adjacency_list[i].entry = NULL;** **}** **list_size = size;**	
Step 6b	• Insert a vertex • Insert into the first available location in the array, unless there is no space available • Watch out for proper use of -> versus . • We couldn't use the = operator if temp was a vertex object instead of a pointer	• Loop through the array of vertices looking for a non-null entry pointer. • If the entire array is full, then return a zero • Once a location is found, create a new journal_entry and use the copy_entry function from the journal_entry class **temp = new journal_entry;** **temp-> copy_entry(argument)** **adjacency_list[i].entry = temp;** • Minimizes the use of the [] operator	
Step 7a	• Find Location • Make sure to use the compare function from the journal class – since we do not have access to the private title to strcmp!	• To attach vertices together, we will be given key_value (arrays of characters). We need to find the actual array location that contains that key_value. • Loop through from zero while the index is less than the list_size, comparing the key value for each journal entry. • Check if the entry is NULL before dereferencing! **if (adjacency_list[i].entry->compare(key)) return i**	
Step 7b	• Provide the pointer diagram	• Does the diagram correctly represent the data structure?	
Step 7c	• Insert an edge • We now need to use the find_location function to determine which vertices hold these values	• Use the find_location function for the two key values: int connection1 = **find_location(first_key_value);** int connection2 = **find_location(second_key_value);**	
Step 7c	• Next, we can add a new node to the beginning of the edge list • Minimizes the use of the [] operator • Notice the use of the address-of operator	• If it is a directed graph, then only insert a new node into one edge list. **temp = new node;** **temp->adjacent = &adjacency_list[connection2];** **temp->next = adjacency_list[connection1].head;** **adjacency_list[connection1].head = temp;**	
Step 7c	• If it is not a directed graph, then the connections should be added for both vertices	Also do: **temp = new node;** **temp->adjacent = &adjacency_list[connection1];** **temp->next = adjacency_list[connection2].head;** **adjacency_list[connection2].head = temp;**	
Step 7d	• Evaluate using a pointer diagram	• Step through each line of your code with a pointer diagram. Make no assumptions. Don't skip any steps!	

Step 8a	• Display the adjacent journal entries for a particular vertex • First find the location in the array of vertices	• We need to find the actual array location that contains that key_value. • Loop through from zero while the index is less than the list_size, comparing the key value for each journal entry. • Check if the entry is NULL before dereferencing! **if (adjacency_list[i].entry->compare(key))** //now we can display!	
Step 8a	• Now we can display all adjacent vertices by traversing the edge list for this vertex • Avoid dereferencing NULL pointers! Never assume.	**node * current = adjacency_list[i].head;** **while (current)** **{** //display the journal_entry by calling the display function **if (current->adjacent)** **current->adjacent->display();** **current = current->next;** **}**	
Step 8c	• Provide the pointer diagram	• Does the diagram correctly represent the data structure?	