Ampeloquio, Joshua Jonathan M.

CIS – 202

LOGICAL EXPLANATION OF CODES

1. LIST

For the example of List as a Data Structure, I used a simple example of an array. The java code is named ‘ListByArray’, the code allows the user to choose the size of the array, and input values inserted to the array, and display the array in a list order. I used a scanner object to use to take the input from the user. The integer array is named ‘number’ and is created by the specified number inputted by the user. Then a for-loop was used to store the values in the array. Then the content of the array was called by using the ‘displayArray’ method, this method prints each element of an integer array that is given as a parameter.

2. LINKED LIST

For the example of Linked List as a Data Structure, I used a simple java program to demonstrate the use of doubly linked list operations like adding, deleting, and displaying the linked list through examples of characters from the Avengers universe. ‘Node’ was used to identify the nodes, the input are String because the values are letters and not integers, ‘prev’ and ‘next’ are used to preview the previous and next nodes. Then a ‘DoublyLinkedList’ class is called to perform and manage the operations on the doubly linked list, the operations include inserting a node at the beginning and the end as well as deleting the node and displaying the elements of the linked list. ‘insertAtBeginning ()’ ‘insertAtEnd()’ was used to add nodes front and back, while ‘delete(String data)’ was used to delete the input then if deleted it displays the list using ‘display()’.

3. STACK

For the example of Stack as a Data Structure, I used the existing activity I already passed as an example for the activity ‘STACK APPLICATION - POSTFIX EVALUATOR JAVA PROGRAM’. Postfix is when the operators were written after their operands. The code has ana array named ‘a’ and an integer named ‘top’ to keep track of the top of the stack, then the constructor initializes the top of the stack to -1. The ‘push’ method in the code was used to push the value onto the stack nad the method ‘pop’ to pop/delete a value from the stack, then a ‘isDigit’ method is called to check if the character is a digit or not. Then a ‘val’ method is called that performs the mathematical operation based on the given operator (x) and two operands (v1 and v2). The ‘eval’ method is called that takes a character array representing a postfix expression and evaluates its result using the stack. Then the ‘Main” method is where the programs lets the user input the postfix expression, and it being cevaluated by the ‘eval’ method using stack and after that the final result is displayed.

4. QUEUE

For the example of Queue as a Data Structure, I used the existing activity I already passed as an example for the activity for ‘MODULE 6 - QUEUE APPLICATIONS LAB ACTVITY’. Ciphertext is an encryption technique where each letter in the plaintext is shifted a certain number of positions down or up the alphabet. The code uses a scanner object to take input from the user, the input needed are the cleartext in String data type and the variable ‘k’ which is the value of the shift. Then, the ‘encrypt’ method is called with the plaintext and ‘k’ the value of shift. The program also checks if the cleartext is has uppercase and lowercase letters. This code demonstrates a simple implementation of the Caesar Cipher encryption technique, allowing users to input a message and a shift value for encryption.

5. TREE

For the example of Tree as a Data Structure, I used the existing activity I already passed from ‘Module 7 - Counting the Leaves of a Tree’. The program started with a class named ‘Node’ that represents the node in a tree. Each node contains a string item, and references to the left and right child nodes. Then the ‘countLeaf’ method is called to take the input, count, and print the leaf nodes of the subtree rooted at that node. If both left and right children are null, it is a leaf node, and its item is printed. In summary, the program builds a tree and determines how many leaf nodes it has. Nodes with no children are called leaf nodes (left and right references are null), and those are printed and counted.

6. BINARY TREE

For the example of Binary Tree as a Data Structure, I used a simple java program that demonstrates how binary trees were made and then it displays its structure using an in-order traversal. The program started with a class named ‘Node’ that represents the node in a tree. Each node contains a string item named ‘data’, and references to the left and right child nodes. In the ‘main’ method an instance of the ‘BinaryTree’ is created, then Nodes are created and connected to form a binary tree and the ‘displayTree’ method is called to display the structure of the binary tree. It repeatedly visits the left subtree, then prints the current node's data, and then go repeatedly visits the right subtree. Then the ‘displayTree’ method is called with the root of the tree to display the structure of the binary tree. In summary, the program shows how to build a binary tree and how to display its structure using the in-order traversal method. The nodes of the example tree are shown in the correct order.

7. GRAPH

For the example of Graph as a Data Structure, I used the existing activity I already passed as an example from the activity for ‘MODULE 8 - GRAPHS IN DATA STRUCTURES’. The program called the ‘Graph’ class and inside it is an inner class named ‘Edge’ to represent the edges in the graph, the class has integer variables ‘vertices’ for the number of vertices and ‘edges’ for the number of edges. Then the ‘Graph’ class which is a constructor initializes the number of vertices and edges and creates an array of Edge objects. In the ‘Main’ method there is a ‘Graph’ class where the number of vertices and edges are specified, and a graph object named ‘g’ is instantiated. The edges were added by using ‘.src’ for source and ‘.dest’ of vertices. The main method then prints the graph by going through the array of edges and displaying the source and destination vertices of each edge. This java program defines a graph with 10 vertices and 18 edges. It then prints the edges of the graph. The graph represents connections between vertices as specified in the code.