National University of Computer and Emerging Sciences

Laboratory Manual

for **Data Structures Lab**

Course Instructor	Dr. Amna Khan
Lab Instructor(s)	Muhammad Saddam Mian Basam
Section	CS-E
Date	03-Dec-2020
Semester	Fall 2020

Department of Computer Science

FAST-NU, Lahore, Pakistan

Objectives:

In this lab, students will practice:

1. Implementation of Huffman Encoding using Heaps and Binary Trees

Question:

Implement a class Huffman which contains the root Node of Huffman tree. Each node in Huffman is of type HNode.

```
struct HNode
{   int freq;   char
character;   HNode
*left, *right;
}
```

Note:

a. A valid character is stored only in leaf nodes in a Huffman Tree.
b. In a leaf node, the value in freq variable corresponds to the frequency of the character that the node represents.
c. In a non-leaf node, the value in freq variable is the sum of freq variables stored in its left child and right child.

1. Implement a member function createHuffman that is passed as parameter a filename. The file contains characters along with their frequencies. The function creates a Huffman tree of the given characters. You will need MinHeap data structure for this. Use your previous MinHeap implementation. `void createHuffman(string const filename)`

Note: Min Heap Code is provided.

Steps to build Huffman Tree

Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.

1. *Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)*

2. *Extract two nodes with the minimum frequency from the min heap.*

3. *Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.*

4. *Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.*

2. Implement a member function printHuffman which prints the Huffman code of each character along with the character and its frequency. `printHuffman() const`
3. Destructor

4. Create a main function which creates a Huffman object, calls the createHuffman function, and then calls the printHuffman function to print Huffman code.