Behjat Bahmani/Mohammed Deeb

Professor Suporn Chenhansa

CS-124-01

8 April 2021

Lab 4

Purpose:

The purpose of this assignment is to know more about hash tables and collision management and try to handle this problem by collecting all colliding in one way: separate chaining(collecting all colliding values in linked-list) or open addressing . We will also be practicing using binary trees, inserting values, and using decisions to access values. At the end will practice avoiding memory leaks and using valgrind. We used the “RAII” (Resource acquisition is initialization) concept to solve memory-leak in my program. We will also be practicing reading input from text files and writing output into text files

Plan:

Input:

1. User chooses if he/she wants to try or not (Yes or No). (stored in a string variable)
2. User chooses if he/she wants to enter a file encode (from English to Morse code) or Decode (from Morse code to English) . (stord in an int variable)
3. If the user chooses to encode or decode and then he will enter the text file. (stored in a string)
4. Users choice of the output txt file. (stored in a string variable)

5. Users choice of repeating the program (yes/no). (stored in a string variable)

Variables:

1. String run: stores users choice of running the program or repeating the program
2. Int choice: stores users choice of encoding or decoding
3. String inputFile: stores user's input of input file name
4. String outputFile: stores users input of output file
5. HashTable encodeTable: stores the morse codes
6. BinaryTree decodeTable: stores the english chars
7. vector<vector<char>> input: stores the input of the user
8. vector<vector<char>> output: stores the results of the program

Classes:

HashTable: class for hash table

Iterator: class for iterating through HashTables

List: linked list class to be used in hashtable class, because we will be using separate chaining

Node: this is a node class that will be used in List class

BinaryTree: class for binary tree will be used to store characters as a data and '-', '.' as decisions

Node: this will be a node class for the BinaryTree

Functions:

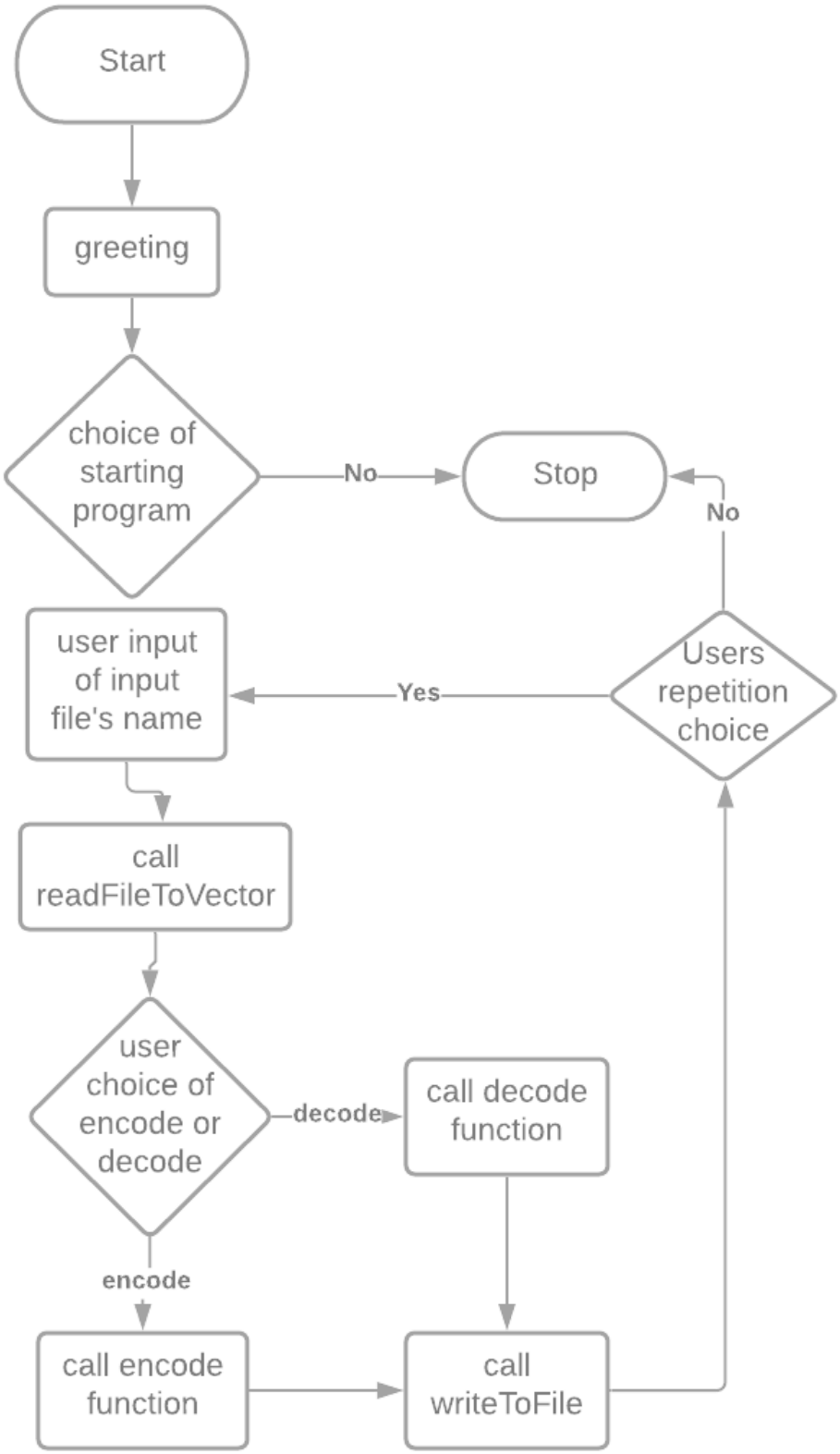
1. Int main(): This is the main function and starting point of our program.
2. Void greet(): greeting and introducing program.

3. Void readFileToVector(vector<vector<char>> & input, string fileName): this function will read the input files and store the characters in each line as chars in the vector
4. Void writeVectorToFile(vector<vector<char>> & input, string fileName): this function writes the results into a file from user's choice.
5. Void BinaryTree::insert (string engChar, string morseCode): this is a member function of binaryTree that inserts the engChar in the binarySearch tree in the right place (following the path that is determined by the morse code.
6. Void BinaryTree::decodeChar(string morse, char & result): This is a member function of class BinaryTree, it modifies the "result" parameter to contain the result of decoding the string morse parameter. Using binary trees is best for decoding because it will be easy and quick to follow the path when the decisions(morse codes) are given. However, its wouldn't be efficient to encode, because since we don't have the decisions ('.', '-'), we would have to trace through the tree to find the character then save the decisions into a string), this wouldn't be efficient.
7. Void HashTable::insert (string engChar, string morseCode): this is a member function of HashTable class, it inserts the morseCode value in the right position (the position is determined by using hashFunction with parameter endChar)
8. Void HashTable::encodeChar(char letter, string & result): This is a member function of class HashTable, search for the letter in the hashtable and modify the result variable to contain the morse code for the letter. It would be best to use hashtable for encoding, because if the hash table stored the morse codes for all character, it would be possible to create a hash function that take a and english char and find a special hash code to be used as an index of the element that stores the morse code of that character. In this case the

array must be the size of the number of all characters that has a morse code. There will be no reason for compressing because the hash function will return a hash code in the range of the array anyway. And no collisions will occur since each character has its own hash code.

9. Void encode(HashTable cipher, vector<vector<char>> input, vector<vector<char>> &result): this is a function that iterates through the input vector, and uses encodeChar function to encode characters separately and modify the result vector to contain the result (morse code).
10. Void decode(BinaryTree cipher, vector<vector<char>> input, vector<vector<char>> &result): this is a function that iterates through the input vector, and uses decodeChar function to decode morse code characters separately and modify the result vector to contain the results (english).
11. Int hashFunction(char c): this function returns a unique hash code for the char it takes as an argument.
12. Void vectorToFile(vector<vector<char>> list, string fileName): this function writes the continent of the vector into a txt file .

To determine if the program is complete we will test each function separately right after it is created. Then make a cumulative test case once the program is done, trying different files.


Start
greeting
choice of starting program
No
Stop
No
user input of input file's name
Yes
Users repetition choice
call readFileToVector
user choice of encode or decode
decode
call decode function
encode
call encode function
call writeToFile