DATA Structures FINAL PROJECT

# CODES:

## source file(.cpp):

|  |
| --- |
| #include<iostream>  #include<fstream>  #include<string>  #include<conio.h>  #include<cstdio>  #include<Windows.h>  #include"implementations.h"  using namespace std;  int main() {//MAIN FUNCTION  Trie\_tree obj\_trie;  Huffman obj\_huff;  int option = 0;  string opt = "";//VARIABLE DECLARATIONS  animation();//ANIMATION FUNCTION CALL  do {//SELECTION OF MODE  system("cls");  cout << "Select Screen Mode!" << endl << endl;  cout << "1) Light Mode" << endl;  cout << "2) Dark Mode" << endl;  cout << "Enter Here : ";  cin >> opt;  } while (opt > "2" || opt < "1");  if (opt == "1") {  system("color 70");  }  else {  system("color 07");  }  if (obj\_trie.load\_trie\_tree("dictionary")) {//MAIN MENU  do {  system("cls");  cout << "Welcome To Text Editor In CPP!" << endl << endl;  cout << "0) To Exit" << endl;  cout << "1) To Enter Writing Mode" << endl;  cout << "2) To Enter Display Mode" << endl;  cout << "3) To Enter Deletion Mode" << endl;  cout << "Enter Here : ";  cin >> option;  switch (option) {  case 0: {  system("cls");  cout << "Thanks For Using The Program!" << endl;  exit(0);  break;  }  case 1: {  string opt = "";  do {  system("cls");  cout << "Welcome To Writing Mode!" << endl << endl;  cout << "Please Select The Mode!" << endl;  cout << "0) Write In The Saved File" << endl;  cout << "1) Write In the New File" << endl;  cout << "2) To Exit" << endl;  cout << "Enter Here : ";  cin >> opt;  if (opt < "0" || opt > "2") {  cout << "Invalid Input!" << endl;  system("pause");  }  } while (opt < "0" || opt > "2");  if (option == 2) {  break;  }  else write\_mode(obj\_huff, obj\_trie, option);  break;  }  case 2: {  cout << "Welcome To Display Mode!" << endl << endl;  display\_mode(obj\_huff);  break;  }  case 3: {  deletion\_mode();  break;  }  }  } while (true);  }  else {  cout << "Suggestions Loading Error!" << endl;  }  system("pause");  return 0;  } |

## prototypes file (.h):

|  |
| --- |
| #pragma once  #pragma warning(suppress : 4996)  #include<iostream>  #include<fstream>  #include<string>  #include<conio.h>  #include<cstdio>  #include<Windows.h>  using namespace std;  //ALL CLASSES AND FUNCTIONS PROTOTYPES  //HUFFMAN NODE CLASS  struct Huff\_node {  char data;  int counter;  string code\_str;  Huff\_node\* next, \* left, \* right;  Huff\_node();  };  //HUFFMAN TREE CLASS  class Huffman {  private:  Huff\_node\* front;  void assign\_code(Huff\_node\* current, string str);  void inorder(Huff\_node\* current);  Huff\_node\* delete\_huff(Huff\_node\* current);  void encode\_alpha(char alpha, Huff\_node\* current, string& code);  void decode\_alpha(Huff\_node\* curr, string coded\_str, string& uncoded, int& index);  void store\_encoded(string& coded\_str, string file\_name);  bool retrieve\_encoded(string& code\_str, string file\_name);  bool is\_empty();  Huff\_node\* get\_top();  void make\_huff();  void enqueue(char value, int num, Huff\_node\* l, Huff\_node\* r);  void dequeue();  void delete\_tree();  void make\_list(string str);  void store\_encryption(string file\_name);  bool retrieve\_encryption(string file\_name);  void encode\_str(string& str);  void decode\_str(string& str);  public:  Huffman();  void display();  void store\_data(string file, string data);  string retrieve\_data(string file);  };  //TRIE NODE CLASS  struct Trie\_node {  bool leaf;  Trie\_node\* next[26];  Trie\_node();  };  //TRIE TREE CLASS  class Trie\_tree {  Trie\_node\* head;  void pre\_order(Trie\_node\* n, string& word, string\*& arr, int& s, int& iter, int num);  public:  Trie\_tree();  void insertion(string word);  bool load\_trie\_tree(string file\_name);  void suggestion(string start, string\*& word, int& size\_str);  };  //GENERIC FUNCTIONS PROTOTYPES  int char\_to\_int(char data);  void alphabet\_counter(string str, int\*& arr);  bool check\_special\_char(string& x);  bool check\_input(char data);  char int\_to\_char(int data);  void check\_case\_char(char& data);  void check\_case\_str(string& str);  void animation();  void input\_data(string& final\_str, Trie\_tree& obj);  void write\_mode(Huffman& huff, Trie\_tree& trie, int mode);  void display\_mode(Huffman& huff);  void deletion\_mode(); |

## implementations file(.h):

|  |
| --- |
| #pragma once  #pragma warning(suppress : 4996)  #include<iostream>  #include<fstream>  #include<string>  #include<conio.h>  #include<cstdio>  #include<Windows.h>  #include"prototypes.h"  using namespace std;  //ALL CLASSES AND FUNCTIONS IMPLEMENTATIONS  //HUFF NODE FUNCTIONS AND CONSTRUCTORS  Huff\_node::Huff\_node() {  data = counter = 0;  next = left = right = NULL;  code\_str = "";  }  //HUFFMAN TREE FUNCTIONS AND CONSTRUCTORS  void Huffman::assign\_code(Huff\_node\* current, string str) {  if (current != NULL) {  if (current->left == NULL && current->right == NULL) {  current->code\_str = str;  }  assign\_code(current->left, str + '0');  assign\_code(current->right, str + '1');  }  }  void Huffman::inorder(Huff\_node\* current) {  if (current != NULL) {  inorder(current->left);  cout << current->data << " " << current->counter << " " << current->code\_str << endl;  inorder(current->right);  }  }  Huff\_node\* Huffman::delete\_huff(Huff\_node\* current) {  if (current != NULL) {  current->left = delete\_huff(current->left);  current->right = delete\_huff(current->right);  delete current;  current = NULL;  return current;  }  else return NULL;  }  void Huffman::encode\_alpha(char alpha, Huff\_node\* current, string& code) {  if (current != NULL) {  if (current->data == alpha) {  code = current->code\_str;  }  else {  encode\_alpha(alpha, current->left, code);  encode\_alpha(alpha, current->right, code);  }  }  }  void Huffman::decode\_alpha(Huff\_node\* curr, string coded\_str, string& uncoded, int& index) {  if (curr == NULL) {  return;  }  if (curr->left == NULL && curr->right == NULL)  {  uncoded += curr->data;  return;  }  if (coded\_str[index] == '0')  {  index += 1;  decode\_alpha(curr->left, coded\_str, uncoded, index);  }  else  {  index += 1;  decode\_alpha(curr->right, coded\_str, uncoded, index);  }  return;  }  void Huffman::store\_encoded(string& coded\_str, string file\_name) {  file\_name += "compressed.txt";  ofstream fout(file\_name);  fout << coded\_str;  }  bool Huffman::retrieve\_encoded(string& code\_str, string file\_name) {  file\_name += "compressed.txt";  ifstream fin(file\_name);  if (fin.is\_open()) {  int counter = 0;  char temp;  while (!fin.eof()) {  fin >> temp;  counter++;  }  fin.close();  if (counter > 1) {  fin.open(file\_name);  fin >> code\_str;  fin.close();  return true;  }  }  else return false;  }  bool Huffman::is\_empty() {  return (front == NULL);  }  Huff\_node\* Huffman::get\_top() {  return front;  }  void Huffman::make\_huff() {  Huff\_node\* l = NULL, \* r = NULL, \* temp = NULL;  int sum = 0;  if (is\_empty()) {  return;  }  while (front->next != NULL) {  temp = get\_top();  r = new Huff\_node;  l = new Huff\_node;  l->data = temp->data;  l->counter = temp->counter;  l->left = temp->left;  l->right = temp->right;  l->next = NULL;  dequeue();  temp = get\_top();  r->data = temp->data;  r->counter = temp->counter;  r->left = temp->left;  r->right = temp->right;  r->next = NULL;  dequeue();  sum = l->counter + r->counter;  enqueue('\0', sum, l, r);  }  assign\_code(front, "");  }  void Huffman::enqueue(char value, int num, Huff\_node\* l, Huff\_node\* r) {  Huff\_node\* temp = new Huff\_node;  temp->data = value;  temp->counter = num;  temp->right = r;  temp->left = l;  if (is\_empty()) {  front = temp;  }  else {  Huff\_node\* current = front;  while (current->next != NULL && current->next->counter <= num) {  current = current->next;  }  if (current->counter > num)  {  temp->next = current;  front = temp;  current = temp;  }  else  {  temp->next = current->next;  current->next = temp;  }  }  }  void Huffman::dequeue() {  if (is\_empty()) {  return;  }  else {  Huff\_node\* current = front;  front = front->next;  delete current;  current = NULL;  }  }  void Huffman::delete\_tree() {  front = delete\_huff(front);  }  void Huffman::make\_list(string str) {  int\* arr = new int[30];  for (int i = 0; i < 30; i++) {  arr[i] = 0;  }  alphabet\_counter(str, arr);  for (int i = 0; i < 30; i++) {  if (arr[i] != 0) {  if (i == 26) {  enqueue(' ', arr[i], NULL, NULL);  }  else if (i == 27) {  enqueue('.', arr[i], NULL, NULL);  }  else if (i == 28) {  enqueue(',', arr[i], NULL, NULL);  }  else if (i == 29) {  enqueue('?', arr[i], NULL, NULL);  }  else {  int temp = int\_to\_char(i);  enqueue(temp, arr[i], NULL, NULL);  }  }  }  }  void Huffman::store\_encryption(string file\_name) {  file\_name += "code.txt";  ofstream fout(file\_name);  Huff\_node\* current = front;  while (current != NULL) {  if (current->data == ' ') {  fout << '!';  }  else {  fout << current->data;  }  fout << current->counter;  current = current->next;  }  fout.close();  }  bool Huffman::retrieve\_encryption(string file\_name) {  file\_name += "code.txt";  ifstream fin(file\_name);  if (fin.is\_open()) {  char alpha;  int num = 0;  char temp;  while (!fin.eof()) {  fin >> temp;  num++;  }  fin.close();  if (num > 1) {  fin.open(file\_name);  while (!fin.eof()) {  fin >> alpha;  if (alpha == '!') {  alpha = ' ';  }  fin >> num;  enqueue(alpha, num, NULL, NULL);  }  fin.close();  }  return true;  }  else {  return false;  }  }  void Huffman::encode\_str(string& str) {  string code\_str = "", temp;  for (int i = 0; i < str.length(); i++) {  temp = "";  encode\_alpha(str[i], get\_top(), temp);  code\_str += temp;  }  str = code\_str;  }  void Huffman::decode\_str(string& str) {  string temp = "";  int i = 0;  while (i < str.length()) {  decode\_alpha(get\_top(), str, temp, i);  }  str = temp;  }  Huffman::Huffman() {  front = nullptr;  }  void Huffman::display() {  if (is\_empty()) {  cout << "Tree Is Empty!" << endl;  }  else {  cout << "Displaying The Huffman Tree ::" << endl << endl;  inorder(front);  }  }  void Huffman::store\_data(string file, string data) {  make\_list(data);  store\_encryption(file);  make\_huff();  encode\_str(data);  store\_encoded(data, file);  delete\_tree();  }  string Huffman::retrieve\_data(string file) {  string str;  if (retrieve\_encryption(file)) {  make\_huff();  if (retrieve\_encoded(str, file)) {  decode\_str(str);  delete\_tree();  return str + " ";  }  else {  return "";  }  }  else {  return "";  }  }  //TRIE NODE FUNCTIONS AND CONSTRUCTORS  Trie\_node::Trie\_node() {  leaf = false;  for (int i = 0; i < 26; i++) {  next[i] = NULL;  }  }  //TRIE TREE FUNCTIONS AND CONSTRUCTORS  void Trie\_tree::pre\_order(Trie\_node\* n, string& word, string\*& arr, int& s, int& iter, int num) {  if (iter != s) {  if (n != NULL) {  word += int\_to\_char(num);  if (n->leaf == true) {  arr[iter] = word;  iter++;  }  for (int i = 0; i < 25; i++) {  pre\_order(n->next[i], word, arr, s, iter, i);  }  string temp\_word = "";  for (int i = 0; i < word.length() - 1; i++) {  temp\_word += word[i];  }  word = temp\_word;  }  }  }  Trie\_tree::Trie\_tree() {  head = new Trie\_node;  }  void Trie\_tree::insertion(string word) {  Trie\_node\* current = head;  for (int i = 0, index; true; i++) {  check\_case\_char(word[i]);  index = char\_to\_int(word[i]);  if (current->next[index] == NULL) {  current->next[index] = new Trie\_node;  }  current = current->next[index];  if (i == word.length() - 1) {  current->leaf = true;  break;  }  }  }  bool Trie\_tree::load\_trie\_tree(string file\_name) {  file\_name += ".txt";  string temp;  ifstream words(file\_name);  if (words.is\_open()) {  for (int i = 0; !words.eof(); i++) {  words >> temp;  check\_special\_char(temp);  insertion(temp);  }  words.close();  return true;  }  else return false;  }  void Trie\_tree::suggestion(string start, string\*& word, int& size\_str) {  int iterator = 0, index = 0;  bool flag = true;  string temp;  Trie\_node\* current = head;  for (int i = 0; i < start.length() - 1; i++) {  if (current->next[char\_to\_int(start[i])] == NULL) {  flag = false;  break;  }  current = current->next[char\_to\_int(start[i])];  temp += start[i];  }  if (flag) {  index = char\_to\_int(start[start.length() - 1]);  pre\_order(current->next[index], temp, word, size\_str, iterator, index);  }  int count = 0;  while (word[count] != "" && count < size\_str) {  count++;  }  size\_str = count;  }  //GENERIC FUNCTIONS  int char\_to\_int(char data) {  return (data - 'a');  }  void alphabet\_counter(string str, int\*& arr)  {  for (int i = 0; i < 30; i++)  {  arr[i] = 0;  }  for (int i = 0; i < str.size(); i++)  {  if (str[i] == ' ')  {  arr[26] += 1;  continue;  }  else if (str[i] == '.')  {  arr[27] += 1;  continue;  }  else if (str[i] == ',')  {  arr[28] += 1;  continue;  }  else if (str[i] == '?')  {  arr[29] += 1;  continue;  }  else {  arr[char\_to\_int(str[i])] += 1;  }  }  }  bool check\_special\_char(string& x) {  for (int i = 0; i < x.length(); i++) {  if (x[i] >= 65 && x[i] <= 91 || x[i] >= 97 && x[i] <= 123) {  continue;  }  else {  return true;  }  }  return false;  }  bool check\_input(char data) {  if (('a' <= data && 'z' >= data) || ('A' <= data && 'Z' >= data) || data == ' ' || data == ';' || data == '/' || data == '=' || data == '.' || data == ',' || data == '?') {  return true;  }  else return false;  }  char int\_to\_char(int data) {  return ('a' + data);  }  void check\_case\_char(char& data) {  if (data >= 65 && data <= 91) {//small case ascii check  data += 32;  }  }  void check\_case\_str(string& str) {  for (int i = 0; i < str.length(); i++) {  check\_case\_char(str[i]);  }  }  void animation() {  string animate1 = R"(  $$$$$$$$\ $$$$$$$$\ $$\ $$\ $$$$$$$$\ $$$$$$$$\ $$$$$$$\ $$$$$$\ $$$$$$$$\ $$$$$$\ $$$$$$$\  \\_\_$$ \_\_|$$ \_\_\_\_\_|$$ | $$ |\\_\_$$ \_\_| $$ \_\_\_\_\_|$$ \_\_$$\ \\_$$ \_|\\_\_$$ \_\_|$$ \_\_$$\ $$ \_\_$$\  $$ | $$ | \$$\ $$ | $$ | $$ | $$ | $$ | $$ | $$ | $$ / $$ |$$ | $$ |  $$ | $$$$$\ \$$$$ / $$ | $$$$$\ $$ | $$ | $$ | $$ | $$ | $$ |$$$$$$$ |  $$ | $$ \_\_| $$ $$< $$ | $$ \_\_| $$ | $$ | $$ | $$ | $$ | $$ |$$ \_\_$$<  $$ | $$ | $$ /\$$\ $$ | $$ | $$ | $$ | $$ | $$ | $$ | $$ |$$ | $$ |  $$ | $$$$$$$$\ $$ / $$ | $$ | $$$$$$$$\ $$$$$$$ |$$$$$$\ $$ | $$$$$$ |$$ | $$ |  \\_\_| \\_\_\_\_\_\_\_\_|\\_\_| \\_\_| \\_\_| \\_\_\_\_\_\_\_\_|\\_\_\_\_\_\_\_/ \\_\_\_\_\_\_| \\_\_| \\_\_\_\_\_\_/ \\_\_| \\_\_|        )";  string animate2 = R"(  $$$$$$\ $$\ $$$$$$$$\ $$$$$$\ $$$$$$\ $$$$$$\ $$$$$$$$\  $$ \_\_$$\ $$$$ | $$ \_\_\_\_\_| $$ \_\_$$\ $$ \_\_$$\ $$$ \_\_$$\ \\_\_\_\_$$ |  \\_\_/ $$ |\\_$$ | $$ | $$ / $$ |\\_\_/ $$ |$$$$\ $$ | $$ /  $$$$$$ | $$ | $$$$$\ $$$$$$\\$$$$$$$ | $$$$$$ |$$\$$\$$ | $$ /  $$ \_\_\_\_/ $$ | $$ \_\_|\\_\_\_\_\_\_|\\_\_\_\_$$ |$$ \_\_\_\_/ $$ \$$$$ | $$ /  $$ | $$ | $$ | $$\ $$ |$$ | $$ |\$$$ | $$ /  $$$$$$$$\ $$$$$$\ $$ | \$$$$$$ |$$$$$$$$\ \$$$$$$ /$$ /  \\_\_\_\_\_\_\_\_|\\_\_\_\_\_\_|\\_\_| \\_\_\_\_\_\_/ \\_\_\_\_\_\_\_\_| \\_\_\_\_\_\_/ \\_\_/        $$$$$$\ $$\ $$$$$$$$\ $$$$$$\ $$\ $$$$$$$$\ $$\  $$ \_\_$$\ $$$$ | $$ \_\_\_\_\_| $$ \_\_$$\ $$$$ | \\_\_\_\_$$ |$$$$ |  \\_\_/ $$ |\\_$$ | $$ | $$ / $$ |\\_$$ | $$ / \\_$$ |  $$$$$$ | $$ | $$$$$\ $$$$$$\\$$$$$$$ | $$ | $$ / $$ |  $$ \_\_\_\_/ $$ | $$ \_\_|\\_\_\_\_\_\_|\\_\_\_\_$$ | $$ | $$ / $$ |  $$ | $$ | $$ | $$\ $$ | $$ | $$ / $$ |  $$$$$$$$\ $$$$$$\ $$ | \$$$$$$ |$$$$$$\ $$ / $$$$$$\  \\_\_\_\_\_\_\_\_|\\_\_\_\_\_\_|\\_\_| \\_\_\_\_\_\_/ \\_\_\_\_\_\_|\\_\_/ \\_\_\_\_\_\_|        $$$$$$\ $$\ $$$$$$$$\ $$$$$$\ $$\ $$$$$$\ $$$$$$$$\  $$ \_\_$$\ $$$$ | $$ \_\_\_\_\_| $$ \_\_$$\ $$$$ | $$$ \_\_$$\ \\_\_\_\_$$ |  \\_\_/ $$ |\\_$$ | $$ | $$ / $$ |\\_$$ | $$$$\ $$ | $$ /  $$$$$$ | $$ | $$$$$\ $$$$$$\\$$$$$$$ | $$ | $$\$$\$$ | $$ /  $$ \_\_\_\_/ $$ | $$ \_\_|\\_\_\_\_\_\_|\\_\_\_\_$$ | $$ | $$ \$$$$ | $$ /  $$ | $$ | $$ | $$\ $$ | $$ | $$ |\$$$ | $$ /  $$$$$$$$\ $$$$$$\ $$ | \$$$$$$ |$$$$$$\\$$$$$$ /$$ /  \\_\_\_\_\_\_\_\_|\\_\_\_\_\_\_|\\_\_| \\_\_\_\_\_\_/ \\_\_\_\_\_\_|\\_\_\_\_\_\_/ \\_\_/        )";  cout << animate1 << endl;  system("pause");  system("cls");  cout << animate2 << endl;  system("pause");  }  void input\_data(string& final\_str, Trie\_tree& obj) {  system("cls");  string\* suggestion\_arr = NULL, user = "";  int size\_str = 0;  char c = ' ';  bool flag = true;  suggestion\_arr = new string[10];  cout << final\_str;  while (flag) {  do {  c = \_getch();  } while (!check\_input(c));  if (c != '/' && c != ' ' && c != '=') {  for (int i = 0; i < 10; i++) {  suggestion\_arr[i] = "";  }  system("cls");  user += c;  cout << final\_str;  cout << user;  size\_str = 10;  check\_case\_str(user);  obj.suggestion(user, suggestion\_arr, size\_str);  cout << endl;  for (int i = 0; i < size\_str; i++) {  cout << suggestion\_arr[i] << endl;  }  }  else if (c == ' ' || c == ',' || c == '.' || c == '?') {  final\_str += user + c;  user = "";  }  else if (c == '=') {  final\_str += user;  flag = false;  break;  }  else {  for (int i = 0; c == '/' && i < size\_str; i++) {  system("cls");  for (int j = 0; j < size\_str; j++) {  cout << suggestion\_arr[j];  if (j == i) {  cout << " <-";  }  cout << endl;  }  do {  c = \_getch();  } while (!check\_input(c));  if (c == ';') {  if (suggestion\_arr[i] != " ") {  user = suggestion\_arr[i];  system("cls");  cout << final\_str;  cout << user;  break;  }  }  }  }  }  }  void write\_mode(Huffman& huff, Trie\_tree& trie, int mode) {  string user = "", file\_name;  ifstream fin;  system("cls");  cout << "Enter The File Name : ";  cin >> file\_name;  fin.open(file\_name + "code.txt");  if (fin.is\_open() || mode) {  user = huff.retrieve\_data(file\_name);  input\_data(user, trie);  huff.store\_data(file\_name, user);  cout << "\nFile Saved Successfuly!" << endl;  }  else {  cout << "File Can Not Be Opened!" << endl;  }  system("pause");  }  void display\_mode(Huffman& huff) {  system("cls");  string display\_str, file\_name;  ifstream fin;  cout << "Enter The File Name : ";  cin >> file\_name;  fin.open(file\_name + "code.txt");  if (fin.is\_open()) {  display\_str = huff.retrieve\_data(file\_name);  system("cls");  cout << "Your File Data Is As Following ::" << endl << endl;  cout << display\_str << endl;  }  else {  cout << "File Does Not Exist!" << endl;  }  system("pause");  }  void deletion\_mode() {  system("cls");  string file\_name = "";  ifstream fin;  cout << "Enter The File Name : ";  cin >> file\_name;  fin.open(file\_name + "code.txt");  if (fin.is\_open()) {  fin.close();  string temp = file\_name + "code.txt";  char\* file\_arr = new char[temp.length() + 1];  for (int i = 0; i < temp.length(); i++) {  file\_arr[i] = temp[i];  }  file\_arr[temp.length()] = '\0';  remove(file\_arr);  delete[] file\_arr;  temp = file\_name + "compressed.txt";  file\_arr = new char[temp.length() + 1];  for (int i = 0; i < temp.length(); i++) {  file\_arr[i] = temp[i];  }  file\_arr[temp.length()] = '\0';  remove(file\_arr);  delete[] file\_arr;  file\_arr = NULL;  cout << "File Deleted Successfuly!" << endl;  }  else {  cout << "File Does Not Exist!" << endl;  }  system("pause");  } |

# console:





































