 ****

**Department of Computer Science**

# National University of Computer & Emerging Sciences



Name: **Husnain Mahmood**

Roll No(s): **21F-9624**

Section: **BCS-3C**

Subject: **Data Structures** **Lab**

# SEMESTER FINAL PROJECT

TEXT EDITOR using C++

#include<iostream>

#include<string>

#include<conio.h>

#include<queue>

#include<iomanip>

#include<fstream>

using namespace std;

ofstream myfile;

int getIndex(char c) //tells the index number of the specfic character

{

char a = 'a';

for (int i = 0; i < 26; i++)

{

if (a == c)

return i;

else

a++;

}

cout << "Invalid input for getindex Function\n";

return -1;

}

void Convert(char& ch) // From Capital to Small Letters

{

if (ch >= 'A' && ch <= 'Z')

{

ch = char(ch + 32);

cout << ch << endl;

}

return;

}

void Minus(string& a)

{

for (int i = 0; i < a.length(); i++)

{

a[i] = a[i + 1];

}

a.resize(a.length() - 1);

}

struct Huff\_node {

char data;

int counter;

string code\_str;

Huff\_node\* next, \* left, \* right;

Huff\_node() {

data = counter = 0;

next = left = right = NULL;

code\_str = "";

}

};

class Huffman {

private:

Huff\_node\* front;

public:

Huffman() {

front = NULL;

}

bool isEmpty() {

return (front == NULL);

}

Huff\_node\* top() {

return front;

}

char top\_char() {

return front->data;

}

int top\_int() {

return front->counter;

}

void make\_huff() {

Huff\_node\* l = NULL, \* r = NULL, \* temp = NULL;

int sum = 0;

if (isEmpty()) {

return;

}

while (front->next != NULL) {

temp = 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 = 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);

}

encode(front, "");

}

void 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 (isEmpty()) {

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 dequeue() {

if (isEmpty()) {

return;

}

else {

Huff\_node\* current = front;

front = front->next;

delete current;

current = NULL;

}

}

void display(Huff\_node\* current) {

if (current != NULL) {

display(current->left);

cout << current->data << " " << current->counter << " " << current->code\_str << endl;

display(current->right);

}

}

void encode(Huff\_node\* current, string str) {

if (current != NULL) {

if (current->left == NULL && current->right == NULL) {

current->code\_str = str;

}

encode(current->left, str + '0');

encode(current->right, str + '1');

}

}

};

void find\_code(char alpha, Huff\_node\* current, string& code) {

if (current != NULL) {

if (current->data == alpha) {

code = current->code\_str;

}

else {

find\_code(alpha, current->left, code);

find\_code(alpha, current->right, code);

}

}

}

string get\_code(Huffman obj, string str) {

string code\_str = "", temp;

for (int i = 0; i < str.length(); i++) {

temp = "";

find\_code(str[i], obj.top(), temp);

code\_str += temp;

}

return code\_str;

}

void decoder(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;

decoder(curr->left, coded\_str, uncoded, index);

}

else

{

index += 1;

decoder(curr->right, coded\_str, uncoded, index);

}

return;

}

string decoded\_str(Huffman obj, string coded\_str)

{

string uncoded = "";

int i = 0;

while (i < coded\_str.length())

{

decoder(obj.top(), coded\_str, uncoded, i);

}

return uncoded;

}

int char\_to\_int(char data) {

return (data - 'a');

}

char int\_to\_char(int data) {

return ('a' + data);

}

void alphabet\_counter(string str, int\*& arr)

{

for (int i = 0; i < 27; i++)

{

arr[i] = 0;

}

for (int i = 0; i < str.size(); i++)

{

if (str[i] == ' ')

{

arr[26] += 1;

continue;

}

arr[char\_to\_int(str[i])] += 1;

}

}

void make\_LL(Huffman& obj, int\* arr)

{

for (int i = 0; i < 27; i++)

{

if (arr[i] != 0)

{

if (i == 26)

{

obj.enqueue(' ', arr[i], NULL, NULL);

}

else

{

obj.enqueue(int\_to\_char(i), arr[i], NULL, NULL);

}

}

}

}

void store\_LL(Huff\_node\* curr) {

ofstream fout("sorted.txt");

while (curr != NULL) {

if (curr->data == ' ')

{

fout << '!';

}

else

{

fout << curr->data;

}

fout << curr->counter;

curr = curr->next;

}

fout.close();

}

void read\_LL(Huffman& obj)

{

ifstream fin("sorted.txt");

char alpha;

int num;

while (!fin.eof())

{

fin >> alpha;

if (alpha == '!')

{

alpha = ' ';

}

fin >> num;

obj.enqueue(alpha, num, NULL, NULL);

}

}

void store\_encoded(string coded\_str)

{

ofstream fout("compressed.txt");

fout << coded\_str;

}

string get\_encoded()

{

string temp;

ifstream fin("compressed.txt");

fin >> temp;

return temp;

}

class Node

{

public:

char data;

Node\* child[26];

bool isComplete;

Node(char data);

bool Compare(char c, int i);

};

Node::Node(char data)

{

this->data = data;

this->isComplete = false;

for (int i = 0; i < 26; i++)

{

child[i] = NULL;

}

}

bool Node::Compare(char c, int i)

{

if (this->child[i] == NULL)

return false;

if (this->child[i]->data == c)

return true;

else

return false;

}

class TrieTree

{

private:

Node\* root; //Root of the TrieTree

bool file\_loaded; //isComplete to tell file is loaded or not

void Insert(Node\* root, string word)const;

public:

TrieTree(); //default constructor

Node\* get\_root(); //function will give the root of the tree

void insert(string word);

bool readfile(); //function to read from the file

bool search\_for\_suggestions(string word, queue<string>& my\_que); //function to pass a specfic child Node to the recurrsive function

void suggesions(Node\* ptr, int i, char str[], int& count, queue<string>& my\_que)const; //recurrsive function to find suggessions

};

TrieTree::TrieTree()

{

root = new Node(' ');

file\_loaded = false;

}

Node\* TrieTree::get\_root()

{

return root;

}

void TrieTree::insert(string word)

{

Insert(this->root, word);

}

void TrieTree::Insert(Node\* root, string word)const //function to insert in the the word

{

if (word.length() == 0)

{

return;

}

if (word.length() == 1)

root->isComplete = true;

int index = getIndex(word[0]);

Convert(word[0]);

if (root->Compare(word[0], index) == true)

{

Minus(word);

Insert(root->child[index], word);

}

else

{

root->child[index] = new Node(word[0]);

Minus(word);

return Insert(root->child[index], word);

}

}

bool TrieTree::readfile() //function to read from the file

{

ifstream infile;

string str;

infile.open("dictionary.txt"); //open file to read

if (!infile.is\_open()) //if file not loaded

{

return false;

}

file\_loaded = true; //tell the status file load

while (!infile.eof())

{

str = "";

getline(infile, str); //read word till tab

insert(str);

}

infile.close(); //close the file

return true;

}

bool TrieTree::search\_for\_suggestions(string word, queue<string>& my\_que)

{

int word\_length = word.size();

Node\* curr = root; //pointer to itrate

for (int i = 0; i < word\_length; i++)

{

if (getIndex(word[i]) == -1) //invalid character

{

return false;

}

if (curr->child[getIndex(word[i])] != NULL)

{

curr = curr->child[getIndex(word[i])];

if (i == word\_length - 1) //going to the specfic substring child

{

int count = 0;

char strarr[70];

for (int i = 0, k = 0; i < 70; i++, k++) //coping the substring and larger index are make null

{

if (k < word\_length) {

strarr[i] = word[k];

}

else

strarr[i] = '\0';

}

//passing curr to specfic child

suggesions(curr, word\_length - 1, strarr, count, my\_que);

}

}

else

{

//if the substring is not present

return false;

}

}

}

//recurrsive function to find suggessions

void TrieTree::suggesions(Node\* ptr, int i, char str[], int& count, queue<string>& my\_que)const

{

if (ptr != NULL && count < 10)

{

str[i] = ptr->data;

i++;

//if the word is present

if (ptr->isComplete == true)

{

count++;

//then enqueue the word in the queue

my\_que.push(str);

}

//calling recurrsive function for each child

for (int k = 0; k < 26; k++)

{

suggesions(ptr->child[k], i, str, count, my\_que);

}

str[i] = '\0';

i--;

}

else

return;

}

void Continue\_Editing()

{

system("Color B5");

cout << "Enter your text: ";

string\* myList = new string[10];

TrieTree trie\_ptr;

Huffman huff;

queue<string> que;

int size = 27;

int\* arr;

arr = new int[size];

trie\_ptr.readfile();

string str, curr = "", sr = "";

vector<string> vec;

ifstream filetemp("open.txt"), currfile("open.txt");

int i = 0, cp = 0;

while (!(filetemp.eof()))

{

getline(filetemp, sr);

i++;

}

cout << i << endl;

if (i == 1)

{

while (!(currfile.eof()))

{

getline(currfile, curr);

str = curr;

}

}

else

{

while (!(currfile.eof()))

{

if (cp == i - 1)

{

getline(currfile, curr);

str = curr;

break;

}

else if (cp < i)

{

getline(currfile, curr);

vec.push\_back(curr);

}

cp++;

}

}

cout << "Enter your text: ";

for (int i = 0; i < vec.size(); i++)

{

cout << vec[i] << " ";

}

cout << str;

int count = 0, inc = 0;

while (true)

{

if (count == 10000)

break;

get:

count++;

char ch = \_getch();

Convert(ch);

if (ch == ' ')

{

vec.push\_back(str);

str = "";

}

else if (ch == ';')

{

if (str == "") {

continue;

}

else if (curr != "" && inc < 10)

{

inc = 0;

str = curr;

system("cls");

myfile.open("open.txt", ios::out);

myfile.close();

myfile.open("open.txt");

cout << "Enter your text: ";

for (int i = 0; i < vec.size(); i++)

{

cout << vec[i] << " ";

myfile << vec[i] << " ";

if (i == vec.size() - 1)

myfile << endl;

}

cout << str;

myfile << str;

trie\_ptr.search\_for\_suggestions(curr, que);

for (int i = 0; i < 10; i++)

{

myList[i] = "";

}

int que\_inc = 0;

while (que.size())

{

myList[que\_inc] = que.front();

que.pop();

que\_inc++;

}

goto get;

}

}

else if (ch == '=')

{

if (str == "")

{

continue;

}

alphabet\_counter(str, arr);

make\_LL(huff, arr);

store\_LL(huff.top());

huff.make\_huff();

cout << endl;

store\_encoded(get\_code(huff, str));

cout << "\nData compressed and stored successfully in compressed.txt." << endl << endl;

break;

}

else if (ch == '\b')

{

inc = 0;

if (str == "")

{

continue;

}

else

{

if (str.length() == 1)

{

str = "";

system("cls");

myfile.open("open.txt", ios::out);

myfile.close();

myfile.open("open.txt");

cout << "Enter your text: ";

for (int i = 0; i < vec.size(); i++)

{

cout << vec[i] << " ";

myfile << vec[i] << " ";

if (i == vec.size() - 1)

myfile << endl;

}

cout << str;

myfile << str;

continue;

}

else

{

string newst = "";

for (int i = 0; i < str.length(); i++)

{

if (i == str.length() - 1) {}

else

{

newst = newst + str[i];

}

}

str = newst;

system("cls");

myfile.open("open.txt", ios::out);

myfile.close();

myfile.open("open.txt");

cout << "Enter your text: ";

for (int i = 0; i < vec.size(); i++)

{

cout << vec[i] << " ";

myfile << vec[i] << " ";

if (i == vec.size() - 1)

myfile << endl;

}

cout << str;

myfile << str;

if (str != "")

{

trie\_ptr.search\_for\_suggestions(str, que);

for (int i = 0; i < 10; i++)

{

myList[i] = "";

}

int que\_inc = 0;

while (que.size())

{

myList[que\_inc] = que.front();

que.pop();

que\_inc++;

}

goto get;

}

}

}

}

else if (ch == '/')

{

system("cls");

if (str == "")

{

goto get;

}

if (inc >= 10)

{

myfile.open("open.txt", ios::out);

myfile.close();

myfile.open("open.txt");

cout << "Enter your text: ";

for (int i = 0; i < vec.size(); i++)

{

cout << vec[i] << " ";

myfile << vec[i] << " ";

if (i == vec.size() - 1)

myfile << endl;

}

cout << str;

myfile << str;

cout << "\t\t\tNo more suggestions";

continue;

}

cout << "Enter your text: ";

for (int i = 0; i < vec.size(); i++)

{

cout << vec[i] << " ";

}

if (myList[inc] != "")

{

cout << str << endl;

int count = str.length() \* 4 + vec.size();

cout << setw(count) << "\t\t" << myList[inc];

}

else

{

cout << str;

}

curr = myList[inc];

inc++;

}

else

{

inc = 0;

str += ch;

system("cls");

myfile.open("open.txt", ios::out);

myfile.close();

myfile.open("open.txt");

system("cls");

cout << "Enter your text: ";

for (int i = 0; i < vec.size(); i++)

{

cout << vec[i] << " ";

myfile << vec[i] << endl;

}cout << str;

myfile << str;

trie\_ptr.search\_for\_suggestions(str, que);

for (int i = 0; i < 10; i++)

{

myList[i] = "";

}

int que\_inc = 0;

while (que.size())

{

myList[que\_inc] = que.front();

que.pop();

que\_inc++;

}

}

}

}

void Typing\_Text()

{

system("Color B5");

cout << "Enter your text: ";

string\* myList = new string[10];

TrieTree trie\_ptr;

Huffman huff;

int size = 27;

int\* arr;

arr = new int[size];

queue<string> que;

trie\_ptr.readfile();

string str, curr = "";

vector<string> vec;

int count = 0, inc = 0;

while (true)

{

if (count == 10000)

break;

get:

count++;

char ch = \_getch();

Convert(ch);

if (ch == ' ')

{

vec.push\_back(str);

str = "";

}

else if (ch == ';')

{

if (str == "") {

continue;

}

else if (curr != "" && inc < 10)

{

inc = 0;

str = curr;

system("cls");

myfile.open("open.txt", ios::out);

myfile.close();

myfile.open("open.txt");

cout << "Enter your text: ";

for (int i = 0; i < vec.size(); i++)

{

cout << vec[i] << " ";

myfile << vec[i] << " ";

if (i == vec.size() - 1)

myfile << endl;

}

cout << str;

myfile << str;

trie\_ptr.search\_for\_suggestions(str, que);

for (int i = 0; i < 10; i++)

{

myList[i] = "";

}

int que\_inc = 0;

while (que.size())

{

myList[que\_inc] = que.front();

que.pop();

que\_inc++;

}

goto get;

}

}

else if (ch == '=')

{

if (str == "")

{

continue;

}

alphabet\_counter(str, arr);

make\_LL(huff, arr);

store\_LL(huff.top());

huff.make\_huff();

cout << endl;

store\_encoded(get\_code(huff, str));

cout << "\nData compressed and stored successfully in compressed.txt." << endl << endl;

break;

}

else if (ch == '\b')

{

inc = 0;

if (str == "")

{

continue;

}

else

{

if (str.length() == 1)

{

str = "";

system("cls");

myfile.open("open.txt", ios::out);

myfile.close();

myfile.open("open.txt");

cout << "Enter your text: ";

for (int i = 0; i < vec.size(); i++)

{

cout << vec[i] << " ";

myfile << vec[i] << " ";

if (i == vec.size() - 1)

myfile << endl;

}

cout << str;

myfile << str;

continue;

}

else

{

string newst = "";

for (int i = 0; i < str.length(); i++)

{

if (i == str.length() - 1) {}

else

{

newst = newst + str[i];

}

}

str = newst;

system("cls");

myfile.open("open.txt", ios::out);

myfile.close();

myfile.open("open.txt");

cout << "Enter your text: ";

for (int i = 0; i < vec.size(); i++)

{

cout << vec[i] << " ";

myfile << vec[i] << " ";

if (i == vec.size() - 1)

myfile << endl;

}

cout << str;

myfile << str;

if (str != "")

{

trie\_ptr.search\_for\_suggestions(str, que);

for (int i = 0; i < 10; i++)

{

myList[i] = "";

}

int que\_inc = 0;

while (que.size())

{

myList[que\_inc] = que.front();

que.pop();

que\_inc++;

}

goto get;

}

}

}

}

else if (ch == '/')

{

system("cls");

if (str == "")

{

goto get;

}

if (inc >= 10)

{

myfile.open("open.txt", ios::out);

myfile.close();

myfile.open("open.txt");

cout << "Enter your text: ";

for (int i = 0; i < vec.size(); i++)

{

cout << vec[i] << " ";

myfile << vec[i] << " ";

if (i == vec.size() - 1)

myfile << endl;

}

cout << str;

myfile << str;

cout << "\t\t\tNo more suggestions";

continue;

}

cout << "Enter your text: ";

for (int i = 0; i < vec.size(); i++)

{

cout << vec[i] << " ";

}

if (myList[inc] != "")

{

cout << str << endl;

int count = str.length() \* 4 + vec.size();

cout << setw(count) << "\t\t" << myList[inc];

}

else

{

cout << str;

}

curr = myList[inc];

inc++;

}

else

{

inc = 0;

str += ch;

system("cls");

myfile.open("open.txt", ios::out);

myfile.close();

myfile.open("open.txt");

system("cls");

cout << "Enter your text: ";

for (int i = 0; i < vec.size(); i++)

{

cout << vec[i] << " ";

myfile << vec[i] << endl;

}cout << str;

myfile << str;

trie\_ptr.search\_for\_suggestions(str, que);

for (int i = 0; i < 10; i++)

{

myList[i] = "";

}

int que\_inc = 0;

while (que.size())

{

myList[que\_inc] = que.front();

que.pop();

que\_inc++;

}

}

}

}

void Info\_Project()

{

cout << "\t\"A Text Editor in C++ in this project we have used Data Structures concept like Trie Tree and Suffix tree.\n";

cout << "\tUsed File Handling to get words from files and store in Trie Tree to get suggestions while entering words.\n";

cout << "\tWithout Pressing Enter Keyword getting suggestions.\n";

cout << "\tWith '/' you can get suggestions and ; you can save it.\n";

cout << "\tBy 'e' data will be compressed and the loop will be breaked\n";

cout << "\tSo the main purpose to implement this is for use of Data structure in in Our Programming.\"\n\n";

}

void credits()

{

cout << "\n\t\t\t\t\tHusnain Mahmood 21F-9624\n\n";

}

void menu\_driven()

{

cout << "\t\t\t\t\t\tTEXT EDITOR IN C++\n\n";

cout << "\t\t\t\t\tEnter 1 for Project Info\n";

cout << "\t\t\t\t\tEnter 2 for Project Credits\n";

cout << "\t\t\t\t\tEnter 3 for Typing Text\n";

cout << "\t\t\t\t\tEnter 4 for Continue Editing\n";

cout << "\t\t\t\t\tEnter 5 for Uncompressing Data and Printing it\n";

cout << "\t\t\t\t\tEnter 0 to Exit(0)\n\n";

}

int main()

{

system("Color B5");

int choice = 0;

Huffman huff;

string codedStr;

while (true)

{

system("cls");

menu\_driven();

cout << "\t\t\tEnter Option: ";

cin >> choice;

if (choice == 0)

break;

else

{

switch (choice)

{

case 1:

Info\_Project();

break;

case 2:

credits();

break;

case 3:

Typing\_Text();

break;

case 4:

Continue\_Editing();

break;

case 5:

read\_LL(huff);

huff.make\_huff();

cout << "Data: \n";

huff.display(huff.top());

cout << endl;

codedStr = decoded\_str(huff, get\_encoded());

cout << "\nUncompressed Data: " << codedStr << endl;

break;

default:

break;

}

}

system("pause");

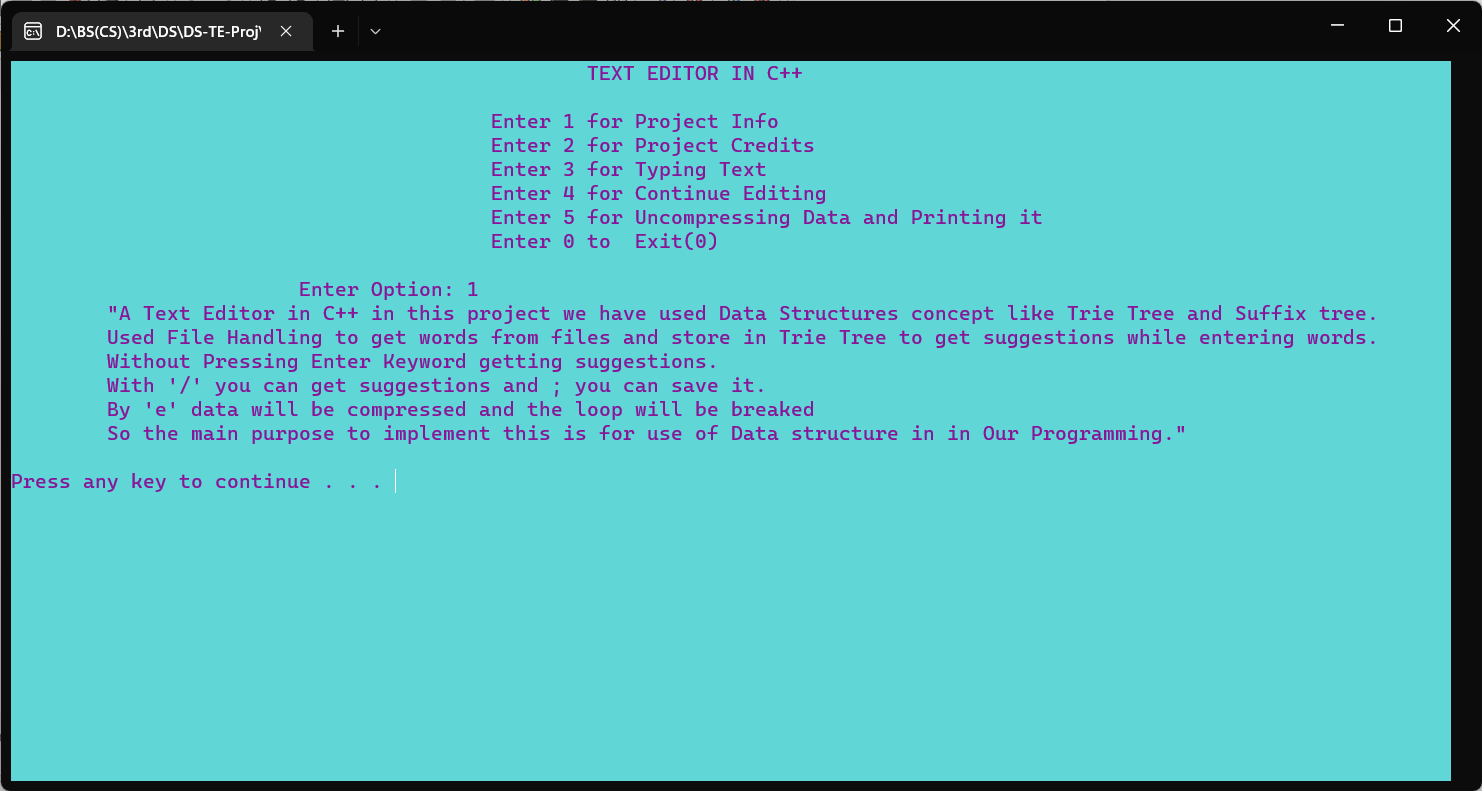
}

system("pause");

return 0;

}

# OUTPUTS:



Text

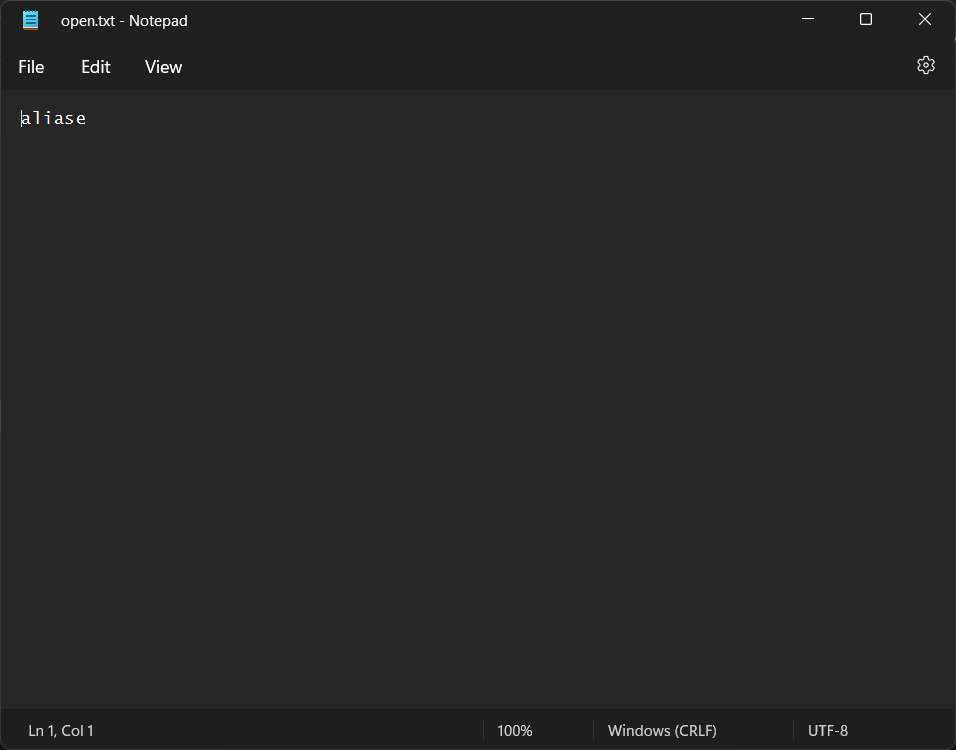
Description automatically generated with medium confidence

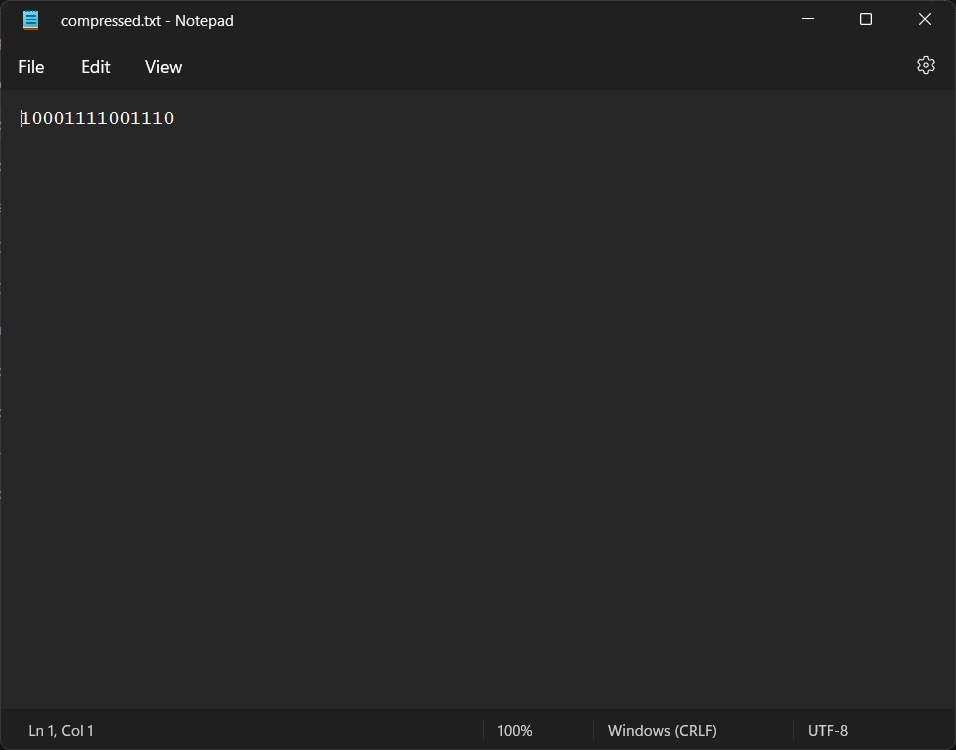
Shape, rectangle

Description automatically generated

Text

Description automatically generated





Graphical user interface, text, application

Description automatically generated