Advance Notepad

Project

21F-9197

21F-9277

**Notepad Project**

**21F-9197**

**21F-9277**

**Code:**

#include <iostream>

#include <iomanip> // setfill or width k lie

#include <string>

#include <sstream>

#include <cstring>

#include<stack>

#include <conio.h> // For \_getch() function

#include<fstream>

#include<windows.h>

#include <vector>

using namespace std;

class CharNode

{

public:

char data;

CharNode\* next;

CharNode\* left;

CharNode\* right;

CharNode\* top;

CharNode\* bottom;

CharNode(char data)

{

this->data = data;

next = NULL;

}

CharNode()

{

}

};

class TextEditor

{

public:

CharNode\* head;

CharNode\* current;

TextEditor() // constructor

{

head = NULL;

current = NULL;

}

stack<std::string> undoStack; // builtin satck used

void insert(const string& s)

{

if (head == nullptr)

{

// If the list is empty, create a new node for the head

//head = new CharNode();

CharNode\* col = new CharNode;

head = col;

// Insert characters of the string in a new column

for (char c : s)

{

CharNode\* newNode = new CharNode(c);

col->next = newNode;

col = newNode;

}

}

else

{

// Traverse to the last row in the linked list

CharNode\* row = head;

while (row->next != nullptr)

{

row = row->next;

}

// Insert a newline character to create a new row

//row->next = new CharNode;

//row = row->next;

CharNode\* col = row;

// Insert characters of the string in a new column

for (char c : s)

{

CharNode\* newNode = new CharNode(c);

col->next = newNode;

col = newNode;

}

}

undoStack.push(s);

}

void changeCursor(int columnPos, int rowPos) //changing position of cursor

{

HANDLE handle;

COORD coord;

coord.X = columnPos;

coord.Y = rowPos;

handle = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleCursorPosition(handle, coord);

}

void deleteText(string s)

{

CharNode\* prev = head;

CharNode\* curr = head->next;

while (curr != NULL)

{

if (curr->data == s[0])

{

bool match = true;

CharNode\* tempCurr = curr;

// Check if the current position matches the string

for (char c : s)

{

// If the current position is null or the data doesn't match, break the loop

if (tempCurr == NULL || tempCurr->data != c)

{

match = false;

break;

}

tempCurr = tempCurr->next;

}

// If there's a match, remove the matched portion from the linked list

if (match)

{

prev->next = tempCurr;

curr = tempCurr;

continue;

}

}

// Move to the next nodes

prev = curr;

curr = curr->next;

}

}

void search(string s)

{

CharNode\* node = head->next;

int count = 0;

while (node != NULL)

{

if (node->data == s[0])

{

bool match = true;

CharNode\* tempNode = node;

// Check if the current position matches the string s

for (char c : s)

{

if (tempNode == NULL || tempNode->data != c)

{

match = false;

break;

}

tempNode = tempNode->next;

}

// If there's a match, increment the count and print a message

if (match)

{

count++;

cout << "\* Found \*" << endl;

}

}

node = node->next;

}

if (count == 0)

{

cout << "String not found." << endl;

}

}

void copy(string& str)

{

string Copy = "";

current = head;

int sx, sy, ex, ey;

cout << "\nEnter the starting position of the string you want to copy: ";

cin >> sx >> sy;

cout << "Enter the ending position of the string you want to copy: ";

cin >> ex >> ey;

// Traverse to the starting position

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

{

current = current->next;

}

// Traverse and copy the characters from starting to ending position

for (int j = sx; j <= ex; j++)

{

Copy += current->data;

current = current->next;

}

str = Copy;

cout << "Copied string: " << Copy << endl;

}

void pasteText(int x, int y, string pastedText)

{

CharNode\* row = head;

int rowCount = 0;

while (row != NULL && rowCount != y)

{

row = row->next;

rowCount++;

}

if (row == NULL) {

// y is out of range

return;

}

CharNode\* col = row;

int colCount = 0;

while (col != NULL && colCount != x) {

col = col->next;

colCount++;

}

if (col == NULL) {

// x is out of range

return;

}

CharNode\* pasteNode = NULL;

for (char c : pastedText)

{

pasteNode = new CharNode(c);

pasteNode->next = col->next;

col->next = pasteNode;

col = pasteNode;

}

if (pasteNode != NULL && pasteNode->data != '\n')

{

// insert a newline character at the end of the pasted text

pasteNode->next = new CharNode('\n');

}

}

void findAndReplace(string& str, const std::string& wordToFind, const string& wordToReplace)

{

size\_t pos = 0; // indicating the beginning of the string.

while ((pos = str.find(wordToFind, pos)) != std::string::npos) //loop occuranne check krne k lie use ki he

{ //npos indicates end of the string or invalid pos

str.replace(pos, wordToFind.length(), wordToReplace);

pos += wordToReplace.length();

}

}

int countWords(TextEditor editor)

{

CharNode\* curr = editor.head->next;

int count = 0;

bool inWord = false; // indicate that the current position is not inside a word.

while (curr != NULL) {

if (curr->data == ' ' || curr->data == '\n' || curr->data == '\t') {

inWord = false;

}

else {

if (!inWord) {

count++;

inWord = true;

}

}

curr = curr->next;

}

return count;

}

void saveToFile(string filename, string key) {

if (key != "@") {

cout << "Invalid key!" << endl;

return;

}

ofstream outfile(filename);

if (!outfile) {

cout << "Error opening file " << filename << endl;

return;

}

CharNode\* curr = head->next;

while (curr != NULL) {

outfile << curr->data;

curr = curr->next;

}

outfile.close();

cout << "Text saved to file " << filename << endl;

}

void paste(const string& copiedData, int x, int y)

{

CharNode\* current = head; // Assuming head is the first node of the linked list

// Traverse to the starting position (x)

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

{

if (current == nullptr)

{

// Handle the case where x is out of bounds

// You can choose to throw an exception or return from the function

return;

}

current = current->next;

}

// Traverse and paste the copied data

for (int i = x; i <= y; i++)

{

if (current == nullptr)

{

// Handle the case where y is out of bounds

// You can choose to throw an exception or return from the function

return;

}

current->data = copiedData[i - x];

current = current->next;

}

}

bool compare(std::string file1, std::string file2)

{

std::ifstream f1(file1);

std::ifstream f2(file2);

std::string line1, line2;

while (std::getline(f1, line1) && std::getline(f2, line2))

{

if (line1 != line2)

{

return false;

}

}

return true;

}

void undo()

{

if (!undoStack.empty())

{

std::string data = undoStack.top();

undoStack.pop();

// Your undo code here using the popped data

std::cout << "Undoing: " << data << std::endl;

}

else

{

std::cout << "Undo stack is empty" << std::endl;

}

}

void displayMessage(string message)

{

// Get the size of the console window

CONSOLE\_SCREEN\_BUFFER\_INFO csbi;

GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &csbi);

int consoleWidth = csbi.srWindow.Right - csbi.srWindow.Left + 1;

int consoleHeight = csbi.srWindow.Bottom - csbi.srWindow.Top + 1;

// Calculate the center coordinates for the message

int messageWidth = message.size();

int messageX = (consoleWidth - messageWidth) / 2;

int messageY = consoleHeight / 2;

// Set the cursor position to the center of the screen

HANDLE handle = GetStdHandle(STD\_OUTPUT\_HANDLE);

COORD coord;

coord.X = messageX;

coord.Y = messageY;

SetConsoleCursorPosition(handle, coord);

// Print the message

cout << message;

// Wait for 2 seconds before clearing the message

Sleep(2000);

// Clear the message by printing spaces

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

{

cout << " ";

}

SetConsoleCursorPosition(handle, coord);

}

void showInstructionsAndContinue()

{

cout << setw(30) << std::setfill('=') << "" << "\n";

cout << setw(30) << std::setfill(' ') << std::left << "Instructions" << "\n";

cout << setw(30) << std::setfill('=') << "" << "\n\n";

cout << setw(30) << std::left << "Press W" << " - Count Words\n";

cout << setw(30) << std::left << "Press X" << " - Delete Words\n";

cout << setw(30) << std::left << "Press U" << " - Undo Text\n";

cout << setw(30) << std::left << "Press S" << " - Search Text\n";

cout << setw(30) << std::left << "Press C" << " - Copy Text\n";

cout << setw(30) << std::left << "Press P" << " - Paste Text\n";

cout << setw(30) << std::left << "Press F" << " - Find Word\n";

cout << setw(30) << std::left << "Press s" << " - Check Similarity\n";

cout << setw(30) << std::left << "Press c" << " - Compare Two Files\n";

cout << setw(30) << std::left << "Press E" << " - Enter Text Again\n";

cout << setw(30) << std::left << "Press \*" << " - To end program\n";

cout << "Press 'X' to continue...\n";

char input;

while (std::cin >> input) {

if (input == 'X' || input == 'x') {

break;

}

std::cout << "Invalid input. Press 'X' to continue...\n";

}

// Rest of your code goes here

std::cout << "Continuing with the rest of the code...\n";

}

void display()

{

CharNode\* node = head->next;

while (node != NULL)

{

if (node->data == '\n')

{

cout << endl;

}

else

{

cout << node->data;

}

node = node->next;

}

cout << endl;

}

};

int j = 0;

//////////////////////////////////////////////////////////

class n1

{

//creating an node sutructure here we will have simple ll nodes which we are using to store our 10auggestion

public:

n1\* n;

string data;

};

void insert1(string w, n1\*& head)//simple ll insertion a we decided to store 10 suggestion in the ll to get them in the main

{

if (head == nullptr)

{

head = new n1;

head->n = head;

head->data = w;

}

else

{

n1\* curr = head;

while (curr->n != head)

{

curr = curr->n;

}

n1\* temp = new n1;

curr->n = temp;

temp->n = head;

temp->data = w;

}

}

///////////////////////////////////////////////////////////////////

class n2

{

//creating a node sutructure so we

public:

n2\* next[26];

bool WE;

};

n2\* getnode()

{

n2\* temp = new n2;

temp->WE = false;

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

{

temp->next[i] = nullptr;

}

return temp;

}

void insert2(n2\* root, string word)//insertion in the tri treee

{

n2\* curr = root;

for (int i = 0; i < word.length(); i++)//here we are inserting the word in the tri tree

{

int idx = word[i] - 'a';//getting the index to create a node on that index

if (!curr->next[idx])

{

curr->next[idx] = getnode();

}

curr = curr->next[idx];

}

curr->WE = true;//after storing a whole word the word end var will be set true

//which will indicate us that the word is completed here

}

////////////////////////////////////////////////

///////////////////////////////////////////////

bool Mylastnode(n2\* root)

{

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

{

if (root->next[i])

{

return 0;

}

}

return 1;

}

void CreatingSuggestion(n2\* root, string word, n1\*& head)

{

if (j == 10)//case to stop our recur..

{

return;

}

if (root->WE)//if the word end var equal to true putt that word into the linklist which we are suing in the main

{

cout << j << ")" << word << endl;

insert1(word, head);

j++;

}

for (int i = 0; i < 26; i++)//here we will create a suggestion

{

if (root->next[i])

{

char c = 'a' + i;//going to that size where the node is present and converting the index to char

// by adding the 'a' in the index and add to out half written word ie concatination them

CreatingSuggestion(root->next[i], word + c, head);

}

}

}

int suggestTheword(n2\* root, string word, n1\*& head)

{

n2\* curr = root;

for (int i = 0; i < word.length(); i++)//as we have the half word so first we will check weather

{ //the related word is present in our tri tree if yes then

int idx = word[i] - 'a';

if (!curr->next[idx])

{

return 0;

}

curr = curr->next[idx];

}

//then we will call the creating suggestion

if (Mylastnode(curr))

{

cout << word;

return -1;

}

CreatingSuggestion(curr, word, head);

}

int main()

{

////////////////////////////////

ifstream in;

in.open("outfile.txt");

string a;

n2\* root = getnode();

while (!(in.eof()))

{

getline(in, a);

insert2(root, a);

}

n1\* head = nullptr;

string msg;

//////////////////////////////////

string copi;

TextEditor editor;

string text;

while (1)

{

editor.displayMessage("Welcome to Advane NotePad ");

cout << endl;cout << endl;

editor.showInstructionsAndContinue();

ofstream f("off.txt", ofstream::trunc);

f.close();

int x = 0, y = 0;

string t2;

cout << "Enter Y and X cordinates :";

cin >> x >> y;

cin.ignore();

cout << "Enter text to insert :";

//////////////////////////////////////////////

/\* try tree main\*/

cout << endl;

cout << endl;

ofstream of;

ofstream oo;

string copy = " ";

int countspace = 0;

int sjcount = 0;

while (copy != "-")

{

cout << "\npress - when you are done with the insert string:";

t2 = \_getch();

if (t2 == " ")

{

countspace++;

}

system("cls");

ifstream ok;

string gg;

ok.open("myorignaldata.txt");

getline(ok, gg);

cout << gg << endl;

ifstream ok1;

string gg1;

ok1.open("off.txt");

getline(ok1, gg1);

cout << gg1;

cout << t2 << endl;

copy = t2;

if (t2 >= "a" && t2 <= "z" || t2 >= "A" && t2 <= "Z")//add all spaashal ch to avoid any incon...

{

of.open("off.txt", ios::app);

of << t2; //yha pr hm ak file ma ch concat krwaty jyengy or phr usko sary ko aktha utha k suggestion dekhaty jyengy

of.close();

ifstream on;

on.open("off.txt");

on >> t2;

on.close();

cout << "to select suggestion press the number of that suggestion \n";

//yhapr 1 sy 10 tk koi b number daal dyna ha wo khud ba khud select ho k file ma jata jyega my orignal data wali ma

suggestTheword(root, t2, head);

char ii = \_getch();//yhapar wo jonsy number wli select kryga

int i1 = ii - 48; //usko int ma krlia

if (i1 >= 0 && i1 <= 10 && head != nullptr)//ll ma suggestion daal k ly aye thy hm ab isma sy check krna k wo ha b k ni mtlb 10 sy chota ho number or head null b na ho

{

int copyii = 0;

n1\* curr = head;

while (curr->n != curr && copyii != i1)

{

curr = curr->n;

copyii++;

}

if (copyii == i1)//iska mtlb k hmy milgya ha wo number mtlb 10sy necy nechy e khin tha

{

oo.open("myorignaldata.txt", ios::app);

sjcount++;

oo << curr->data;//phr select ki wi suggestion ko hmna chuk k wha daal dia myorignal data wali file ma

oo << " ";

oo.close();

ofstream file("off.txt", ofstream::trunc);

file.close();

/\* string temp;

ifstream uthao;

uthao.open("myorignaldata.txt", ios::app);

getline(uthao, temp);

cout << temp;

uthao.close();\*/

}

}

j = 0;

head = nullptr;

}

else if (t2 != "-") //suppose agr wo space daal dyta to mtlb ak jo ab dalna shoro kryga usk hisab sy suggestion dekhani ha

{

string all;

ifstream on;

on.open("off.txt");

on >> all;

on.close();

oo.open("myorignaldata.txt", ios::app);

oo << all;

oo << " ";

oo.close();

ofstream file("off.txt", ofstream::trunc);

file.close();

/\* string temp;

ifstream uthao;

uthao.open("myorignaldata.txt", ios::app);

getline(uthao, temp);

cout << temp;

uthao.close();\*/

}

}

system("cls");

if (countspace != 0)

{

ifstream my;

my.open("myorignaldata.txt", ios::app);

getline(my, text);

ofstream g("myorignaldata.txt", ofstream::trunc);

g.close();

}

else

{

if (countspace == 0 && sjcount != 0)

{

ifstream my;

my.open("myorignaldata.txt", ios::app);

getline(my, text);

ofstream g("myorignaldata.txt", ofstream::trunc);

g.close();

}

else

{

ifstream my;

my.open("off.txt", ios::app);

getline(my, text);

ofstream g("off.txt", ofstream::trunc);

g.close();

}

}

editor.insert(text);

editor.changeCursor(x, y);

editor.display();

while (true)

{

if (\_kbhit()) // Check if a key is pressed

{

char key = \_getch(); // Get the pressed key

if (key == 'E') // ASCII value for Ctrl+C

break;

else if (key == 'W')

{

int wordCount = editor.countWords(editor);

cout << "The editor contains " << wordCount << " words." << endl;

}

else if (key == 'X') // Compare ASCII value for T

{

string sample;

cout << "Enter text you want to delete :";

cin >> sample;

editor.deleteText(sample);

editor.display();

cout << endl;

}

else if (key == 'u')

{

editor.undo();

}

else if (key == 'S')

{

string searchStr;

cout << "Enter text to search :";

cin >> searchStr;

editor.search(searchStr);

}

else if (key == 'C')

{

editor.copy((copi));

}

else if (key == 'P')

{

int pasteX, pasteY;

cout << "Enter the x and y position to paste the copied text: ";

cin >> pasteX >> pasteY;

if (pasteX < 0 || pasteY < 0)

{

cout << "Invalid position." << endl;

}

else

{

/\* system("cls");\*/

cin.ignore();

editor.changeCursor(pasteX, pasteY);

cout << copi;

/\*editor.paste(copi, pasteX, pasteY);

editor.display();\*/

}

cout << endl;

}

else if (key == 'F')

{

string wordToFind;

cout << "Enter word to find:";

cin >> wordToFind;

string wordToReplace;

cout << "Enter word to Replace:";

cin >> wordToReplace;

cout << "Before findAndReplace (orignal string) : " << text << std::endl;

editor.findAndReplace(text, wordToFind, wordToReplace);

cout << "After findAndReplace (Updated string) : " << text << std::endl;

}

else if (key == 's')

{

string str1;

string str2;

string alpha;

label:

cin.ignore();

cout << "Enter text to insert to check 1st file saved simalirity:";

getline(cin, str1);

editor.insert(str1);

cout << "Enter @ to save data :";

cin >> alpha;

if (alpha == "@")

{

editor.saveToFile("output.txt", alpha);

editor.saveToFile("output2.txt", alpha);

cout << "Data saved in both files successfuly ";

}

else

{

cout << "try again and press @ to save data in files " << endl;

goto label;

}

}

else if (key == 'c')

{

if (editor.compare("output.txt", "output2.txt"))

{

cout << "Files are equal" << endl;

}

else

{

cout << "Files are different" << endl;

}

}

else if (key == '\*')

{

exit(0);

}

}

}

}

}