**DATA STRUCTURES**

**LAB FINAL PROJECT**

**Name: Abdul Raheem**

**Roll no: 21F-9051**

**Sec: 4D**

#include<iostream>

#include<string>

#include<fstream>

#include<iomanip>

const int MAX\_SUGGESTIONS = 10;

using namespace std;

struct Node {

char data = ' ';

Node\* next = nullptr;

};

class node {

public:

int position;

char data;

node\* prev;

node\* next;

node\* up;

node\* down;

node() {

position = 0;

data = '\0';

prev = next = up = down = nullptr;

}

node(char data, int position) {

this->position = position;

this->data = data;

prev = next = up = down = nullptr;

}

};

class Stack {

Node\* top = nullptr;

int size = 0;

public:

Stack() {}

bool isEmpty();

void push(char data);

char top\_data();

};

bool Stack::isEmpty() {

if (top == nullptr) {

return true;

}

else return false;

}

void Stack::push(char data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = NULL;

newNode->next = top;

top = newNode;

++size;

}

char Stack::top\_data() {

if (isEmpty()) {

cout << "stack top is empty: " << endl;

return ' ';

}

else {

return top->data;

}

}

class Advanced\_Notepad {

public:

Advanced\_Notepad() {

position = 0;

temp1 = tail = head = new node('\0', position);

}

void index\_insertion(char data);

void insert(char data);

void down\_link();

void del();

void print();

void move\_up();

void move\_down();

void move\_left();

void move\_right();

void copy();

void paste();

void undo();

void pos\_check();

void stack\_top();

void find\_and\_replace();

private:

node\* head;

node\* tail;

node\* temp1;

Stack obj;

int position;

//pointer helper functions

void set\_tail\_at\_temp1() {

tail = temp1;

}

void set\_tail\_at\_head() {

tail = head;

}

void temp1\_at\_head() {

temp1 = head;

}

//basic 4 D link list helper functions

void insertion\_at\_index(char choice, char data) {

node\* newNode = new node(data, ++position);

if (tail->next == nullptr)

insert(data);

if (head->data == data) {

newNode->next = head;

newNode->next->prev = newNode;

head = newNode;

}

else {

node\* temp = head;

while (temp) {

if (temp->data == choice) {

newNode->next = temp->next;

newNode->prev = temp->next->prev;

temp->next = newNode;

break;

}

temp = temp->next;

}

}

}

void deletion(node\* temp3) {

if (head->next == nullptr) {

cout << "list is empty: " << endl;

return;

}

node\* temp = temp3;

if (temp == head) {

head->down = head->next->down;

head->up = head->next->down;

head = temp->next;

head->prev = nullptr;

temp1 = head;

}

else {

if (temp3->up != nullptr) {

temp3->up->down = temp3->next;

temp3->next->up = temp3->up;

temp3->down = temp3->next->down;

temp3 = temp->next;

temp3->prev = nullptr;

temp1 = temp3;

}

else {

temp1 = temp->prev;

temp1->next = temp->next;

if (temp->next != nullptr) {

node\* temp2 = temp->next;

temp2->prev = temp->prev;

}

}

}

obj.push(temp->data);

free(temp);

}

void display(node\* n) {

cout << "data->" << n->data << "at position " << n->position << endl;

if (n->next != nullptr) {

display(n->next);

}

if (n->down != nullptr) {

cout << "\nDown nodes: " << endl;

display(n->down);

}

}

void check() {

if (head == nullptr) {

cout << "head is null: " << endl;

return;

}

if (head->prev == nullptr) {

cout << "previous is null: " << endl;

}

if (head->next == nullptr) {

cout << "next is null: " << endl;

}

if (head->prev != nullptr && head->next == nullptr) {

{

cout << "Node is not NULL: " << endl;

}

return;

}

return;

}

void recalulate\_positions(node\* n) {

n->position = ++position;

if (n->next != nullptr)

recalulate\_positions(n->next);

if (n->down != nullptr)

recalulate\_positions(n->down);

}

void Undo(int pos, node\* n) {

if (n->next != nullptr) {

if (pos == n->position) {

int cordinate = n->position;

node\* newNode = new node(obj.top\_data(), ++cordinate);

newNode->next = n->next;

newNode->next->prev = newNode;

n->next = newNode;

newNode->prev = n;

}

Undo(pos, n->next);

}

if (n->down != nullptr) {

cout << "\nDown nodes: " << endl;

Undo(pos, n->down);

}

pos\_check();

}

void find\_and\_replace\_traversal(int pos, node\* n) {

if (n->next != nullptr) {

if (pos == n->position) {

char data;

cout << "Enter data to replace with: ";

cin >> data;

n->data = data;

}

find\_and\_replace\_traversal(pos, n->next);

}

if (n->down != nullptr) {

cout << "\nDown nodes: " << endl;

find\_and\_replace\_traversal(pos, n->down);

}

}

};

//insertions all cases handeled

void Advanced\_Notepad::index\_insertion(char data) {

insertion\_at\_index(temp1->data, data);

}

void Advanced\_Notepad::insert(char data) {

if (tail == nullptr)

tail = head;

check();

if (tail == nullptr) {

cout << "tail is null: " << endl;

return;

}

++position;

node\* newNode = new node(data, position);

tail->next = newNode;

newNode->prev = tail;

tail = tail->next;

}

//creation at a point in the down link

void Advanced\_Notepad::down\_link() {

node\* newNode = new node;

temp1->down = newNode;

newNode->up = temp1;

temp1 = newNode;

cout << "setting tail equal to temp 1 so insertion can be done: " << endl;

set\_tail\_at\_temp1();

}

//deletion at any point

void Advanced\_Notepad::del() {

deletion(temp1);

}

//display

void Advanced\_Notepad::print() {

display(head);

}

//cursor control keys and check data

void Advanced\_Notepad::move\_up() {

if (temp1->up != nullptr) {

temp1 = temp1->up;

cout << "\npointer is moved to up pointer: " << endl;

cout << "\n data-> " << temp1->data << endl;

}

if (temp1->up == nullptr) {

cout << "up is null: " << endl;

return;

}

}

void Advanced\_Notepad::move\_down() {

if (temp1->down != nullptr) {

temp1 = temp1->down;

cout << "\npointer is moved to down pointer: " << endl;

cout << "\n data-> " << temp1->data << endl;

}

if (temp1->down == nullptr) {

cout << "down is null: " << endl;

return;

}

}

void Advanced\_Notepad::move\_left() {

if (temp1->prev != nullptr) {

temp1 = temp1->prev;

cout << "\npointer is moved to previous pointer: " << endl;

cout << "\n data-> " << temp1->data << endl;

}

if (temp1->prev == nullptr) {

cout << "previous is null: " << endl;

return;

}

}

void Advanced\_Notepad::move\_right() {

if (temp1->next != nullptr) {

temp1 = temp1->next;

cout << "\npointer is moved to next pointer: " << endl;

cout << "\n data-> " << temp1->data << endl;

}

if (temp1->next == nullptr) {

cout << "next is null: " << endl;

return;

}

}

void Advanced\_Notepad::stack\_top() {

cout << "top data of stack is: " << obj.top\_data() << endl;

}

//tools of notepad

void Advanced\_Notepad::copy() {

obj.push(temp1->data);

}

void Advanced\_Notepad::paste() {

int pos = 0;

print();

cout << "Enter position at which you want to paste your word: ";

cin >> pos;

node\* tempr = head;

Undo(pos, tempr);

pos\_check();

}

void Advanced\_Notepad::undo() {

int pos = 0;

print();

cout << "Enter position at which you want to enter your word: ";

cin >> pos;

node\* tempr = head;

Undo(pos, tempr);

}

void Advanced\_Notepad::pos\_check() {

position = -1;

recalulate\_positions(head);

}

void Advanced\_Notepad::find\_and\_replace() {

int pos = -1;

print();

cout << "Enter position yo want to find and change data of: ";

cin >> pos;

find\_and\_replace\_traversal(pos, head);

}

// Function to read similar words from a file

void readSimilarWordsFromFile(const std::string& filename, const std::string& inputString, std::string suggestions[]) {

std::ifstream file(filename);

if (file.is\_open()) {

std::string word;

int index = 0;

while (file >> word && index < MAX\_SUGGESTIONS) {

if (word.substr(0, inputString.length()) == inputString) {

suggestions[index] = word;

index++;

}

}

file.close();

}

}

// Function to display suggestions based on user input

void displaySuggestions(const std::string& inputString, const std::string suggestions[]) {

std::cout << "Suggestions for '" << inputString << "':" << std::endl;

for (int i = 0; i < MAX\_SUGGESTIONS; i++) {

std::cout << "- " << suggestions[i] << std::endl;

}

}

void instructions() {

cout << endl << endl << endl;

cout << "<------------------------------------------------ADVANCED NOTEPAD------------------------------------------------>";

cout << endl << endl << endl;

cout << setw(4) << "\n0 (----------->To exit loop<----------)";

cout << "\n1 (----------> To insert <----------)";

cout << "\n2 (-----------> To display <----------)";

cout << "\n3 (-----------> To deletion <----------)";

cout << "\n4 (-----------> move left node <----------)";

cout << "\n5 (-----------> move node <----------)";

cout << "\n6 (-----------> move up node <----------)";

cout << "\n7 (-----------> move down node <----------)";

cout << "\n8 (-----------> get down link <----------)";

cout << "\n9 (-----------> copy <----------)";

cout << "\n10 (-----------> paste <----------)";

cout << "\n11 (-----------> undo <----------)";

cout << "\n12 (-----------> stack top <----------)";

cout << "\n13 (-----------> find and replace <----------)";

cout << "\n14 (-----------> to load suggestions from the file and display them <----------)";

cout << endl << endl;

}

void operate() {

Advanced\_Notepad obj;

int choice;

char data = ' ';

int size = 100;

string arr = " ";

std::string inputString;

std::string suggestions[MAX\_SUGGESTIONS];

do {

instructions();

cout << "Enter choice: ";

cin >> choice;

cin.ignore(); // ignore the newline character left by cin

if (choice == 1) {

system("cls");

cout << "Enter data: ";

getline(cin, arr);

for (char c : arr) {

obj.insert(c);

}

}

if (choice == 2) {

system("cls");

cout << " data " << endl;

obj.print();

}

if (choice == 3) {

system("cls");

obj.del();

}

if (choice == 4) {

system("cls");

obj.move\_left();

}

if (choice == 5) {

system("cls");

obj.move\_right();

}

if (choice == 6) {

system("cls");

obj.move\_up();

}

if (choice == 7) {

system("cls");

obj.move\_down();

}

if (choice == 8) {

system("cls");

obj.down\_link();

}

if (choice == 9) {

system("cls");

obj.copy();

}

if (choice == 10) {

system("cls");

obj.paste();

}

if (choice == 11) {

system("cls");

obj.undo();

}

if (choice == 12) {

system("cls");

obj.stack\_top();

}

if (choice == 13) {

system("cls");

obj.find\_and\_replace();

}

if (choice == 14) {

system("cls");

cout << "Enter a string: ";

cin >> inputString;

readSimilarWordsFromFile("outfile.txt", inputString, suggestions);

displaySuggestions(inputString, suggestions);

}

} while (choice != 0);

}

int main() {

operate();

}