

|  |  |
| --- | --- |
| **DSA PROJECT** | |
| **Course Title:** | Data Structures and Algorithms |
| **Submitted to:** | Mr. Hilmand Khan |
| **Submitted By:** | Muhammad Huzaifa -221564  Hassan bin Aman-221490  Sharjeel Iftikhar-221534 |
| **Class:** | BSCYS III-A |

**Faculty of Computing and Artificial Intelligence**

**Air University, Islamabad**

**Text Editor with Spell Checker in C++**

1. **Introduction:**

**Purpose of the Program:**

The primary purpose of this program is to implement a text editor with a spell checker in C++. The text editor uses a *2D doubly linked list* to store the text, where each node in the list represents a character. The list is “*2D*” because it allows for multiple lines of text, each represented as a separate doubly linked list. The text editor supports various operations such as inserting a character, deleting a character, and moving the cursor up, down, left, or right. The cursor is represented as a special node in the list. When a new character is inserted, it is added at the cursor’s position and the cursor is moved to the right.

The spell checker is a separate function that takes a dictionary of words and a word as input. It generates alternative words by substituting, omitting, inserting, and reversing letters in the word. If any of the alternative words are found in the dictionary, they are printed out.

The program also includes functions to save the text to a file and load the text from a file. The text is saved without the cursor, and when it is loaded, the cursor is placed at the end of the text.

**Overview of the code:**

The code is divided into several parts. The Node class represents a node in the doubly linked list.0

Each *node* has five properties:

*Data*,*before*,*after*,*below*, *and above.*

The *Doublylinkedlist* class represents the doubly linked list itself. It has five private properties:

*head*,*tail*,*line*,and*cursor*, and several public methods to manipulate the list.

The *SpellChecker* function checks if a given word is spelled correctly. It takes a dictionary of words and a word as input. It then generates alternative words by *substituting*, *omitting*, *inserting*, and *reversing* letters in the word. If any of the alternative words are found in the dictionary, they are printed out.

The main function is where the program starts. It creates a Doublylinkedlist object and a dictionary of words. It then enters a loop where it waits for the user to press a key. Depending on the key pressed, it performs various operations like moving the cursor, inserting a character, deleting a character, saving the text, loading the text, and checking the spelling of a word. The loop continues until the user presses the ESC key, at which point the program exits.

1. **Data Structures Used:**

The main data structures used in this program are the Node

and Doublylinkedlist classes. These classes represent the nodes and the list in the 2D doubly linked list used to store the text.

**Explanation of the Node Class:**

The Node class represents a node in the doubly linked list. Each node has five properties:

* Data: The character data stored in the node.
* before: A pointer to the node before the current node.
* after: A pointer to the node after the current node.
* below: A pointer to the node below the current node.
* above: A pointer to the node above the current node.

The Node class also has a constructor that initializes these properties. The Data property is initialized with the input character, and the before, after, below, and above pointers are all initialized to NULL.

**Explanation of the Doubly Linked List Class**

The Doublylinkedlist class represents the doubly linked list itself. It has five private properties:

* head: A pointer to the first node in the list.
* tail: A pointer to the last node in the list.
* line: A pointer to the line where the cursor is located.
* cursor: A pointer to the node where the cursor is located.

The Doublylinkedlist class also has several public methods to manipulate the list, such as Insert(chardata), Moveright(),

Moveabove(), and others. These methods perform various operations on the list, such as inserting a new character at the cursor’s position, moving the cursor to the right, moving the cursor up, and so on.

In summary, the Node and Doublylinkedlist classes form the backbone of the text editor’s data structure. They provide the structure of the doubly linked list and enable the various operations that can be performed on the list.

1. **Text Editor Operations**

**Inserting a Character**

The Insert(char data) method is used to insert a new character at the cursor’s position. The method starts by creating a new node with the data of the last node in the line where the cursor is located. This new node is then linked to the last node and any nodes above or below it. After that, the method shifts the data of all nodes from the cursor to the last node to the right. Finally, the new data is inserted at the cursor’s position and the cursor is moved to the right.

Following is the code for this operation:

void Insert(char data)

{

Node\* temp = cursor;

while(temp->after)

{

temp = temp->after;

}

Node\* NewNode = new Node(temp->Data);

NewNode->before = temp;

temp->after = NewNode;

if (temp->above && temp->above->after)

{

NewNode->above = temp->above->after;

NewNode->above->below = NewNode;

}

if (temp->below && temp->below->after)

{

NewNode->below = temp->below->after;

NewNode->below->above = NewNode;

}

while (temp != cursor)

{

temp->Data = temp->before->Data;

temp = temp->before;

}

temp->Data = data;

cursor = cursor->after;

}

**Deleting a Character**

The Delete() method is used to delete the character before the cursor and shift all characters after the cursor to the left. The method starts by checking if there is a node before the cursor. If there is, it sets a temporary pointer to the node before the cursor and moves the cursor to that node. Then, it shifts the data of all nodes after the cursor to the left. Finally, it deletes the last node in the line.

Following’s the code for this operation:

void Delete()

{

if (cursor->before==NULL)

{

return;

}

Node\* temp = cursor->before;

cursor = cursor->before;

while (temp->after)

{

temp->Data = temp->after->Data;

temp = temp->after;

}

if (temp->above)

{

temp->above->below = NULL;

}

if (temp->below)

{

temp->below->above = NULL;

}

if (temp->before)

{

temp->before->after = NULL;

}

temp->above = NULL;

temp->below = NULL;

temp->after = NULL;

temp->before = NULL;

delete temp;

}

**Moving the Cursor**

The cursor can be moved up, down, left, or right using the Moveabove(), Movebelow(), Moveleft(), and Moveright() methods, respectively. Each of these methods updates the cursor pointer to point to a different node in the list. The specific node depends on the direction of the movement and the current position of the cursor. If the movement would take the cursor outside the list, the method either does nothing (for left and up movements) or adds a new node or line to the list (for right and down movements).

1. **Spell Checker Function**

The spell checker function in this program is implemented through the SpellChecker(vector<string> Dictionary, string text) method. This function checks if a given word is spelled correctly based on a provided dictionary of words. If the word is not found in the dictionary, the function generates alternative words and checks if any of them are in the dictionary.

**Generating Alternative Words**

The spell checker generates alternative words by performing four operations on the given word:

1. **Letter Substitution**: For each position in the word, it substitutes the letter at that position with every possible letter from ‘a’ to ‘z’.
2. **Letter Omission**: For each position in the word, it omits the letter at that position.
3. **Letter Insertion**: For each position in the word, it inserts every possible letter from ‘a’ to ‘z’ at that position.
4. **Letter Reversal**: For each pair of adjacent letters in the word, it reverses their order.

Each of these operations could potentially generate a correctly spelled word from a misspelled word. For example, if the user types “hte”, reversing the first two letters generates the correctly spelled word “the”.

**Checking the Dictionary**

After generating the alternative words, the spell checker checks if any of them are in the dictionary. The dictionary is represented as a vector of strings. For each alternative word, the spell checker goes through the entire dictionary and checks if the word is in it. If the word is found, it is printed out as a suggested correction for the misspelled word.

If no alternative words are found in the dictionary, the spell checker prints “You are a hopeless speller!” to indicate that it couldn’t find any potential corrections for the misspelled word.

Here’s the code for this operation:

void SpellChecker(vector<string> Dictionary, string text)

{

string temp;

int hl = 0;

cout << "Here are the alternatives for given word:\n";

// Generate and check alternative words

// ...

if (hl == 0)

cout << "\nYou are a hopeless speller!\n";

}

This spell checker function is a simple but effective way to provide spelling corrections. It’s based on the idea that most spelling errors are due to small mistakes such as typing the wrong letter, omitting a letter, or reversing the order of two letters. By generating alternative words that correct these types of mistakes and checking if they are in the dictionary, the spell checker can provide useful corrections for many common spelling errors.

1. **File I/O Functions**

**Saving the Text to a File**

The Saving() method is used to save the text to a file. The method starts by creating an empty string. It then goes through each node in the list (except the cursor) and adds the node’s data to the string. Finally, it returns the string.

Here’s the code for this operation:

string Saving()

{

string Save="";

Node\* temp = head;

while (temp)

{

Node\* temp2 = temp;

while (temp2)

{

if (temp2 != cursor)

{

Save += temp2->Data;

}

temp2 = temp2->after;

}

temp = temp->below;

}

return Save;

}

The actual saving to a file is done in the main function when the user presses Ctrl + S. The Saving() method is called to get the string representation of the text, and this string is then written to the file.

if (ch == 19) { // Ctrl + S (Save)

ofstream file("Save.txt");

if (file) {

file << text.Saving();

file.close();

cout << "Text saved successfully." << endl;

}

else {

cout << "Unable to open or create the file for saving." << endl;

}

}

**Loading the Text from a File**

The loading from a file is done in the main function when the user presses Ctrl + L. The program opens the file, reads each line, and for each character in the line, it calls the Insert(char data) method to add the character to the list.

if (ch == 12) { // Ctrl + L (Load)

ifstream file("Save.txt");

if (file) {

while (getline(file, line)) {

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

{

text.Insert(line[i]);

}

}

file.close();

}

else {

cout << "File not found or unable to open." << endl;

}

}

These file I/O functions allow the text editor to save the text to a file and load the text from a file, providing a way to persist the text across different sessions of using the text editor.

1. **Main Function**

The main function is the entry point of the program. It’s responsible for initializing the text editor and handling user input.

**Initializing the Text Editor**

At the start of the main function, several variables and objects are initialized:

* A Doublylinkedlist object named text is created. This object represents the text in the text editor.
* A vector<string> named Dictionary is created to store the dictionary of words for the spell checker.
* A string named line and a char named ch are created to store a line of text and a character, respectively.
* The dictionary of words is loaded from a file named “dictionary.txt” into the Dictionary vector.

Here’s the code for this part:

int main()

{

char ch='\0';

string line;

string word="";

Doublylinkedlist text;

vector<string> Dictionary;

ifstream file("dictionary.txt");

if (file) {

string line;

while (getline(file, line)) {

Dictionary.push\_back(line);

}

file.close();

}

// ...

}

**User Input Loop**

After initializing the text editor, the main function enters a loop where it waits for the user to press a key. Depending on the key pressed, it performs various operations:

* If the user presses a character key, the character is inserted at the cursor’s position in the text.
* If the user presses the backspace key, the character before the cursor is deleted.
* If the user presses the arrow keys, the cursor is moved in the corresponding direction.
* If the user presses Ctrl + S, the text is saved to a file.
* If the user presses Ctrl + L, the text is loaded from a file.
* If the user presses ESC, the program exits.

Here’s the code for this part:

while (true) {

clearScreen();

if(word!="")

SpellChecker(Dictionary, word);

cout << "Text Notepad" << endl;

cout << "-------------" << endl;

text.Print();

ch = \_getch();

// Handle the key pressed by the user

// ...

return 0;

}

This loop continues until the user exits the program. This allows the user to continuously interact with the text editor, inserting and deleting characters, moving the cursor, saving and loading the text, and checking the spelling of words. The state of the text editor is updated after each operation and is printed to the console at the start of each iteration of the loop. This provides immediate feedback to the user about the result of their actions.

1. **Conclusion**

**Summary of the Program’s Functionality**

This C++ program effectively implements a text editor with a spell checker. The text editor uses a 2D doubly linked list to store the text, allowing for multiple lines of text. Each node in the list represents a character, and the list supports various operations such as inserting a character, deleting a character, and moving the cursor up, down, left, or right.

The spell checker function takes a dictionary of words and a word as input. It generates alternative words by substituting, omitting, inserting, and reversing letters in the word. If any of the alternative words are found in the dictionary, they are printed out.

The program also includes functions to save the text to a file and load the text from a file. The text is saved without the cursor, and when it is loaded, the cursor is placed at the end of the text.

***END***