HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

FACULTY OF ELECTRICAL – ELECTRONICS ENGINEERING

COMPUTER SYSTEM ENGINEERING C++ PROJECT REPORT

Topic: Write a simple text editor using the list ADT

Lecturer: Assoc. Prof. Trương Quang Vinh

Class: TT01 **Group:** 02

Student's Name	Student's ID
Nguyễn Nhật Tuấn Minh	2151229
Trương Minh Khôi	2151216
Lương Hoàng Phúc	2010525
Châu Hồng Phước	2051175

CONTENTS

I. Introduction

As our topic is “write a simple text editor using the list ADT” so at first, we want to define what is the text editor and list ADT.

A text editor is a software tool designed for creating and editing plain text files. It provides a simple and efficient interface for users to write and modify text-based content. Text editors are widely used by programmers, writers, and anyone who needs to work with text-based documents.

Text editors come in various forms, ranging from basic editors that offer minimal functionality to advanced editors with powerful features and customizable options. Some popular text editors include Notepad (Windows), TextEdit (Mac), and Visual Studio Code.

The List Abstract Data Type (ADT) is a fundamental data structure in computer science that represents a collection of elements arranged in a specific order. It provides a way to store and manipulate a sequence of items, where each item can be accessed using its position or index within the list.

The List ADT supports various operations to manage the elements it contains. These operations typically include:

. Insertion: Adding an element to the list at a specified position.
. Deletion: Removing an element from the list at a specified position.
. Access: Retrieving the value of an element at a given position.
. Search: Finding the position of a specific element within the list.
. Update: Modifying the value of an element at a particular position.

The List ADT is fundamental in many programming languages and serves as a building block for more complex data structures like stacks, queues, and trees. It is widely used in various applications, including data processing, database management, and algorithm design.

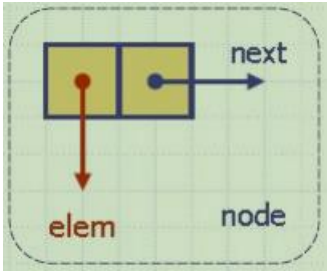
In summary, a simple text editor using the List ADT provides the basic functionality of creating, editing, and manipulating text-based content. By utilizing a linked list as the underlying data structure, the text editor can efficiently handle operations such as insertion, deletion, access, search, and update, enabling users to work with text in a flexible and intuitive manner.

I. Theory

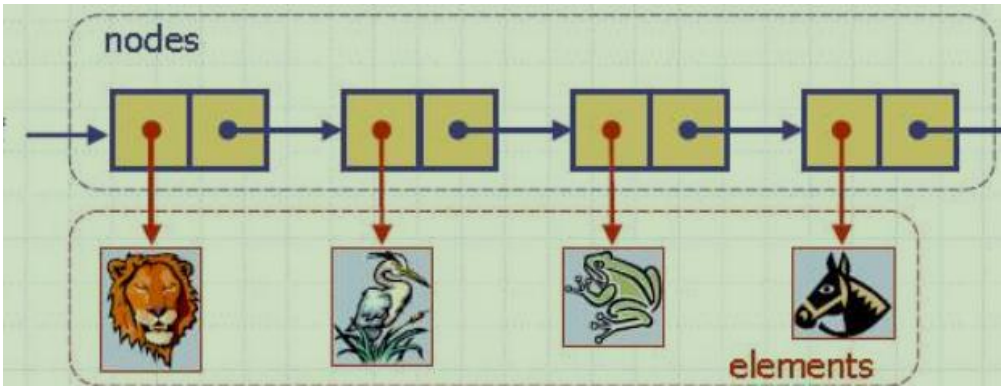
To write a simple text editor using the List Abstract Data Type (ADT), these are some definition to understanding:

1. Define the single Link List structure:

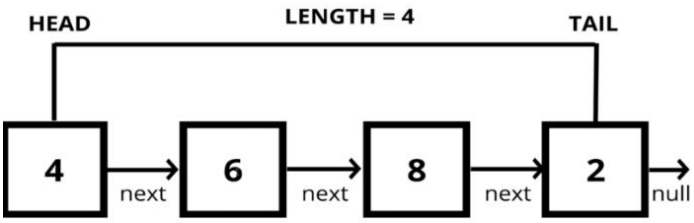
- Create a structure or class called `IntSLLNode` that represents a node in the linked list.



- Each node should contain a character or a string to store the text data, as well as pointers to the next nodes.



- Declare private member variables such as `head`, `tail`, and `size` to keep track of the linked list.



2. Implement the data structure operation:

-To operate the data structure, we construct the function intSLL (mentioned in III part) as ‘insert’, ‘delete’, ‘move’, ‘replace’, …

In my program we have 3 different delete function namely **deleteFromTail , deleteFromHead, deleteNode** which serve different purpose:

insertNode(,): To insert the new node in a specific positon in a linked list.

deleteFromHead(): Removes the first node by updating the head pointer to the second node

deleteFromTail(): Removes the last node by updating the next pointer of the second-to-last node to NULL.

- **deleteNode**(): Removes a specific node by updating the next pointer of the node before the node to be deleted to the after node. Then the delete the node.

- **findNodeElementText**(): To find the node contain in a specific text in a linked list.

- **moveNode**(,): To move a node from a current position to the new position in a linked list.

- **replaceNode**(): To replace the current node for the new data in a linked list

- **addToHead**(): To add the node to the first of a linked list.

- **addToTail**(): To add the node to the last of a linked list.

- **deleteAll**(): To delete all nodes in a linked list.

3. Implement the user's command

To process the command from user, we construct the function 'MyFunctions' (mentioned in part III).

LeftTrim()
RightTrim()
SplitSringBySpaceToArray()
SplitStringByDelimeter()
ToLowerCase()

- For function 'LeftTrim' and 'RightTrim', it find the space in the command of user.

- Function 'SplitSringBySpaceToArray', it splits the command into the parts to implement based on the space.

For example:

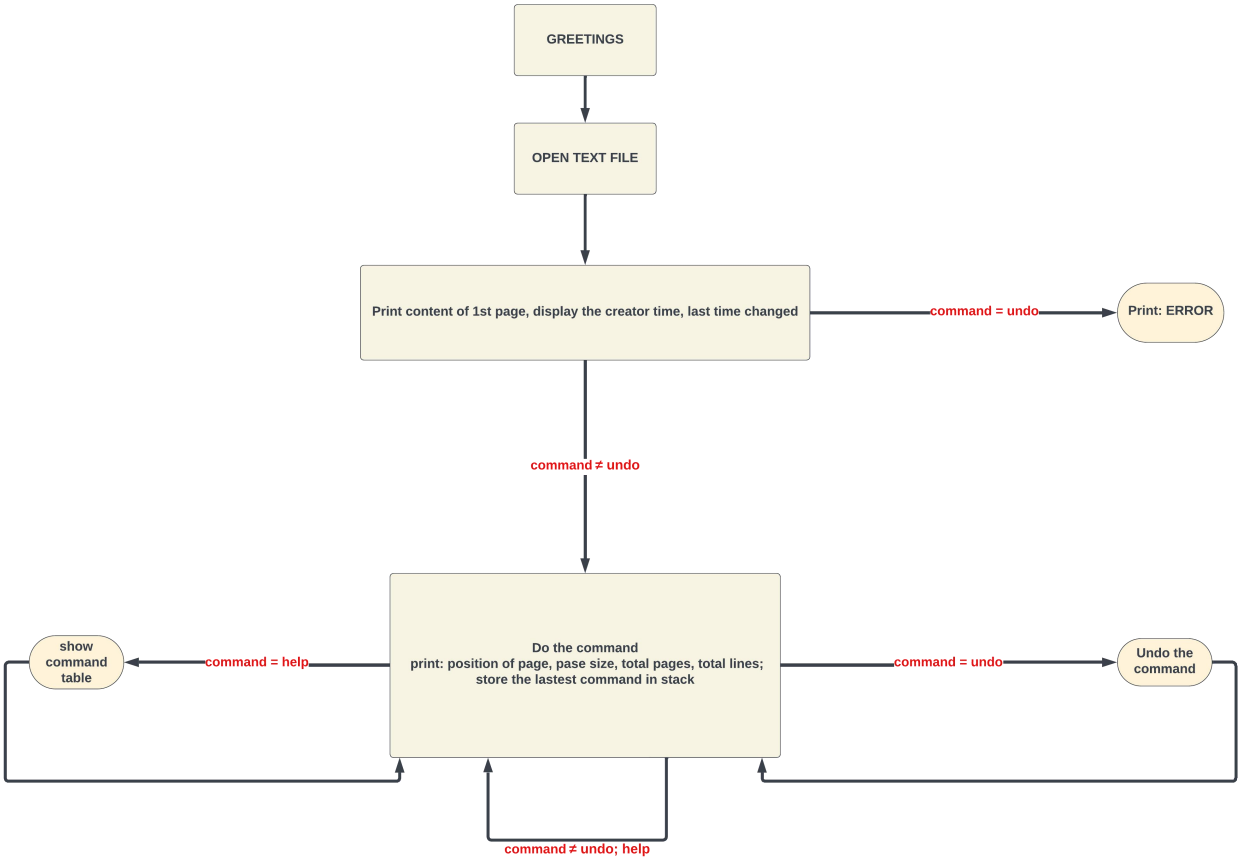
- The command that has 1 argument like ‘open text.txt’, we split it into 2 part “open” and “text.txt”
- The command that has 3 argument like ‘insert 3 contend’(insert the contend into node 3) , we split it into 3 part “insert” ; “3” ; “contend”.

  - Function ‘SplitStringByDelimeter’, it process the nearest function in undoStack, its operation similar to SplitSringBySpaceToArray, but it work based on character “#” instead of the space

For example :
- insert#3#contend

  - Function ‘ToLowerCase’, it convert the command into lowercase.

The flow chart of our program.



III. Main code

The main benchmarks code involved 6 files which are 3 cpp file and 3 header files. We also provide 1 text file to test our program.
- MyFunctions.cpp : All the functions to run the text editor
- intSLList.cpp : Implement data structure as Singly Links List

- main.cpp: main execute code.
- MyMessage.h : My message that I display on the console to guide the user.
- MyFunctions.h
- intSLList.h
- text.txt: a text file to test the text editor

MyFunctions.cpp

```
#include <string>
#include <sstream>
#include <fstream>
#include <iostream>
#include "intSLList.h"
#include "MyFunctions.h"

using namespace std;

string MyFunctions::LeftTrim(const string& s)
{
	size_t start = s.find_first_not_of(whitespaces);
	return (start == string::npos) ? "" : s.substr(start);
}

string MyFunctions::RightTrim(const std::string& s)
{
	size_t end = s.find_last_not_of(whitespaces);
	return (end == std::string::npos) ? "" : s.substr(0, end + 1);
}

string MyFunctions::LeftAndRightTrimSpaces(const string& s)
{
	return RightTrim(LeftTrim(s));
}

string MyFunctions::ToLowerCase(string input)
{
	// by using & does NOT create a copy of each element of input
	// but instead directly references and stores it in c as an alias.
	// This means that modifying c does modify input.
	for (char& c : input)
		c = tolower(c);

	return input;
}

int MyFunctions::PageCountCalculator(int lineCount, int pageSize) {

	int totalPageCount = (lineCount / pageSize);

	if (totalPageCount == 0)
		totalPageCount++;
	else if (lineCount % pageSize != 0)
		totalPageCount++;

	return totalPageCount;
}
```

```
string* MyFunctions::SplitStringBySpaceToArray(string data)
{
	int i = 0;

	string* input = new string[3];
	string temp = LeftAndRightTrimSpaces(data); // remove spaces from begin and end of the string

	stringstream ssin(temp);

	while (ssin.good() && i < 3) {
		ssin >> temp;
		if (i == 2)
		{

			data.replace(0, input[0].length(), "");
			data = LeftAndRightTrimSpaces(data);
			data.replace(0, input[1].length(), "");	input[i] =
LeftAndRightTrimSpaces(data);
		}
		else {
			input[i] = temp;
		}

		i++;
	}
	return input;
}

string* MyFunctions::SplitStringByDelimiterToArray(string s, string delim) {
	int i = 0;
	string* input = new string[5];
	int start = 0;
	int end = s.find(delim);

	while (end != string::npos)
	{
		input[i] = s.substr(start, static_cast<std::basic_string<char, std::char_traits<char>,
std::allocator<char>>::size_type>(end) - start);
		i++;
		start = end + delim.length();
		end = s.find(delim, start);
	}

	input[i] = s.substr(start, end);

	return input;
}
```

MyFunctions.h

```
#include <string>
#include "intSLList.h"
```

```
#ifndef MYFUNCTIONS_H
#define MYFUNCTIONS_H


using namespace std;

class MyFunctions {
public:
	string LeftTrim(const string& s);
	string RightTrim(const string& s);
	int PageCountCalculator(int, int);
	string* SplitStringBySpaceToArray(string data);
	string* SplitStringByDelimiterToArray(string, string);
	string LeftAndRightTrimSpaces(const string& s);
	string ToLowerCase(string);

private:
	string whitespaces = " \n\r\t\f\v";
};
#endif
```

intSLList.cpp

```
//************************  intSLList.cpp  **************************

#include <fstream>
#include <iostream>
#include "intSLList.h"

using namespace std;

IntSLList::~IntSLList() {
	for (IntSLLNode* p; !isEmpty(); ) {
		p = head->next;
		delete head;
		head = p;
	}
}

void IntSLList::addToHead(string el) {
	if (head == NULL || tail == NULL)
		head = tail = 0;

	head = new IntSLLNode(el, head);
	if (tail == 0)
		tail = head;
}

void IntSLList::addToTail(string el) {
	if (head == NULL || tail == NULL)
		head = tail = 0;

	if (tail != 0) {	// if list not empty;
		tail->next = new IntSLLNode(el);
```

```
                tail = tail->next;
        }
        else head = tail = new IntSLLNode(el);
}

string IntSLList::deleteFromHead() {
        string deletedText = head->info;
        head = head->next;
        return deletedText;
}

string IntSLList::deleteFromTail() {
        if (head == NULL)
                return " ";

        if (head->next == NULL)
                return deleteFromHead();

        IntSLLNode* second_last = head;
        while (second_last->next->next != NULL)
                second_last = second_last->next;    // Find the second last node

        delete (second_last->next);             // Delete last node


        second_last->next = NULL;               // Change next of second last

        return " ";
}

void IntSLList::deleteAll() {
        if (head != NULL)
        {
                IntSLLNode* current = head;
                IntSLLNode* next;

                while (current != NULL)
                {
                        next = current->next;
                        free(current);
                        current = next;
                }

                head = NULL;
        }
}

void IntSLList::moveNode(int currentPos, int targetPos)
{
        if (currentPos != targetPos)
        {
                IntSLLNode* tmp = head;

                string nodeTxt = findNodeElementText(currentPos);
                deleteNode(currentPos);
                insertNode(targetPos, nodeTxt);
        }
}
```

```
string IntSLList::findNodeElementText(int pos)
{
	IntSLLNode* tmp = head;
	string nodeTxt = "";

	// itterate over a node until the pos value reached
	for (int i = 0; tmp != NULL && i < pos; i++)
	{
		if ((pos - 1) == i)
			nodeTxt = tmp->info;

		tmp = tmp->next;
	}
	return nodeTxt;
}

string IntSLList::deleteNode(int pos)
{
	string deletedText = " ";

	if (head != NULL)
	{
		IntSLLNode* previousNodeOFTheNodeToBeDeleted = head;

		if (pos == 1) //if head needs to be removed
			return deleteFromHead();

		for (int i = 1; head != NULL && i < pos - 1; i++)	// Find previous node of the node to
be deleted
			previousNodeOFTheNodeToBeDeleted = previousNodeOFTheNodeToBeDeleted-
>next;

		deletedText = previousNodeOFTheNodeToBeDeleted->next->info; // can not be null as we
expect filled txt file
		IntSLLNode* next = previousNodeOFTheNodeToBeDeleted->next->next;	// Store
pointer to the next of node to be deleted
		delete(previousNodeOFTheNodeToBeDeleted->next);		// Delete the node from
linked list
		previousNodeOFTheNodeToBeDeleted->next = next;		// link next to new node
	}

	return deletedText;
}

string IntSLList::replaceNode(int pos, string newText)
{
	string replacedText = "";
	if (head == NULL)
		return replacedText;

	IntSLLNode* tmp = head;

	for (int i = 0; tmp != NULL && i < pos; i++)
	{
		replacedText = tmp->info;
		if ((pos - 1) == i)
			tmp->info = newText;

		tmp = tmp->next;
```

```
	}
	return replacedText;
}

void IntSLList::insertNode(int pos, string newText)
{

	if (pos == 1)
	{
		addToHead(newText);
	}
	else
	{
		if (head != NULL)
		{
			IntSLLNode* tmp = head;
			IntSLLNode* newNode = new IntSLLNode(newText);

			for (int i = 1; head != NULL && i < pos; i++)
			{
				if ((pos - 1) == i)
					newNode->next = tmp->next;
				else
					tmp = tmp->next;
			}

			tmp->next = newNode;
		}
	}

}

void IntSLList::printAll(int page, int size, int totalPageCount, int totalLineCount) {
	int i = 0;
	int lineCounter = 0;

	cout << endl;

	for (IntSLLNode* tmp = head; tmp != 0; tmp = tmp->next)
	{
		i++;
		if (((page - 1) * size) < i)
		{
			if (lineCounter < size)
			{
				lineCounter++;
				cout << lineCounter + ((page - 1) * size) << " - " << tmp->info << " " <<
endl;
			}

			if (lineCounter == size)
				break;
		}
	}
	cout << endl << "Current Page Number : " << page << " of " << totalPageCount << " --> ( " <<
totalLineCount << " lines )" << endl;
}

void IntSLList::save(string userFileName) const {
```

```
	ofstream MyFile(userFileName);
	if (MyFile.is_open())
	{
		for (IntSLLNode* tmp = head; tmp != 0; tmp = tmp->next)
			MyFile << tmp->info << " " << endl;
		MyFile.close();
	}
}
```

intSLList.h

```
//************************  intSLList.h  *************************
//	singly-linked list class to store integers
#include <string>

#ifndef INT_LINKED_LIST
#define INT_LINKED_LIST


using namespace std;

class IntSLLNode {
public:
	IntSLLNode() {
		next = 0;
	}
	IntSLLNode(string el, IntSLLNode* ptr = 0) {
		info = el; next = ptr;
	}
	string info = "";
	IntSLLNode* next = NULL;
};

class IntSLList {
public:
	IntSLList() {
		head = tail = 0;
	}
	~IntSLList();
	bool isEmpty() {
		return head == 0;
	}
	void addToHead(string);
	void addToTail(string);
	void deleteAll();
	string findNodeElementText(int);
	string deleteFromHead(); // delete the head and return its info;
	string deleteFromTail();
	string deleteNode(int);
	void moveNode(int, int);
	string replaceNode(int, string);
	void insertNode(int, string);
	void printAll(int, int, int, int);
	void save(string) const;
```

```
private:
	IntSLLNode* head, * tail;
};

#endif#pragma once
```

main.cpp

```
#include <stack>
#include <string>
#include <fstream>
#include <iostream>
#include <algorithm>
#include "intSLList.h"
#include "MyMessages.h"
#include "MyFunctions.h"
#include <windows.h>
#include <string>


using namespace std;

MyMessages msg;
IntSLList mainList;
string* splittedCommand;
stack<string> undoStack;
MyFunctions myFunctions;
string nameOfFile, command, userNewTextInput, userNewFileNameToSave, deletedText, undoAction,
originalTextOfNode, originalFileName;

int currentPage = 1, cur = 0, pageSize = 10, totalLinesAdded = 0, totalPageCount = 0, totalLineCount = 0,
userPositionInput1 = 0, userPositionInput2 = 0;

bool readFile(string filename)
{
	ifstream testFile(filename);

	if (testFile.good())
	{
		originalFileName = filename;
		//init list,stack and file details
		mainList.deleteAll();

		while (!undoStack.empty())
			undoStack.pop();

		string line;
		currentPage = 1;
		totalPageCount = 0;
		totalLineCount = 0;

		// fill list with the lines read from txt
		while (getline(testFile, line)) {
			mainList.addToTail(line);
			totalLineCount++;
```

```
		}

		if (totalLineCount > 0)
				return true;
		else
			cout << msg.errEmptyFile;
	}
	cout << msg.errNoSuchFile;
	return false;
}

bool openMyFile(string fileName)
{
	fileName = myFunctions.ToLowerCase(fileName);
	std::string fullPath = "D:\\" + fileName;
	const char* filePath = fullPath.c_str();
	if (readFile(fileName))
	{
		totalPageCount = myFunctions.PageCountCalculator(totalLineCount, pageSize);
		mainList.printAll(currentPage, pageSize, totalPageCount, totalLineCount);
		//std::string filePath = "path/to/your/file.txt";
		//std::ifstream file(filePath);
		//cout << fileName;
		HANDLE hFile = CreateFileA(filePath, GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

		//if (hFile == INVALID_HANDLE_VALUE) {
			//cerr << "Unable to open the file. Error: " << GetLastError() << endl;
			//return main();
		//}
		FILETIME creationTime, lastAccessTime, lastWriteTime;
		if (GetFileTime(hFile, &creationTime, &lastAccessTime, &lastWriteTime)) {
			// Convert time from UTC to local time for creation time
			SYSTEMTIME stCreation;
			FileTimeToSystemTime(&creationTime, &stCreation);

			SYSTEMTIME localCreationTime;
			if (SystemTimeToTzSpecificLocalTime(NULL, &stCreation,
&localCreationTime)) {
				cout << "File creation time: " << localCreationTime.wYear << "/" <<
localCreationTime.wMonth << "/" << localCreationTime.wDay << " "
					<< localCreationTime.wHour << ":" <<
localCreationTime.wMinute << ":" << localCreationTime.wSecond << endl;
			}
			else {
				cerr << "Unable to convert creation time to local time. Error: " <<
GetLastError() << endl;
			}

			// Convert time from UTC to local time for last write time
			SYSTEMTIME stLastWrite;
			FileTimeToSystemTime(&lastWriteTime, &stLastWrite);

			SYSTEMTIME localLastWriteTime;
			if (SystemTimeToTzSpecificLocalTime(NULL, &stLastWrite,
&localLastWriteTime)) {
				cout << "File last write time: " << localLastWriteTime.wYear << "/" <<
localLastWriteTime.wMonth << "/" << localLastWriteTime.wDay << " "
					<< localLastWriteTime.wHour << ":" <<
```

```
localLastWriteTime.wMinute << ":" << localLastWriteTime.wSecond << endl;
                                }
                                else {
                                        cerr << "Unable to convert last write time to local time. Error: " <<
GetLastError() << endl;
                                }
                        }
                        else {
                                cerr << "Unable to retrieve time information. Error: " << GetLastError() << endl;
                        }

                        // Close the file handle
                        CloseHandle(hFile);
                        return true;
                }

                cout << msg.errNotValidOpenCommand;
                return false;
}

void updatePageToNavigate()
{
                currentPage = userPositionInput1 / pageSize;

                if (currentPage == 0)
                        currentPage = 1;
                else if ((userPositionInput1 % pageSize) != 0)
                        currentPage++;
}

int main()
{
                // Please first read the ReadMe.txt
                cout << msg.introMessage << endl;

                bool anyFileOpened = false;

                while (!anyFileOpened)     // open file
                {
                        cout << msg.infoOpenFile;
                        getline(cin, command);

                        string* userInputSplittedBySpace = myFunctions.SplitStringBySpaceToArray(command);

                        command = userInputSplittedBySpace[0];
                        nameOfFile = userInputSplittedBySpace[1];
                        command = myFunctions.ToLowerCase(command);

                        anyFileOpened = openMyFile(nameOfFile);
                }

                while (true)          // operations on file opened
                {
                        cout << msg.infoEnterCommand;
                        getline(cin, command);

                        // Splitting user commands into an array of string to parse user request
                        splittedCommand = myFunctions.SplitStringBySpaceToArray(command);
```

```
		command = splittedCommand[0];
		command = myFunctions.ToLowerCase(command);

		if (command == "help")
		{
			cout << msg.infoCommandListHelper;
		}

		else if (command == "undo")
		{
			// We will feed stack with a unique combination using #
			// for example if user will enter insert 5 asd  we will store this in stack as
			// insert#5#asd
			// then when we will need undo, we will split this string to array by '#'
			if (undoStack.empty())
				cout << msg.errNoUndoActionFound;
			else
			{
				undoAction = undoStack.top();

				// if any action found, split it by '#'
				string* splittedAction =
myFunctions.SplitStringByDelimiterToArray(undoAction, "#");

				// After split, we will get the action details
				string command = splittedAction[0]; // this will be always the command
details
				string undoParam1 = splittedAction[1]; // sometime this will be int, some
times text
				string undoParam2 = splittedAction[2]; // sometime this will be int, some
times text

				if (command == "delete") // if user deleted then we should insert it again
				{
					userPositionInput1 = stoi(undoParam1);
					originalTextOfNode = undoParam2;

					mainList.insertNode(userPositionInput1, originalTextOfNode);

					totalLineCount++;
					updatePageToNavigate();
				}

				else if (command == "insert") // if user inserted then we should delete it /
them again
				{
					userPositionInput1 = stoi(undoParam1);
					totalLinesAdded = stoi(undoParam2);

					if (totalLinesAdded != 1) // if multiple lines added with some of
them spaces
					{
						for (int i = 0; i < totalLinesAdded; i++)
						{
							mainList.deleteFromTail(); // delete all the
lines one by one from tail
							totalLineCount--;
						}
						currentPage = (userPositionInput1 - totalLinesAdded) /
```

```
pageSize;

                                        if (currentPage == 0)
                                                currentPage = 1;
                                        else if (((userPositionInput1 - totalLinesAdded) %
pageSize) != 0)
                                                currentPage++;
                                }
                                else // if just 1 line added then delete that specific line
                                {
                                        if (userPositionInput1 > 0 && userPositionInput1 <=
totalLineCount)
                                        {
                                                mainList.deleteNode(userPositionInput1);
                                                totalLineCount--;
                                        }
                                        else
                                                cout << msg.errNotValidPosition <<
totalLineCount << " !!!\n";
                                }
                        }

                        else if (command == "move") // reverse the move action
                        {
                                userPositionInput1 = stoi(undoParam1);
                                userPositionInput2 = stoi(undoParam2);

                                mainList.moveNode(userPositionInput2, userPositionInput1);

                                // after move request, we may need to move to the related page
to see the result
                                updatePageToNavigate();
                        }

                        else if (command == "replace") // reverse the line to its original text value
                        {
                                userPositionInput1 = stoi(undoParam1);
                                originalTextOfNode = undoParam2;

                                mainList.replaceNode(userPositionInput1, originalTextOfNode);

                                // after replace request, we may need to move to the related page
to see the result
                                updatePageToNavigate();
                        }

                        else if (command == "next") // go back to the page user commands next
                        {
                                // turn back to the page where user enters next
                                currentPage = stoi(undoParam1);

                                // if you want to trigger oposite of next (prev) then uncomment
below code
                                //if (currentPage > 1)
                                //      currentPage--;
                        }

                        else if (command == "prev") // go back to the page user commands next
                        {
```

```
                                        // turn back to the page where user enters prev
                                        currentPage = stoi(undoParam1);

                                        // if you want to trigger oposite of prev (next) then uncomment
below code
                                        //if (currentPage < totalPageCount)
                                        //      currentPage++;
                                }

                                totalPageCount = myFunctions.PageCountCalculator(totalLineCount,
pageSize);
                                mainList.printAll(currentPage, pageSize, totalPageCount,
totalLineCount);
                                undoStack.pop();

                                cout << msg.infoUndoComplete << command << " " << undoParam1 <<
endl;
                        }
                }

                else if (command == "insert")
                {
                        try
                        {
                                userNewTextInput = splittedCommand[2];
                                userPositionInput1 = stoi(splittedCommand[1]);

                                // input position must be greater than 0
                                if (userPositionInput1 > 0 && userPositionInput1 <= 2147483647) // int
max, not sugested :)
                                {
                                        // lines added with spaces + 1 (the last line added)
                                        // if totalLinesAdded is 1 then we understand that just 1 line
added
                                        // if it is more than 1 then we understand that some lines added
with spaces
                                        totalLinesAdded = userPositionInput1 - totalLineCount;

                                        if (totalLinesAdded <= 1)
                                        {
                                                totalLinesAdded = 1; // only 1 line added
                                                mainList.insertNode(userPositionInput1,
userNewTextInput);
                                                totalLineCount++; // due to insertion, line count
increases
                                        }
                                        else
                                        {
                                                string tempText = " ";
                                                for (int i = 1; i <= totalLinesAdded; i++) // (count of
diff) so multiple lines added
                                                {
                                                        totalLineCount++; // due to insertion, line
count increases
                                                        if (i == totalLinesAdded)
                                                                tempText = userNewTextInput; // just
the very last line will have text value
                                                        // middle lines will be added with spaces
                                                        mainList.insertNode(totalLineCount,
```

```
tempText);
                                                                }
                                                        }
                                                        totalPageCount =
myFunctions.PageCountCalculator(totalLineCount, pageSize);

                                                        // after insertions, we may need to move next pages
                                                        // if new pages added, then we will navigate user to the page at
position he/she inserted
                                                        // so currentpage recalculated
                                                        updatePageToNavigate();

                                                        mainList.printAll(currentPage, pageSize, totalPageCount,
totalLineCount);
                                                        undoStack.push(command + "#" + to_string(userPositionInput1)
+ "#" + to_string(totalLinesAdded));
                                                }
                                                else
                                                        cout << msg.errNotValidPosition << "2147483647 !!!\n";
                                        }
                                        catch (exception ex)
                                        {
                                                cout << msg.errNotValidPosition << totalLineCount << "!!!\n";
                                        }
                                }

                                else if (totalLineCount == 0)
                                {
                                        cout << msg.infoInseretLine;
                                }

                                else if (command == "open")
                                {
                                        nameOfFile = splittedCommand[1];
                                        openMyFile(nameOfFile);
                                }

                                else if (command == "next")
                                {
                                        cur = currentPage;
                                        currentPage++;

                                        // if we are already at the last page
                                        if (currentPage > totalPageCount)
                                        {
                                                currentPage = totalPageCount;
                                                cout << msg.infoLastPage;
                                        }

                                        mainList.printAll(currentPage, pageSize, totalPageCount, totalLineCount);
                                        undoStack.push(command + "#" + to_string(cur));
                                }

                                else if (command == "prev")
                                {
                                        cur = currentPage;
                                        currentPage--;
                                        // check if we are already at the first page
                                        if (currentPage < 1)
```

```
{
currentPage = 1;
cout << msg.infoFirstPage;
}

mainList.printAll(currentPage, pageSize, totalPageCount, totalLineCount);
undoStack.push(command + "#" + to_string(cur));
}

else if (command == "delete")
{
try
{
userPositionInput1 = stoi(splittedCommand[1]);

// input position must be between 1 and total line count
if (userPositionInput1 > 0 && userPositionInput1 <= totalLineCount)
{
deletedText = mainList.deleteNode(userPositionInput1);

totalLineCount--; // if any node deleted, this means lines are
decreasing

// maybe too many lines deleted, so we need to update the total
page count
totalPageCount =
myFunctions.PageCountCalculator(totalLineCount, pageSize);

// after deletions, we may need to move that pages to see the
action result.
// if pages deleted, then we will navigate user to the page to the 1
previous page
// so currentpage recalculated
updatePageToNavigate();

// if too many lines are deleted, and user is at that last page
// which does not exists anymore, we will navigate user the last
page.
if (currentPage > totalPageCount)
currentPage = totalPageCount;

mainList.printAll(currentPage, pageSize, totalPageCount,
totalLineCount);
cout << "\nLine " << userPositionInput1 << " is deleted." <<
endl;
undoStack.push(command + "#" + to_string(userPositionInput1)
+ "#" + deletedText);
}
else
cout << msg.errNotValidPosition << totalLineCount << " !!!\n";
}
catch (exception ex)
{
cout << msg.errNotValidPosition << totalLineCount << " !!!\n";
}
}

else if (command == "replace") {
try
```

```
{
	userNewTextInput = splittedCommand[2];
	userPositionInput1 = stoi(splittedCommand[1]);

	// input values must be between 1 and total line count
	if (userPositionInput1 > 0 && userPositionInput1 <= totalLineCount)
	{
		originalTextOfNode =
mainList.replaceNode(userPositionInput1, userNewTextInput);

		// after replace request, we may need to move to the related page
to see the result
		updatePageToNavigate();

		mainList.printAll(currentPage, pageSize, totalPageCount,
totalLineCount);
		undoStack.push(command + "#" + to_string(userPositionInput1)
+ "#" + originalTextOfNode);
	}
	else
		cout << msg.errNotValidPosition << totalLineCount << " !!!\n";
}
catch (exception ex)
{
	cout << msg.errNotValidPosition << totalLineCount << "!!!\n";
}
}

else if (command == "move")
{
	try
	{
		userPositionInput1 = stoi(splittedCommand[1]);
		userPositionInput2 = stoi(splittedCommand[2]);

		// input values must be between 1 and total line count
		if (userPositionInput2 > totalLineCount || userPositionInput1 >
totalLineCount || userPositionInput1 < 1 || userPositionInput2 < 1)
		{
			cout << msg.errNotValidPosition << totalLineCount << "!!!\n";
		}
		else
		{
			mainList.moveNode(userPositionInput1, userPositionInput2);

			// after move request, we may need to move to the related page
to see the result
			currentPage = userPositionInput2 / pageSize;

			if (currentPage == 0)
				currentPage = 1;
			else if ((userPositionInput2 % pageSize) != 0)
				currentPage++;
			mainList.printAll(currentPage, pageSize, totalPageCount,
totalLineCount);
			undoStack.push(command + "#" + to_string(userPositionInput1)
+ "#" + to_string(userPositionInput2));
		}
	}
```

```
                catch (exception ex)
                {
                        cout << msg.errNotValidPosition << totalLineCount << "!!!\n";
                }
        }

        else if (command == "save")
        {
                // If any name given by user, a new file will be created and filled with data
                // If just save comment sent by the user, current file will be overwritten

                userNewFileNameToSave = splittedCommand[1];

                if (userNewFileNameToSave == "")
                        userNewFileNameToSave = originalFileName;

                mainList.save(userNewFileNameToSave);
                cout << userNewFileNameToSave << " created / modified." << endl;
        }

        else
        {
                cout << msg.errNotValidCommand;
        }
    }
}
```

text.txt

```
This is the line 1
This is the line 2
This is the line 3
This is the line 4
This is the line 5
This is the line 6
This is the line 7
This is the line 8
This is the line 9
This is the line 10
This is the line 11
This is the line 12
This is the line 13
This is the line 14
This is the line 15
This is the line 16
This is the line 17
This is the line 18
This is the line 19
This is the line 20
This is the line 21
This is the line 22
This is the line 23
This is the line 24
This is the line 25
This is the line 26
This is the line 27
This is the line 28
```

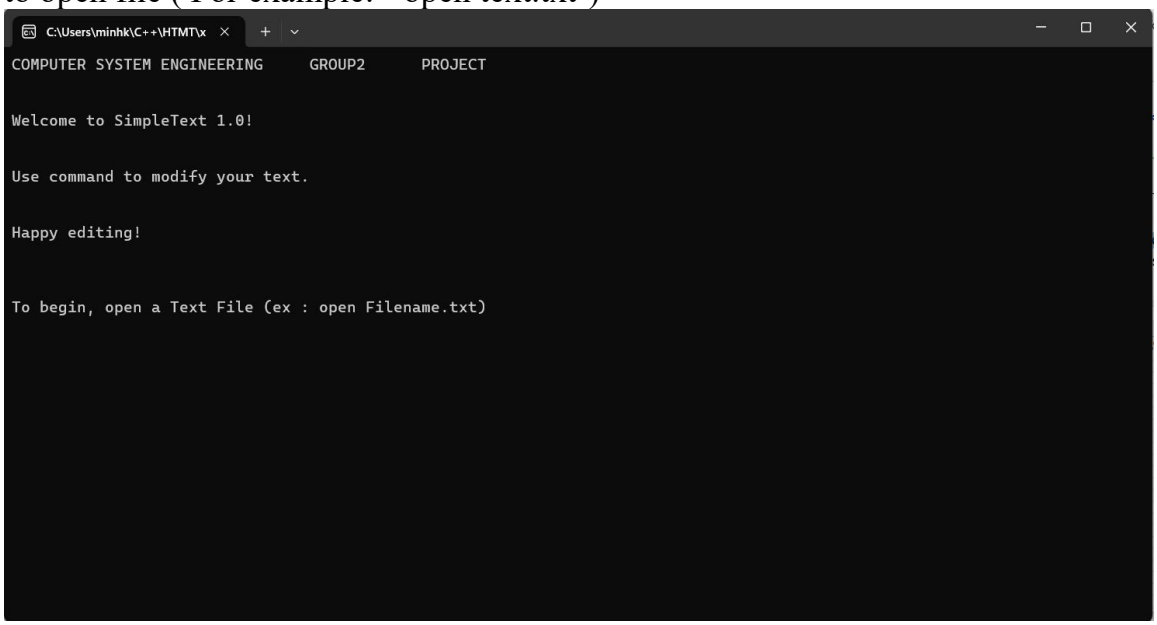
This is the line 29
This is the line 30
This is the line 31
This is the line 32
This is the line 33
This is the line 34
This is the line 35
This is the line 36
This is the line 37
This is the line 38
This is the line 39
This is the line 40
This is the line 41
This is the line 42
This is the line 43
This is the line 44
This is the line 45
This is the line 46
This is the line 47
This is the line 48
This is the line 49
This is the line 50
This is the line 51
This is the line 52
This is the line 53
This is the line 54
This is the line 55
This is the line 56
This is the line 57
This is the line 58
This is the line 59
This is the line 60
This is the line 61
This is the line 62
This is the line 63
This is the line 64
This is the line 65
This is the line 66
This is the line 67
This is the line 68
This is the line 69
This is the line 70
This is the line 71
This is the line 72
This is the line 73
This is the line 74
This is the line 75
This is the line 76
This is the line 77
This is the line 78
This is the line 79
This is the line 80
This is the line 81
This is the line 82
This is the line 83
This is the line 84
This is the line 85
This is the line 86
This is the line 87

This is the line 88
This is the line 89
This is the line 90
This is the line 91
This is the line 92
This is the line 93
This is the line 94
This is the line 95
This is the line 96
This is the line 97
This is the line 98
This is the line 99
This is the line 100
This is the line 101
This is the line 102
This is the line 103
This is the line 104
This is the line 105
This is the line 106
This is the line 107
This is the line 108
This is the line 109
This is the line 110
This is the line 111
This is the line 112
This is the line 113
This is the line 114
This is the line 115
This is the line 116
This is the line 117
This is the line 118
This is the line 119
This is the line 120
This is the line 121
This is the line 122
This is the line 123
This is the line 124
This is the line 125
This is the line 126
This is the line 127
This is the line 128
This is the line 129
This is the line 130
This is the line 131
This is the line 132
This is the line 133
This is the line 134
This is the line 135
This is the line 136
This is the line 137
This is the line 138
This is the line 139
This is the line 140
This is the line 141
This is the line 142
This is the line 143
This is the line 144
This is the line 145
This is the line 146

This is the line 147
This is the line 148
This is the line 149
This is the line 150
This is the line 151
This is the line 152
This is the line 153
This is the line 154
This is the line 155
This is the line 156
This is the line 157
This is the line 158
This is the line 159
This is the line 160
This is the line 161
This is the line 162
This is the line 163
This is the line 164
This is the line 165
This is the line 166
This is the line 167
This is the line 168
This is the line 169
This is the line 170
This is the line 171
This is the line 172
This is the line 173
This is the line 174
This is the line 175
This is the line 176
This is the line 177
This is the line 178
This is the line 179
This is the line 180
This is the line 181
This is the line 182
This is the line 183
This is the line 184
This is the line 185
This is the line 186
This is the line 187
This is the line 188
This is the line 189
This is the line 190
This is the line 191
This is the line 192
This is the line 193
This is the line 194
This is the line 195
This is the line 196
This is the line 197
This is the line 198
This is the line 199
This is the line 200

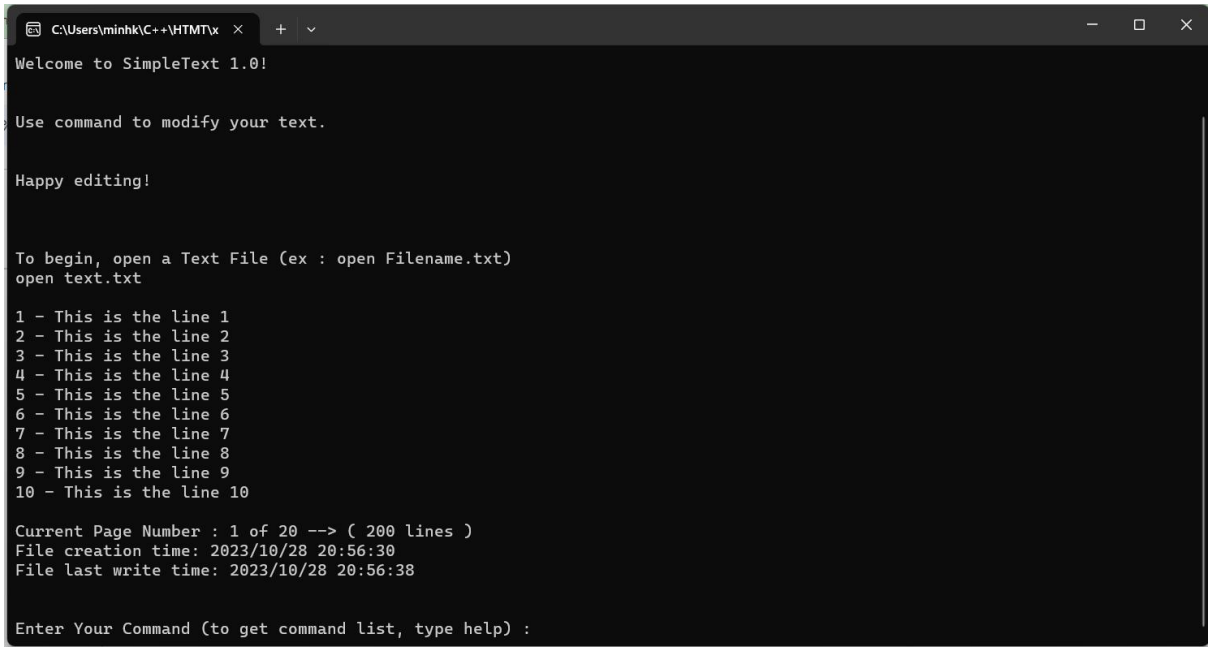
IV. Code execution

After running the benchmark, there will be messages on the console guiding the user to open a text file to start editing. User need to use command to open file (For example: ‘open text.txt’)



After opening the text file. This text editor will print the contents of a file to the screen, page by page.

Each page consists of 10 lines. Each line is 1 node of our link list. An example image is given below.



We have implemented a stack to keep track of all actions carried out, so you can undo actions if needed. We also write a function to track the time that file was created as well as the time that file was written.
This is all command our program provides:

```
C:\Users\minhk\C++\HTMT\x
9 - This is the line 9
10 - This is the line 10

Current Page Number : 1 of 20 --> ( 200 lines )
File creation time: 2023/10/28 20:56:30
File last write time: 2023/10/28 20:56:38


Enter Your Command (to get command list, type help) :  help


--------------------------------------------------------------------------------

 + Command List :

--------------------------------------------------------------------------------
 open                        - Open New File                 (ex : open filename.txt)
 next                        - Advance To Next Page
 prev                        - Return To Previous Page
 insert [line no] [new text] - Insert A New Line             (ex : insert 1 NewText)
 delete [line no]            - Delete A Line                 (ex : delete 1)
 replace [line no] [new text] - Replace A Line               (ex : replace 1 NewText)
 move [line no1] [line no2]  - Move Target Line To New Line  (ex : move 1 2)
 save                        - Save Or Create A New File
 undo                        - Undo Last Action
--------------------------------------------------------------------------------



Enter Your Command (to get command list, type help) :
```

Demonstration of some command:

- Delete node 1 :

```
C:\Users\minhk\C++\HTMT\x
 prev                        - Return To Previous Page
 insert [line no] [new text] - Insert A New Line             (ex : insert 1 NewText)
 delete [line no]            - Delete A Line                 (ex : delete 1)
 replace [line no] [new text] - Replace A Line               (ex : replace 1 NewText)
 move [line no1] [line no2]  - Move Target Line To New Line  (ex : move 1 2)
 save                        - Save Or Create A New File
 undo                        - Undo Last Action
--------------------------------------------------------------------------------



Enter Your Command (to get command list, type help) :  delete 1

1 - This is the line 2
2 - This is the line 3
3 - This is the line 4
4 - This is the line 5
5 - This is the line 6
6 - This is the line 7
7 - This is the line 8
8 - This is the line 9
9 - This is the line 10
10 - This is the line 11

Current Page Number : 1 of 20 --> ( 199 lines )

Line 1 is deleted.


Enter Your Command (to get command list, type help) :
```

- Replace contend of node 1 by “BK University”

```
C:\Users\minhk\C++\HTMT\x
4 - This is the line 5
5 - This is the line 6
6 - This is the line 7
7 - This is the line 8
8 - This is the line 9
9 - This is the line 10
10 - This is the line 11

Current Page Number : 1 of 20 --> ( 199 lines )

Line 1 is deleted.


Enter Your Command (to get command list, type help) :  replace 1 BK university

1 - BK university
2 - This is the line 3
3 - This is the line 4
4 - This is the line 5
5 - This is the line 6
6 - This is the line 7
7 - This is the line 8
8 - This is the line 9
9 - This is the line 10
10 - This is the line 11

Current Page Number : 1 of 20 --> ( 199 lines )


Enter Your Command (to get command list, type help) :
```

- Advance to the next page

```
C:\Users\minhk\C++\HTMT\x
2 - This is the line 3
3 - This is the line 4
4 - This is the line 5
5 - This is the line 6
6 - This is the line 7
7 - This is the line 8
8 - This is the line 9
9 - This is the line 10
10 - This is the line 11

Current Page Number : 1 of 20 --> ( 199 lines )


Enter Your Command (to get command list, type help) :  next

11 - This is the line 12
12 - This is the line 13
13 - This is the line 14
14 - This is the line 15
15 - This is the line 16
16 - This is the line 17
17 - This is the line 18
18 - This is the line 19
19 - This is the line 20
20 - This is the line 21

Current Page Number : 2 of 20 --> ( 199 lines )


Enter Your Command (to get command list, type help) :
```

V. CONCLUSION

In conclusion, the implementation of a simple text editor using a list ADT has been a valuable learning experience. Through this project, we have explored the fundamental concepts of data structures and their practical applications. The text editor efficiently handles text manipulation operations such as insertion, deletion, and searching, showcasing the power and flexibility of the list ADT. By successfully creating a functional text editor, we have gained a deeper understanding of how data structures can be leveraged to build useful and intuitive software tools. This project serves as a solid foundation for further

exploration and development in the field of data structures and C++ programming