SEMESTER

231

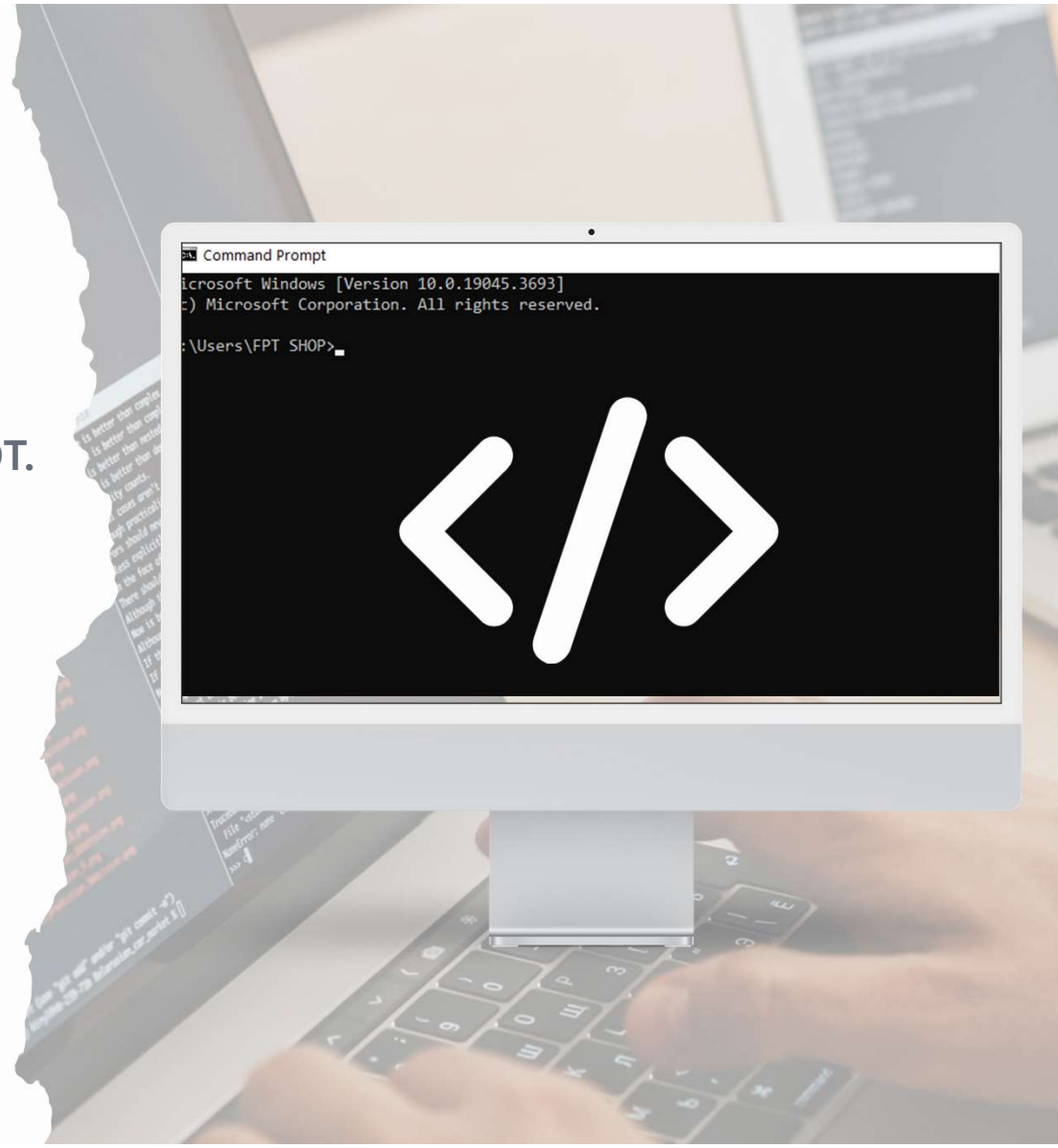
COMPUTER SYSTEMS ENGINEERING (EE3401)

Write a simple text editor using the list ADT.

STUDENT

Nguyễn Nhật Tuấn Minh	2151229
Trương Minh Khôi	2151216
Lương Hoàng Phúc	2010525
Châu Hồng Phước	2051175

PROJECT



CONTENTS

NEXT

PART 1

Introduction

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.

NEXT

WHAT IS THE TEXT EDITOR AND LIST ADT?

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.

NEXT

The List Abstract Data Type (ADT)

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 Abstract Data Type (ADT)

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 Abstract Data Type (ADT)

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

The List Abstract Data Type (ADT)

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.

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.

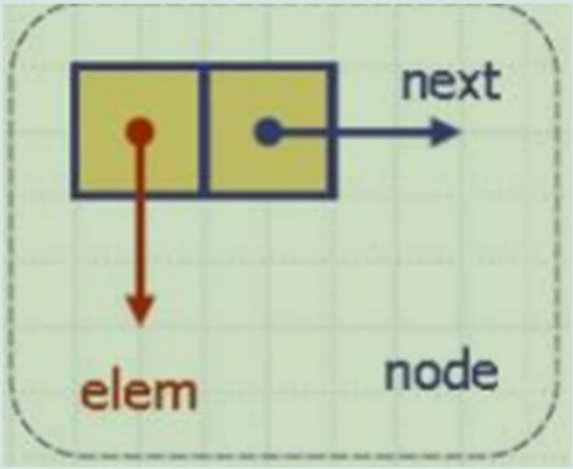
NEXT

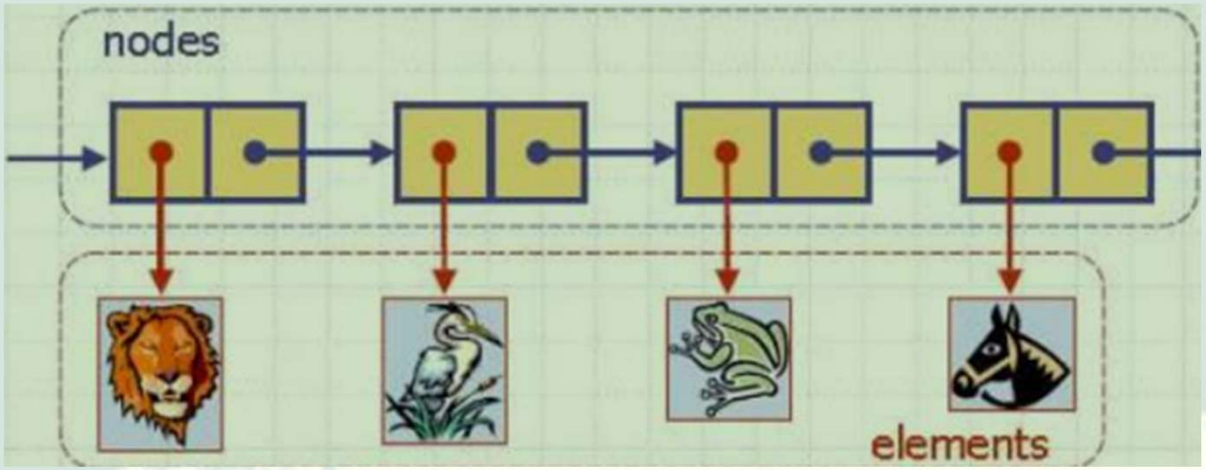
PART 2

Theory

THEORY

Define the single Link List structure



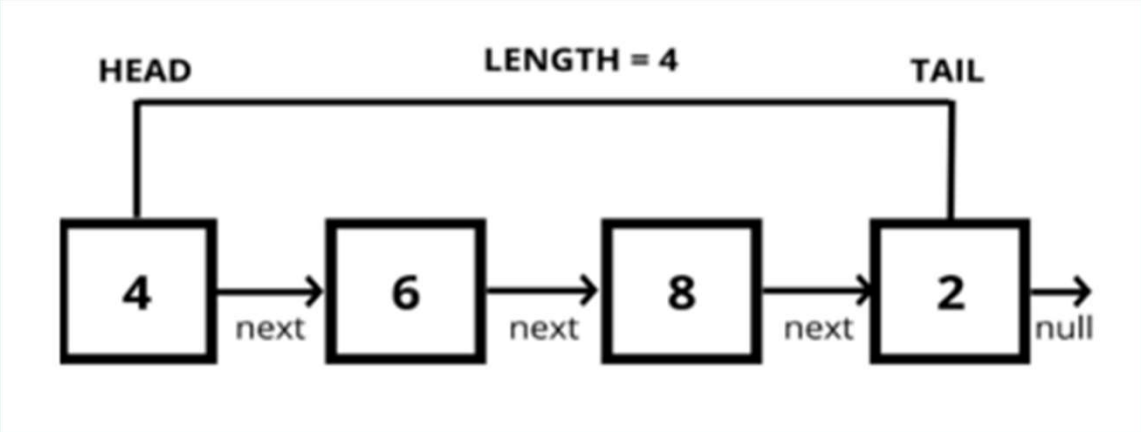


Create a structure or class called `IntSLLNode` that represents a node in the linked list.

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

THEORY

Define the single Link List structure



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

NEXT

THEORY

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 this program we have 3 different delete functions namely deleteFromTail , deleteFromHead, deleteNode which serve different purpose:

THEORY

Implement the data structure operation

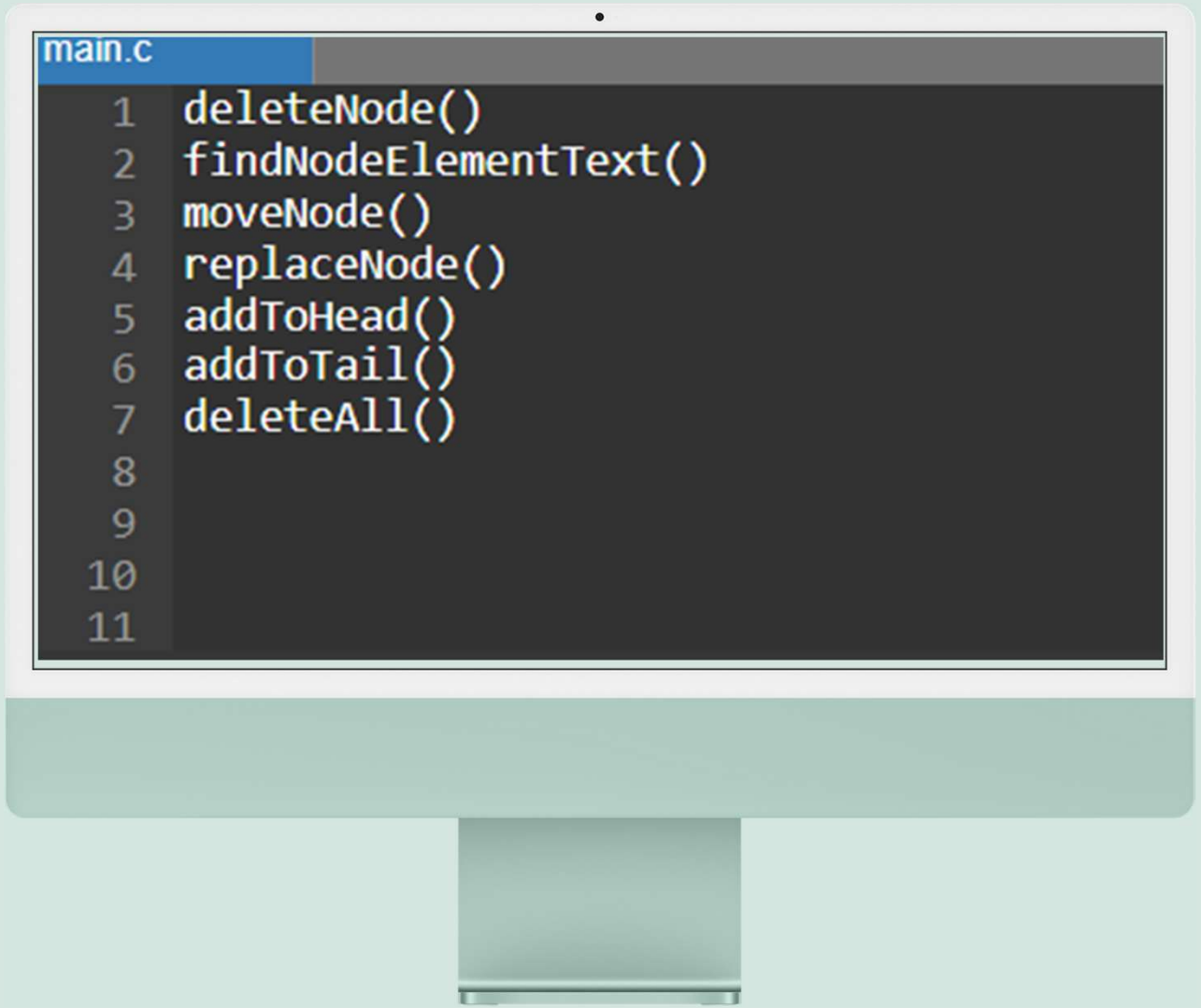
In this program we have 3 different delete functions namely ***deleteFromTail , deleteFromHead, deleteNode*** which serve different purpose:

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

NEXT


main.c
1 deleteNode()
2 findNodeElementText()
3 moveNode()
4 replaceNode()
5 addToHead()
6 addToTail()
7 deleteAll()
8
9
10
11

NEXT

THEORY

Implement the data structure operation

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.

THEORY

Implement the data structure operation

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.

NEXT

THEORY

Implement the user's command

To process the command from user, we construct the function 'MyFunctions'

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

THEORY

Implement the user's command

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

```
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);
}
```

NEXT

THEORY

Implement the user's command

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

```
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;
}
```

NEXT

THEORY

Implement the user's command

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".

THEORY

Implement the user's command

```
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;
}
```

Function '**SplitStringByDelimeter**', it process the nearest function in undoStack, its operation similar to SplitSringBySpaceToArray, but it work based on character "#" instead of the space

NEXT

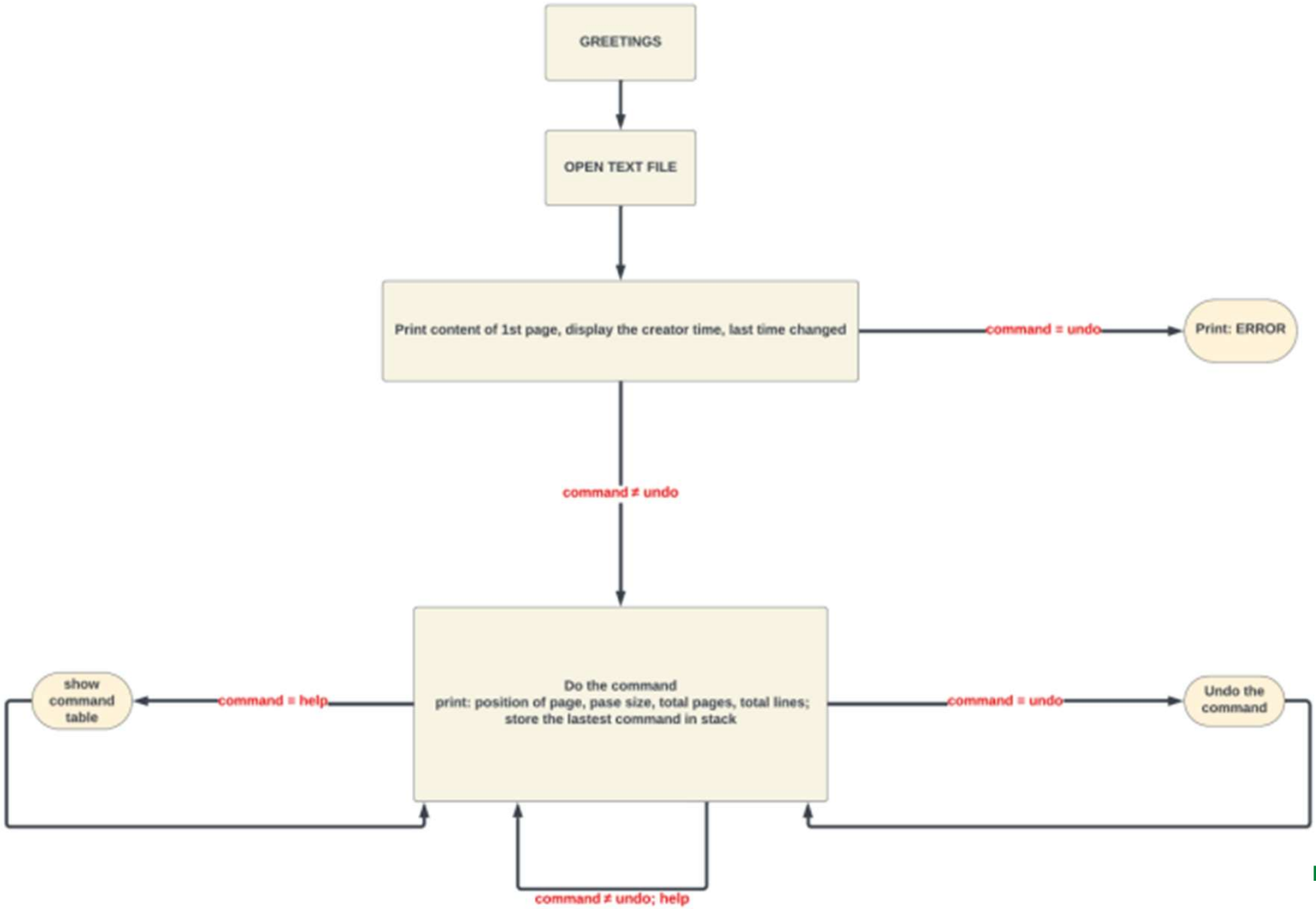
THEORY

Implement the user's command

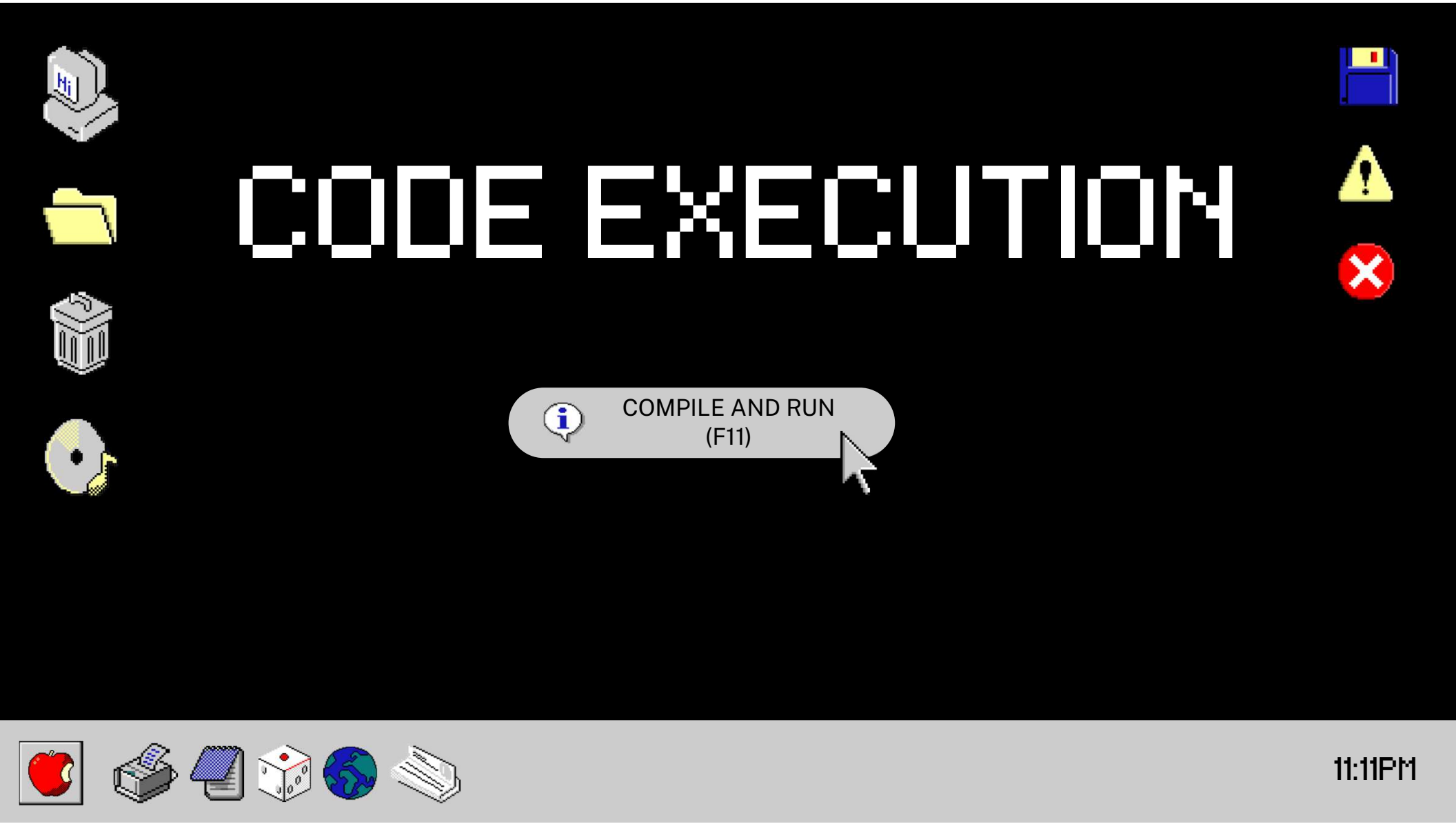
```
string MyFunctions::ToLowerCase(string input)
{
        // by using & does NOT create a copy of each element of input
        // but instead directly references and stores it in c as an alias.
        // This means that modifying c does modify input.
        for (char& c : input)
               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

        return input;
}
```

Function '**ToLowerCase**' converts the command into lowercase so as to avoid user command case sensitivity error.

NEXT


GREETINGS
OPEN TEXT FILE
Print content of 1st page, display the creator time, last time changed
command = undo
Print: ERROR
command ≠ undo
Do the command
print: position of page, pase size, total pages, total lines;
store the lastest command in stack
show command table
command = help
command = undo
Undo the command
command ≠ undo; help

NEXT

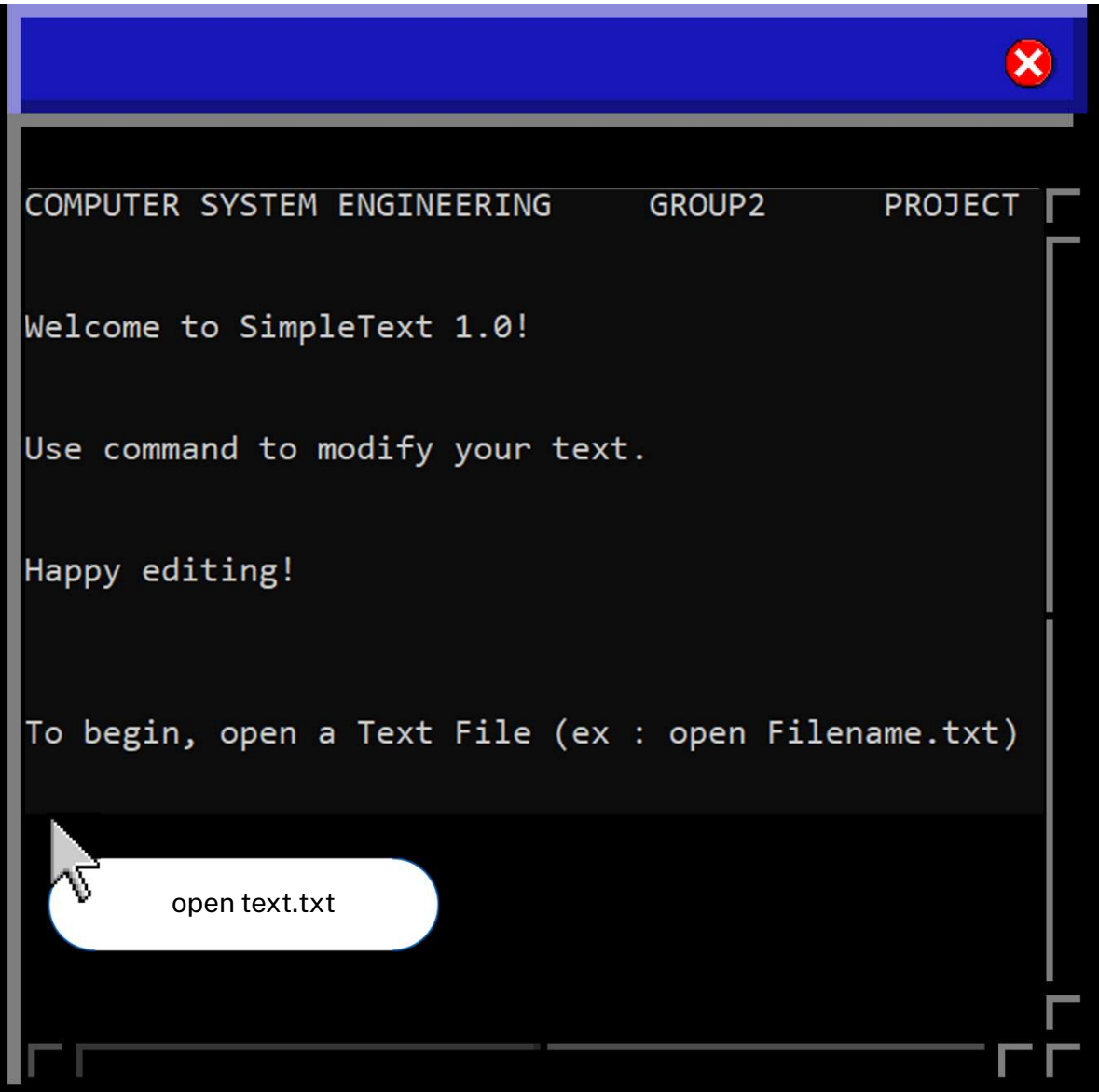

CODE EXECUTION
COMPILE AND RUN
(F11)
11:11PM

OPEN THE FILE

FOR EXAMPLE

We create a text document name “text.txt”

User need to use command to open file ‘open text.txt’



OPEN THE FILE

FOR EXAMPLE

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.

```
To begin, open a Text File (ex : open Filename.txt)
open text.txt

1 - Changed in line 0
2 - This is the line 1
3 - This is the line 2
4 - This is the line 3
5 - This is the line 4
6 - This is the line 5
7 - This is the line 6
8 - This is the line 7
9 - This is the line 8
10 - This is the line 9

Current Page Number : 1 of 21 --> ( 201 lines )
File creation time: 2023/11/24 10:21:53
File last write time: 2023/11/24 11:44:57


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

OPEN THE FILE

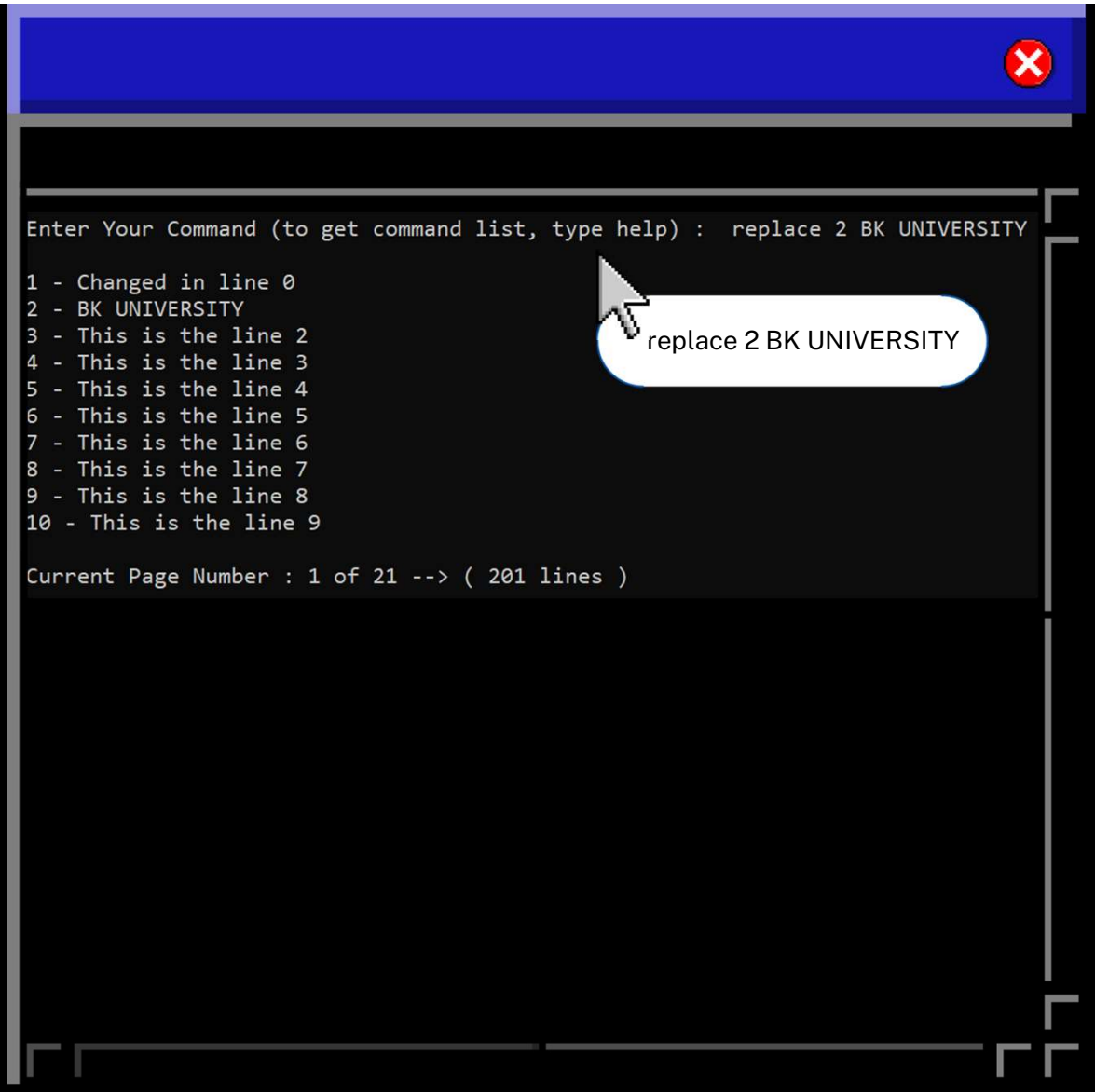
FOR EXAMPLE

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.

OPEN THE FILE

FOR EXAMPLE

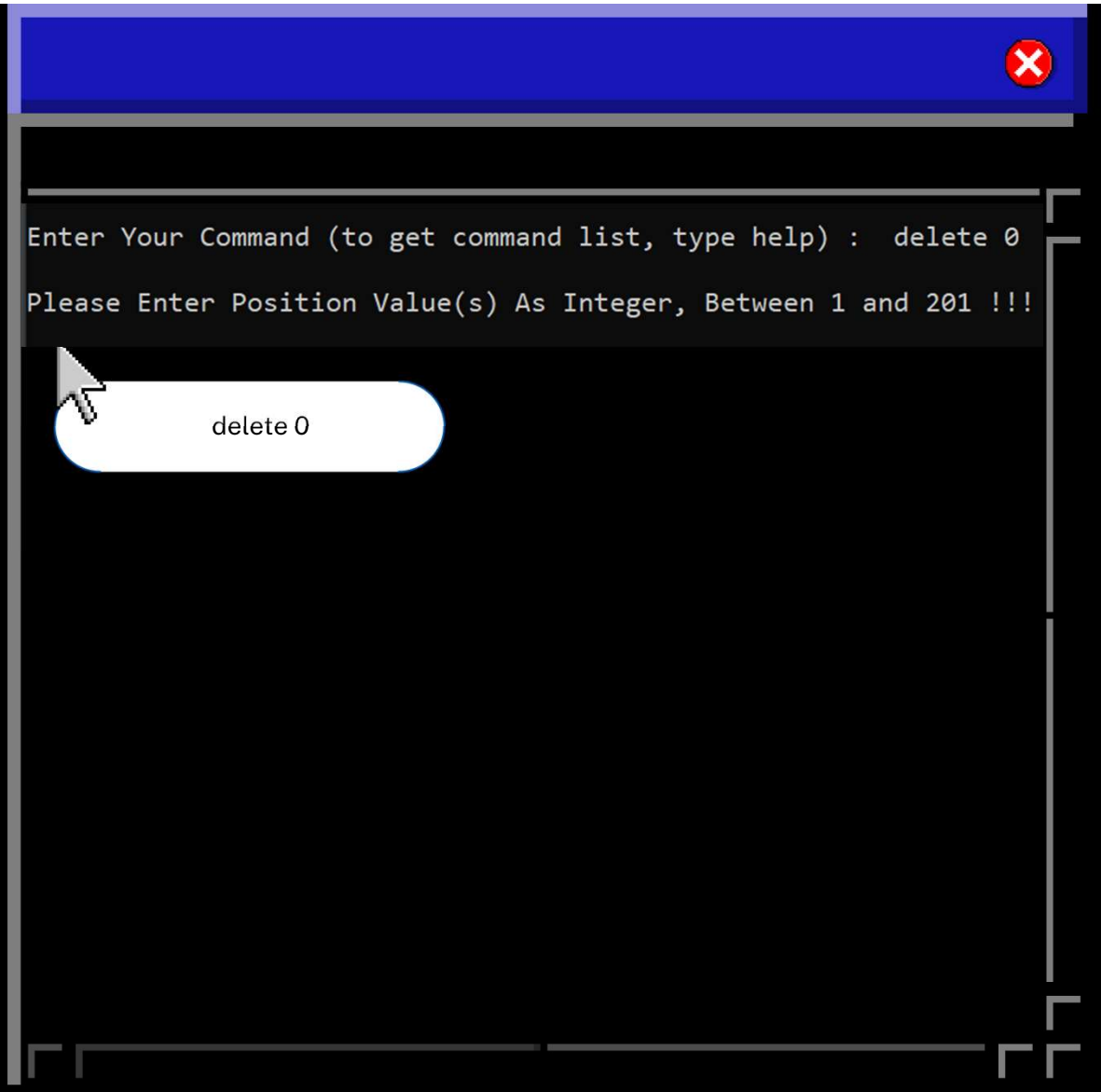
Replace the test in line 2 by “BK UNIVERSITY”



OPEN THE FILE

FOR EXAMPLE

Delete the text in line 0



OPEN THE FILE

FOR EXAMPLE

Delete the text in line 1

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

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

Current Page Number : 1 of 20 --> ( 200 lines )

Line 1 is deleted.


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

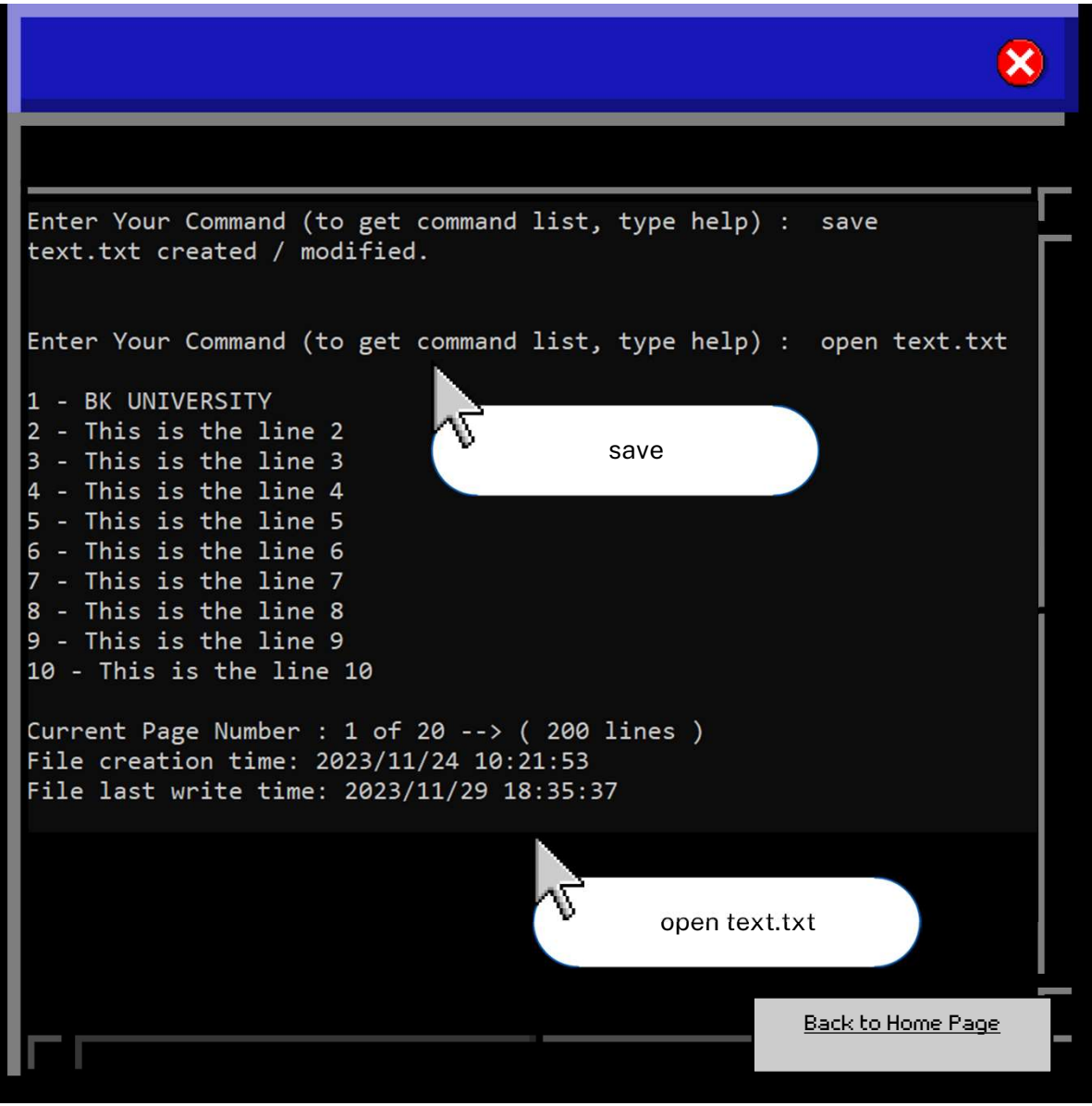
delete 1

OPEN THE FILE

FOR EXAMPLE

Save and reopen the file

It can be seen that the content of the file has been updated



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.

Back to Home Page

REFERENCE

1. Assoc. Prof. Truong Quang Vinh. (2023). Computer System Engineering. [Online]. Available: https://e-learning.hcmut.edu.vn/my/courses.php

2. “5.2. The List ADT — CS3 Data Structures & Algorithms,” opendsaserver.cs.vt.edu *https://opendsaserver.cs.vt.edu/ODSA/Books/CS3/html/ListADT.html?fbclid=IwAR254dvV1ycoxU_agwIP_0sxD2W82SXeiSoQbpma4jY-86fFSBdkh2Tteq8* (accessed Nov. 26, 2023).

Back to Home Page

THANK YOU