**FACULTY OF COMPUTING**

SEMESTER 1 2024/2025

**ASSIGNMENT II**

**SECJ2013 - DATA STRUCTURE AND ALGORITHM**

SECTION 02

**LECTURER: DR ZURAINI BINTI ALI SHAH**

| NAME | MATRIC NUMBER |
|---|---|
| DHESHIEGHAN A/L SARAVANA MOORTHY | A23CS0072 |
| LAU YAN KAI | A23CS0098 |
| TEH RU QIAN | A23CS0191 |
| NURUL ADRIANA BINTI KAMAL JEFRI | A23CS0258 |

# Table of Content

## Task 1 Understanding the Concept of Stack

### *I. Basic Operations of a Stack*

A **Stack** is a linear data structure that follows the **LIFO (Last-In-First-Out)** principle, meaning the last element added is the first to be removed.

**Push**: Adds an element to the top of the stack.

```
// Example of Push operation
stack.push(10); // Stack: [10]
stack.push(20); // Stack: [10, 20]
stack.push(30); // Stack: [10, 20, 30]
```

**Pop**: Removes the top element from the stack.

```
// Example of Pop operation
stack.pop(); // Removes 30, Stack: [10, 20]
```

**stackTop**: Retrieves the top element without removing it.

```
// Example of stackTop operation
int topElement = stack.top(); // topElement = 20, Stack: [10, 20]
```

**IsEmpty**: Checks if the stack is empty.

```
// Example of IsEmpty operation
if (stack.empty()) {
    cout << "Stack is empty!" << endl;
} else {
    cout << "Stack is not empty!" << endl;
}
```

**IsFull**: Checks if the stack is full (relevant in implementations with a fixed size, such as an array-based stack).

```
// Example of IsFull operation (assuming maxSize is defined)
if (stack.size() == maxSize) {
    cout << "Stack is full!" << endl;
} else {
    cout << "Stack is not full!" << endl;
}
```

### *II. LIFO Principle and Relevance in Real-World Scenarios*

1) **LIFO (Last-In-First-Out)**

   The most recent element pushed onto the stack is the first one to be removed.

2) **Real-world relevance**
   - **Undo/Redo functionality**

     In text editors, the last action performed is the first to be undone.
   - **Browser navigation**

     The last page visited is the first one to return to when pressing "Back".
   - **Backtracking algorithms**

     When navigating paths (e.g., solving mazes), the last tried path is revisited first.

## Task 2 Application of Stack in a Real-world Scenario

### *I. Browser Back and Forward Navigation*

In the case of web browsers, there exist two stacks, namely Back Stack and Forward Stack.

How it works:

**1) Loading New Page:**

- The URL of the current page gets pushed onto the Back Stack.
- Forward Stack gets cleared.

*Example:*

Visit Page1, Page2, Page3: Back Stack = [Page1, Page2, Page3], Forward Stack = []

**2) Click "Back"**

- Top of Back Stack gets popped and gets added to the Forward Stack.
- You go to the new top of the Back Stack.

*Example:*

Press "Back" at Page3: Back Stack = [Page1, Page2], Forward Stack = [Page3]

**3) Press "Forward"**

- The top of the Forward Stack is popped and added back to the Back Stack.
- You navigate to this page.

*Example:*

Press "Forward" at Page2: Back Stack = [Page1, Page2, Page3], Forward Stack = []

### *II. Role of Stack in Browser Navigation*

The stack ensures:

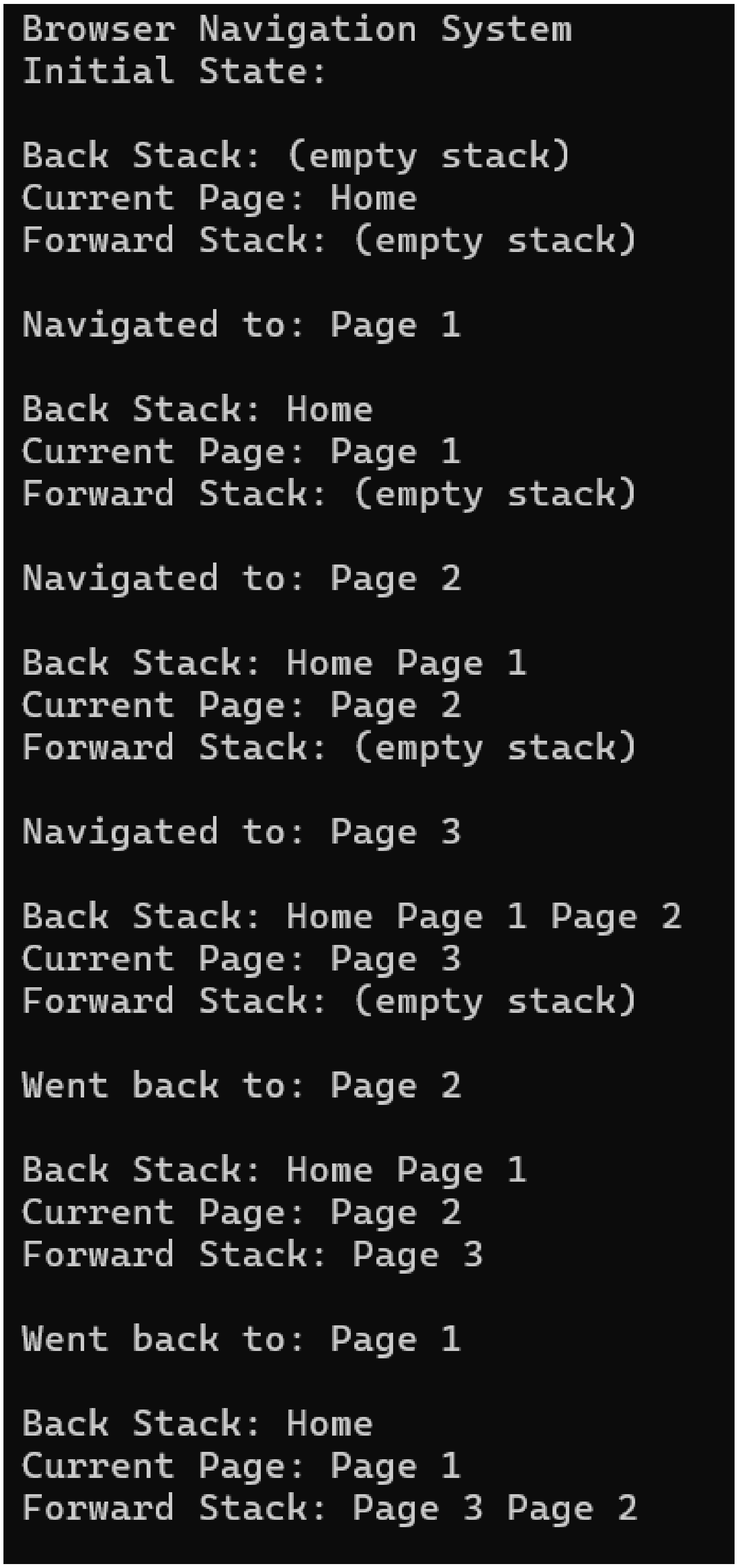
- The navigation history is managed efficiently.
- Following the LIFO principle while revisiting pages.

## Task 3 Implementation in C++

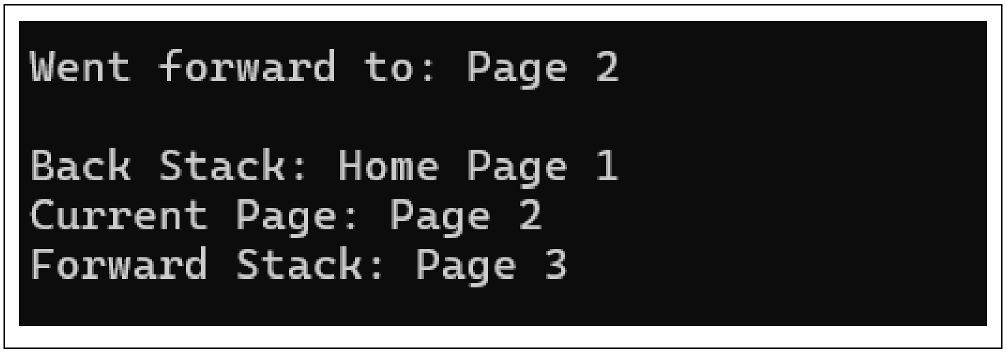
***Array-based Implementation***

```cpp
#include <iostream>
#include <string>
using namespace std;

const int MAX_SIZE = 10;

class Stack {
    private:
        int top;
        string data[MAX_SIZE];

    public:
    void createStack() {
        top=-1;
    }

    void push(string newitem) {
        if(isFull()){
            cout << "Sorry, Cannot push item. Stack is now
full!" << endl;
        }
        else {
            top = top + 1;
            data[top] = newitem;
        }
    }

    string pop() {
        if (isEmpty()){
            cout << "Sorry, Cannot pop item.Stack is empty!"
<< endl;
            return "";
        }
        else{
            string item = data[top];
            top = top - 1;
            return item;
        }
    }

    string stackTop() {
        if (isEmpty()){
            cout << "Sorry, stack is empty!" << endl;
            return "";
        }

        else
            return data[top];
    }

    bool isFull() {
        return (top == MAX_SIZE-1);
    }

    bool isEmpty() {
        return (top == -1);
    }

    void display() {
        if (isEmpty()){
            cout << "(empty stack)";
        }
        for (int i = 0; i <= top; i++){
            cout << data[i] << " ";
        }
        cout << endl;
    }
};

class Browser {
    private:
        Stack backStack;
        Stack forwardStack;
        string currentPage;

    public:
        void initialize() {
            backStack.createStack();
            forwardStack.createStack();
            currentPage = "Home";
        }

        void navigate(string page) {
            page = "Page " + page;

            if (!currentPage.empty()) {
                backStack.push(currentPage);
            }
            currentPage = page;
            while (!forwardStack.isEmpty()) {
                forwardStack.pop();
            }
            cout << "Navigated to: " << currentPage << endl;
            displayStacks();
        }

        void goBack() {
            if (backStack.isEmpty()) {
                cout << "No pages in back history!" << endl;
                return;
            }
            forwardStack.push(currentPage);
            currentPage = backStack.pop();
            cout << "Went back to: " << currentPage << endl;
            displayStacks();
        }

        void goForward() {
            if (forwardStack.isEmpty()) {
                cout << "No pages in forward history!" <<
endl;
                return;
            }
            backStack.push(currentPage);
            currentPage = forwardStack.pop();
            cout << "Went forward to: " << currentPage <<
endl;
            displayStacks();
        }

        void displayStacks() {
            cout << "\nBack Stack: ";
            backStack.display();
            cout << "Current Page: " << currentPage << endl;
            cout << "Forward Stack: ";
            forwardStack.display();
            cout << endl;
        }

};

int main (){
    Browser browser;
    browser.initialize();

    cout << "Browser Navigation System" << endl;
    cout << "Initial State:\n";
    browser.displayStacks();

    browser.navigate("1");
    browser.navigate("2");
    browser.navigate("3");

    browser.goBack();
    browser.goBack();
    browser.goForward();
    return 0;
}
```

***[Array-based Implementation] Output Result***



```
Browser Navigation System
Initial State:

Back Stack: (empty stack)
Current Page: Home
Forward Stack: (empty stack)

Navigated to: Page 1

Back Stack: Home
Current Page: Page 1
Forward Stack: (empty stack)

Navigated to: Page 2

Back Stack: Home Page 1
Current Page: Page 2
Forward Stack: (empty stack)

Navigated to: Page 3

Back Stack: Home Page 1 Page 2
Current Page: Page 3
Forward Stack: (empty stack)

Went back to: Page 2

Back Stack: Home Page 1
Current Page: Page 2
Forward Stack: Page 3

Went back to: Page 1

Back Stack: Home
Current Page: Page 1
Forward Stack: Page 3 Page 2
```

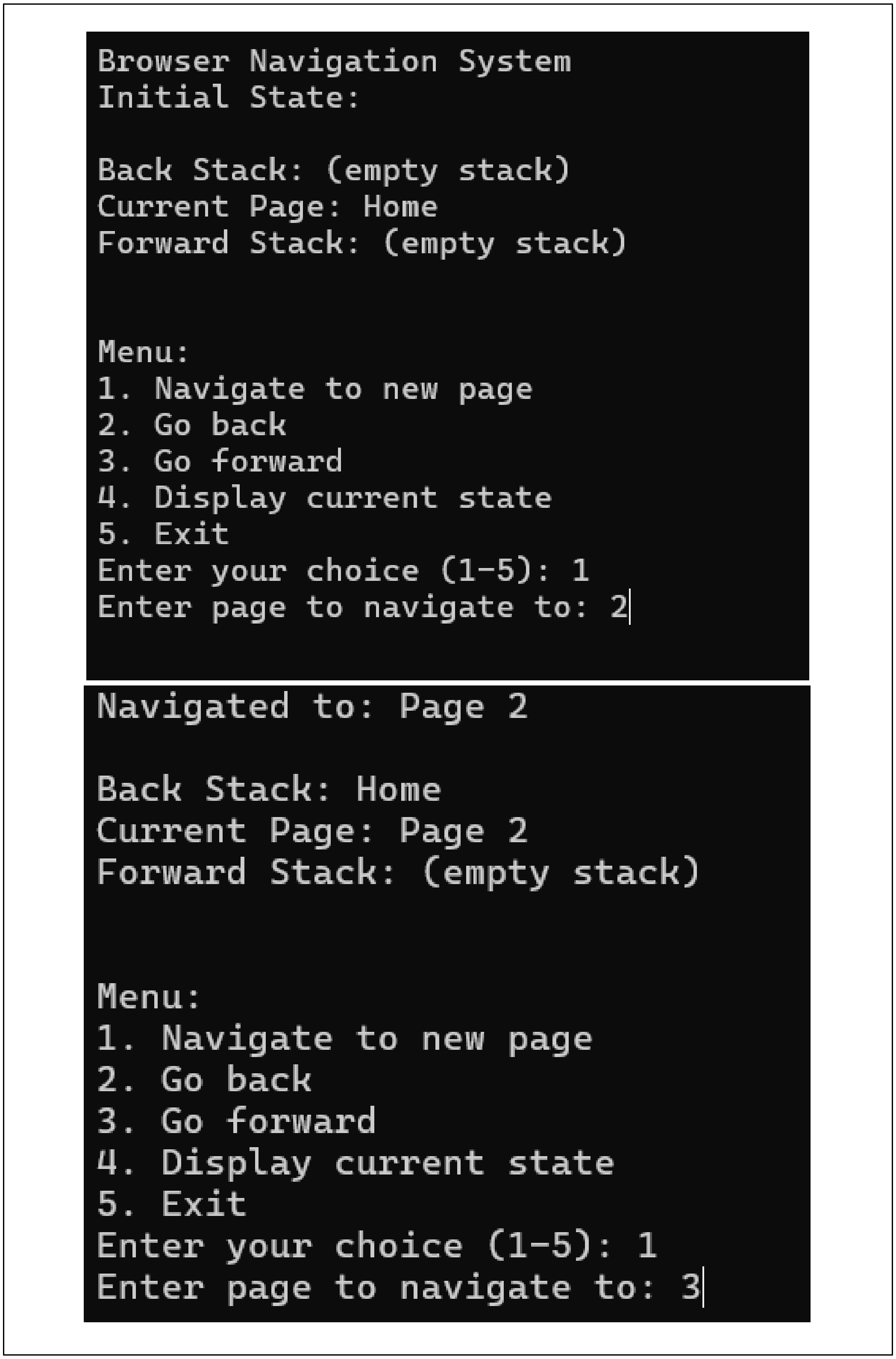


```
Went forward to: Page 2

Back Stack: Home Page 1
Current Page: Page 2
Forward Stack: Page 3
```

***[Array-based Implementation] Menu-driven***

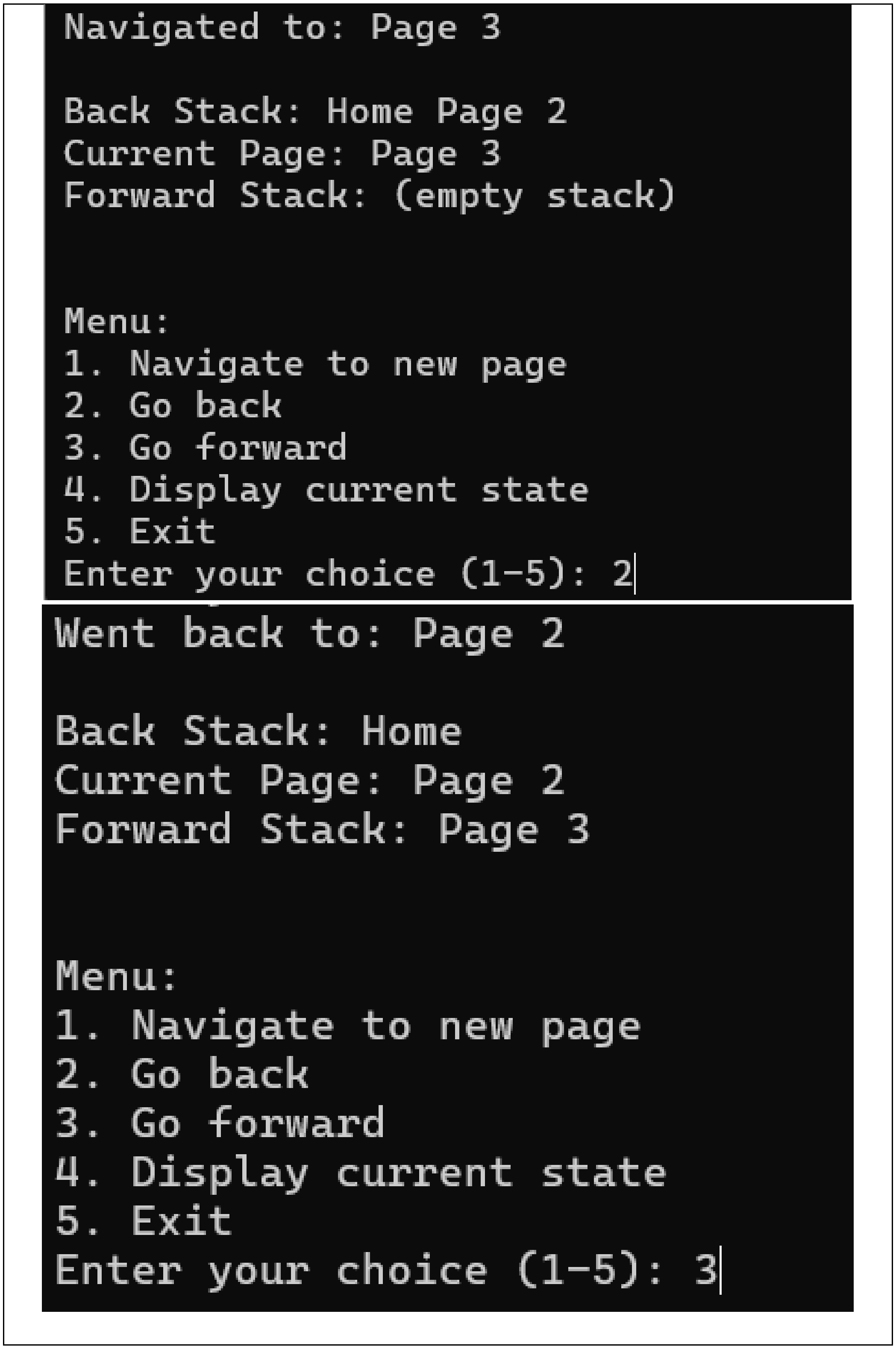
```cpp
#include <iostream>
#include <string>
using namespace std;

const int MAX_SIZE = 10;

class Stack {
    private:
        int top;
        string data[MAX_SIZE];

    public:
    void createStack() {
        top=-1;
    }

    void push(string newitem) {
        if(isFull()){
            cout << "Sorry, Cannot push item. Stack is now
full!" << endl;
        }
        else {
            top = top + 1;
            data[top] = newitem;
        }
    }

    string pop() {
        string item;
        if (isEmpty()){
            cout << "Sorry, Cannot pop item.Stack is empty!"
<< endl;
            return "";
        }
        else{
            item = data[top];
            top = top - 1;
            return item;
        }
    }

    string stackTop() {
        if (isEmpty()){
            cout << "Sorry, stack is empty!" << endl;
            return "";
        }

        else
            return data[top];
    }

    bool isFull() {
        return (top == MAX_SIZE-1);
    }

    bool isEmpty() {
        return (top == -1);
    }

    void display() {
        if (isEmpty()){
            cout << "(empty stack)" << endl;
            return;
        }
        for (int i = 0; i <= top; i++){
            cout << data[i] << " ";
        }
        cout << endl;
    }
};

class Browser {
    private:
        Stack backStack;
        Stack forwardStack;
        string currentPage;

    public:
        void initialize() {
            backStack.createStack();
            forwardStack.createStack();
            currentPage = "Home";
        }

        void navigate(string page) {
            page = "Page " + page;

            if (!currentPage.empty()) {
                backStack.push(currentPage);
            }
            currentPage = page;
            while (!forwardStack.isEmpty()) {
                forwardStack.pop();
            }
            cout << "Navigated to: " << currentPage << endl;
            displayStacks();
        }

        void goBack() {
            if (backStack.isEmpty()) {
                cout << "No pages in back history!" << endl;
                return;
            }
            forwardStack.push(currentPage);
            currentPage = backStack.pop();
            cout << "Went back to: " << currentPage << endl;
            displayStacks();
        }

        void goForward() {
            if (forwardStack.isEmpty()) {
                cout << "No pages in forward history!" <<
endl;
                return;
            }
            backStack.push(currentPage);
            currentPage = forwardStack.pop();
            cout << "Went forward to: " << currentPage <<
endl;
            displayStacks();
        }

        void displayStacks() {
            cout << "\nBack Stack: ";
            backStack.display();
            cout << "Current Page: " << currentPage << endl;
            cout << "Forward Stack: ";
            forwardStack.display();
            cout << endl;
        }

};

int main (){
    Browser browser;

    browser.initialize();

    int choice;
    string page;

    cout << "Browser Navigation System\n";
    cout << "Initial State:\n";

    browser.displayStacks();

    do {
        cout << "\nMenu:\n";
        cout << "1. Navigate to new page\n"
             << "2. Go back\n"
             << "3. Go forward\n"
             << "4. Display current state\n"
             << "5. Exit\n"
             << "Enter your choice (1-5): ";
        cin >> choice;

        switch(choice) {
            case 1:
                cout << "Enter page to navigate to: ";
                cin >> page;
                browser.navigate(page);
                break;

            case 2:
                browser.goBack();
                break;

            case 3:
                browser.goForward();
                break;

            case 4:
                cout << "Current Browser State:\n";
                browser.displayStacks();
                break;

            case 5:
                cout << "Exiting program...\n";
                break;

            default:
                cout << "Invalid choice! Please enter a 
number between 1-5.\n";
                browser.displayStacks();
        }
    }while (choice != 5);


    return 0;
}
```

***[Array-based Menu-driven Implementation] Output Result***



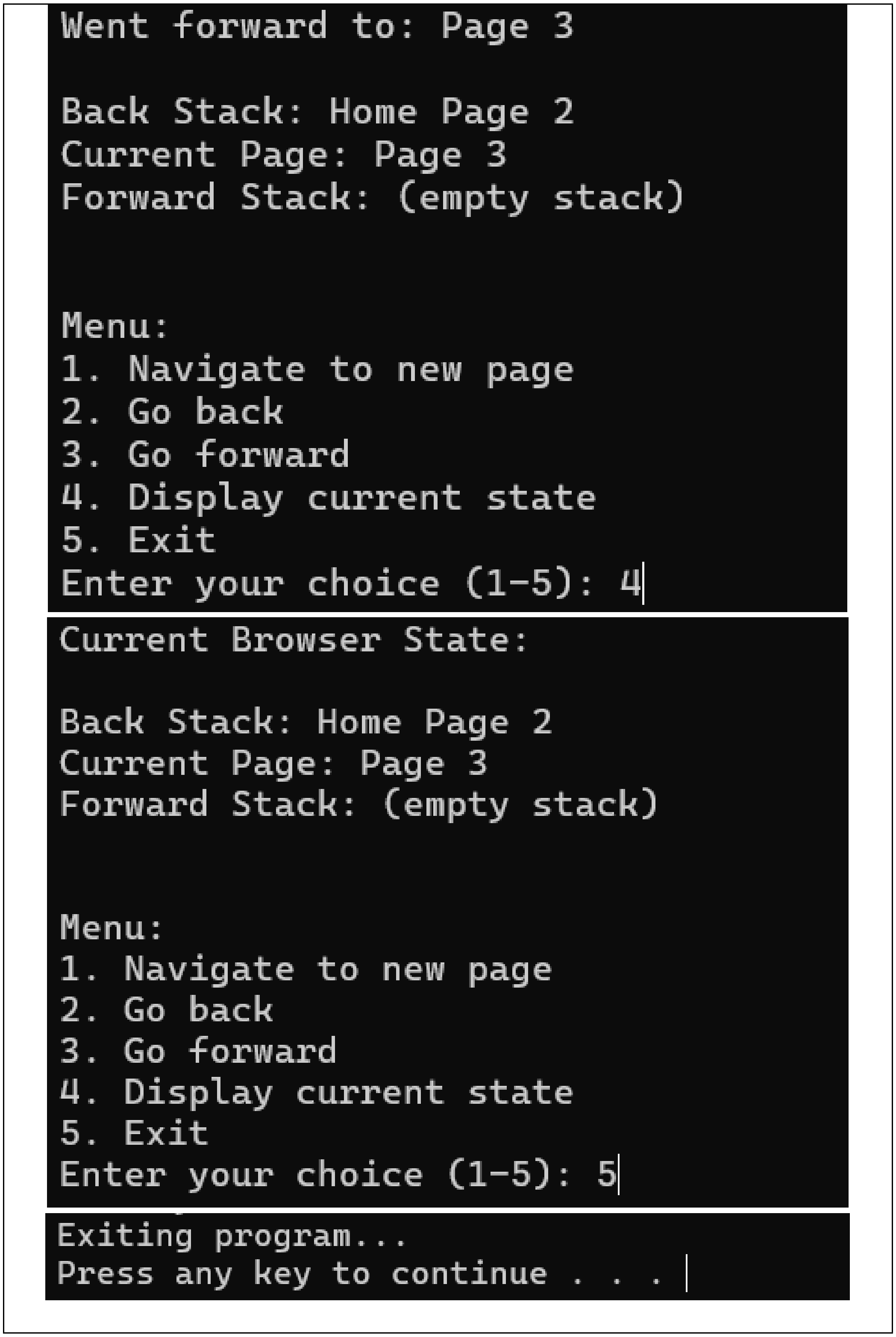
```
Browser Navigation System
Initial State:

Back Stack: (empty stack)
Current Page: Home
Forward Stack: (empty stack)


Menu:
1. Navigate to new page
2. Go back
3. Go forward
4. Display current state
5. Exit
Enter your choice (1-5): 1
Enter page to navigate to: 2
```

```
Navigated to: Page 2

Back Stack: Home
Current Page: Page 2
Forward Stack: (empty stack)


Menu:
1. Navigate to new page
2. Go back
3. Go forward
4. Display current state
5. Exit
Enter your choice (1-5): 1
Enter page to navigate to: 3
```



```
Navigated to: Page 3

Back Stack: Home Page 2
Current Page: Page 3
Forward Stack: (empty stack)


Menu:
1. Navigate to new page
2. Go back
3. Go forward
4. Display current state
5. Exit
Enter your choice (1-5): 2
```

```
Went back to: Page 2

Back Stack: Home
Current Page: Page 2
Forward Stack: Page 3


Menu:
1. Navigate to new page
2. Go back
3. Go forward
4. Display current state
5. Exit
Enter your choice (1-5): 3
```



```
Went forward to: Page 3

Back Stack: Home Page 2
Current Page: Page 3
Forward Stack: (empty stack)


Menu:
1. Navigate to new page
2. Go back
3. Go forward
4. Display current state
5. Exit
Enter your choice (1-5): 4
```

```
Current Browser State:

Back Stack: Home Page 2
Current Page: Page 3
Forward Stack: (empty stack)


Menu:
1. Navigate to new page
2. Go back
3. Go forward
4. Display current state
5. Exit
Enter your choice (1-5): 5
```

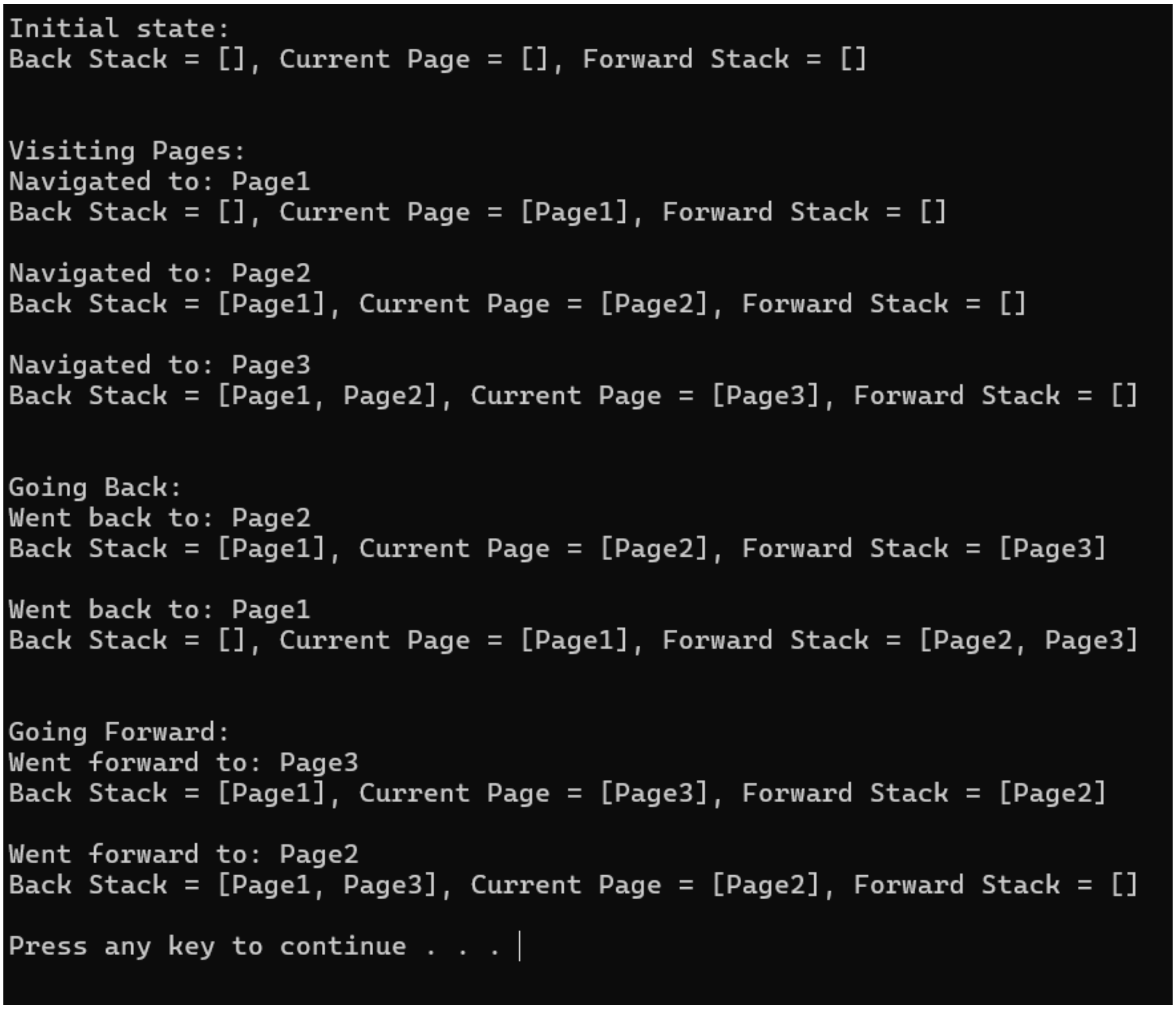
```
Exiting program...
Press any key to continue . . .
```

### ***Pointer-based Implementation***

```
#include <iostream>
#include <string>
using namespace std;

class Node{
    public:
        string data;
        Node *next;
};

class Stack{
    private:
        Node *head;

    public:
        Stack(){
            head = nullptr;
        }

        ~Stack(){
            while (!isEmpty()) {
                pop();
            }
        }

        void push(string newItem) {
            Node *newNode = new Node;
            newNode->data = newItem;
            newNode->next = head;
            head = newNode;
        }

        string pop() {
            if (isEmpty()) {
                cout << "Sorry, cannot pop item. Stack is
empty." << endl;
                return "";
            }

            Node *delNode = head;
            string popData = head->data;
            head = head->next;
            delete delNode;
            return popData;
        }

        string top() const {
            if (isEmpty()) {
                return "";
            }
            return head->data;
        }

        bool isEmpty() const {
            return head == nullptr;
        }

        void display() const {
            Stack tempStack; // Create a temporary stack to
reverse the order
            Node* current = head;

            while (current != nullptr) { // Copy items to
temp stack
                tempStack.push(current->data);
                current = current->next;
            }

            current = tempStack.head; // Reverse the order
            bool isFirst = true;
            while (current != nullptr) {
                if (!isFirst) cout << ", ";
                cout << current->data;
                current = current->next;
                isFirst = false;
            }
        }
};

class Browser {
private:
    Stack backStack;
    Stack forwardStack;
    string currentPage;

public:
    Browser() : currentPage("") {}

    void navigate(string page) {
        if (!currentPage.empty()) {
            backStack.push(currentPage);
        }
        currentPage = page;
        cout << "Navigated to: " << page << endl;
        forwardStack = Stack(); // Clear forward stack
        displayStacks();
    }

    void goBack() {
        if (backStack.isEmpty()) {
            cout << "No pages in back history!" << endl;
            return;
        }

        Stack tempForward; // Forward history

        while (!forwardStack.isEmpty()) { // Add forward
history to temp stack
            tempForward.push(forwardStack.pop());
        }

        forwardStack.push(currentPage); // Add current page
to forward stack

        while (!tempForward.isEmpty()) { // Add the rest of
forward history
            forwardStack.push(tempForward.pop());
        }

        currentPage = backStack.pop();
        cout << "Went back to: " << currentPage << endl;
        displayStacks();
    }

    void goForward() {
        if (forwardStack.isEmpty()) {
            cout << "No pages in forward history!" << endl;
            return;
        }
        backStack.push(currentPage);
        currentPage = forwardStack.pop();
        cout << "Went forward to: " << currentPage << endl;
        displayStacks();
    }

    void displayStacks() {
        cout << "Back Stack = [";
        backStack.display();
        cout << "], Current Page = [" << currentPage;
```

```cpp
        cout << "], Forward Stack = [";
        forwardStack.display();
        cout << "]" << "\n\n";
    }
};

int main() {
    Browser browser;

    cout << "Initial state:" << endl;
    browser.displayStacks();

    cout << "\nVisiting Pages:" << endl;
    browser.navigate("Page1");
    browser.navigate("Page2");
    browser.navigate("Page3");

    cout << "\nGoing Back:" << endl;
    browser.goBack();  // Back to Page2
    browser.goBack();  // Back to Page1

    cout << "\nGoing Forward:" << endl;
    browser.goForward();  // Forward to Page2
    browser.goForward();  // Forward to Page3

    return 0;
}
```

### *[Pointer-based Implementation] Output Result*

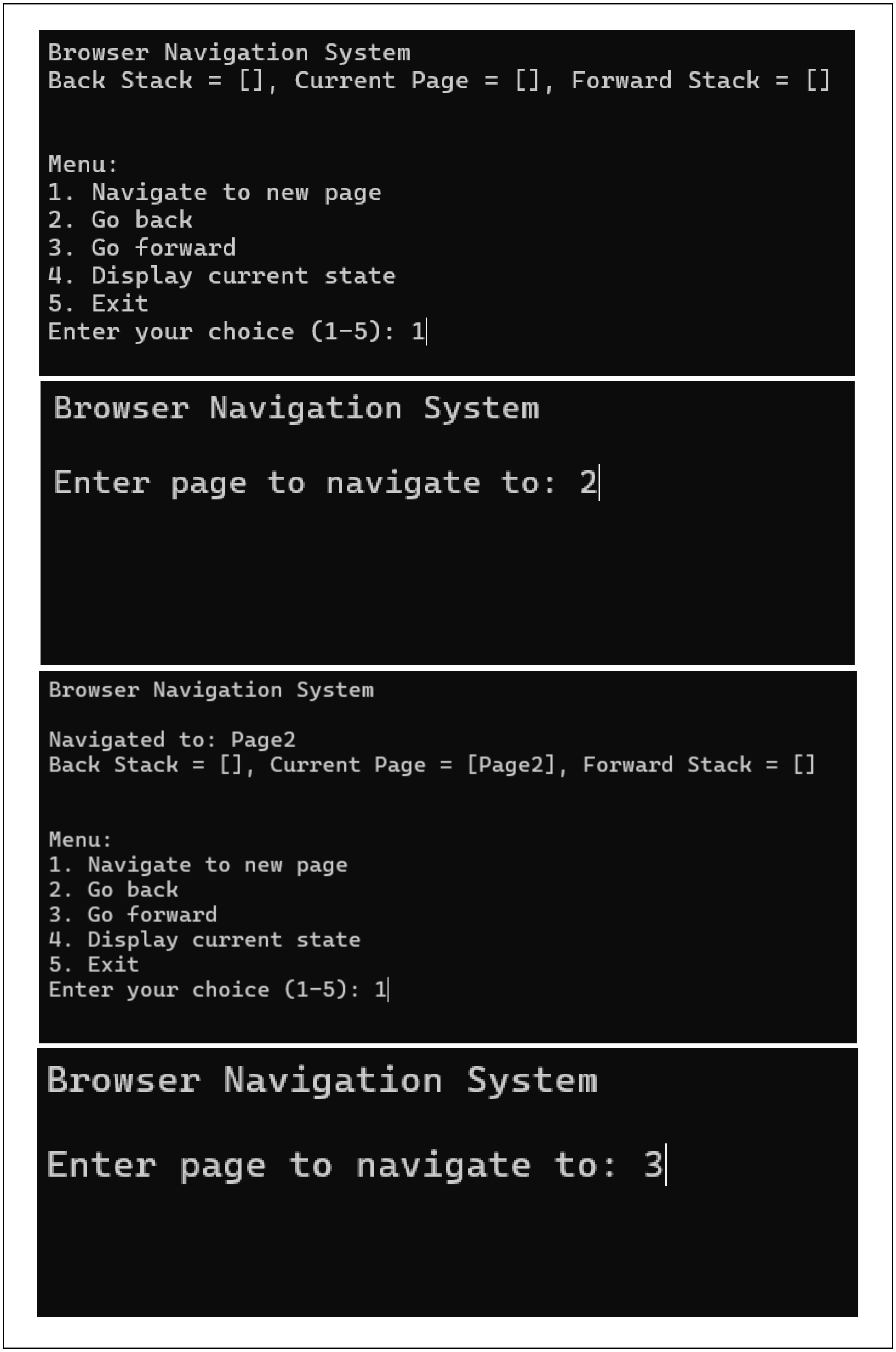


```
Initial state:
Back Stack = [], Current Page = [], Forward Stack = []


Visiting Pages:
Navigated to: Page1
Back Stack = [], Current Page = [Page1], Forward Stack = []

Navigated to: Page2
Back Stack = [Page1], Current Page = [Page2], Forward Stack = []

Navigated to: Page3
Back Stack = [Page1, Page2], Current Page = [Page3], Forward Stack = []


Going Back:
Went back to: Page2
Back Stack = [Page1], Current Page = [Page2], Forward Stack = [Page3]

Went back to: Page1
Back Stack = [], Current Page = [Page1], Forward Stack = [Page2, Page3]


Going Forward:
Went forward to: Page3
Back Stack = [Page1], Current Page = [Page3], Forward Stack = [Page2]

Went forward to: Page2
Back Stack = [Page1, Page3], Current Page = [Page2], Forward Stack = []

Press any key to continue . . .
```

### *[Pointer-based Implementation] Menu-driven*

```cpp
#include <iostream>
#include <string>
using namespace std;

class Node{
    public:
        string data;
        Node *next;
};

class Stack{
    private:
        Node *head;

    public:
        Stack(){
            head = nullptr;
        }

        ~Stack(){
            while (!isEmpty()) {
                pop();
            }
        }

        void push(string newItem) {
            Node *newNode = new Node;
            newNode->data = newItem;
            newNode->next = head;
            head = newNode;
        }

        string pop() {
            if (isEmpty()) {
                cout << "Sorry, cannot pop item. Stack is
empty." << endl;
                return "";
            }

            Node *delNode = head;
            string popData = head->data;
            head = head->next;
            delete delNode;
            return popData;
        }

        string top() const {
            if (isEmpty()) {
                return "";
            }
            return head->data;
        }

        bool isEmpty() const {
            return head == nullptr;
        }

        void display() const {
            Stack tempStack; // Create a temporary stack to
reverse the order
            Node* current = head;

            while (current != nullptr) { // Copy items to
temp stack
                tempStack.push(current->data);
                current = current->next;
            }

            current = tempStack.head; // Reverse the order
            bool isFirst = true;
            while (current != nullptr) {
                if (!isFirst) cout << ", ";
                cout << current->data;
                current = current->next;
                isFirst = false;
            }
        }
};

class Browser {
private:
    Stack backStack;
    Stack forwardStack;
    string currentPage;

public:
    Browser() : currentPage("") {}

    void navigate(string page) {
        if (!currentPage.empty()) {
            backStack.push(currentPage);
        }
        currentPage = page;
        cout << "Navigated to: " << page << endl;
        forwardStack = Stack(); // Clear forward stack
        displayStacks();
    }

    void goBack() {
        if (backStack.isEmpty()) {
            cout << "No pages in back history!" << endl;
            return;
        }

        Stack tempForward; // Forward history

        while (!forwardStack.isEmpty()) { // Add forward
history to temp stack
            tempForward.push(forwardStack.pop());
        }

        forwardStack.push(currentPage); // Add current page
to forward stack

        while (!tempForward.isEmpty()) { // Add the rest of
forward history
            forwardStack.push(tempForward.pop());
        }

        currentPage = backStack.pop();
        cout << "Went back to: " << currentPage << endl;
        displayStacks();
    }

    void goForward() {
        if (forwardStack.isEmpty()) {
            cout << "No pages in forward history!" << endl;
            return;
        }
        backStack.push(currentPage);
        currentPage = forwardStack.pop();
        cout << "Went forward to: " << currentPage << endl;
        displayStacks();
    }

    void displayStacks() {
        cout << "Back Stack = [";
        backStack.display();
        cout << "], Current Page = [" << currentPage;
        cout << "], Forward Stack = [";
        forwardStack.display();
        cout << "]" << "\n\n";
    }
};

int main() {
    Browser browser;
    int choice;
    string page;
    int pageNumber;

    system("cls");
    cout << "Browser Navigation System\n";
    browser.displayStacks();

    do {
        cout << "\nMenu:\n";
        cout << "1. Navigate to new page\n";
        cout << "2. Go back\n";
        cout << "3. Go forward\n";
        cout << "4. Display current state\n";
        cout << "5. Exit\n";
        cout << "Enter your choice (1-5): ";
        cin >> choice;

        system("cls");
        cout << "Browser Navigation System\n\n";

        switch (choice) {
            case 1:
                cout << "Enter page to navigate to: ";
                cin >> pageNumber;
                system("cls");
                cout << "Browser Navigation System\n\n";
                page = "Page" + to_string(pageNumber);
                browser.navigate(page);
                break;

            case 2:
                browser.goBack();
                break;

            case 3:
                browser.goForward();
                break;

            case 4:
                cout << "Current Browser State:\n";
                browser.displayStacks();
                break;

            case 5:
                cout << "Exiting program...\n";
                break;

            default:
                cout << "Invalid choice! Please enter a 
number between 1-5.\n";
                browser.displayStacks();
        }
    } while (choice != 5);

    return 0;
}
```

*[Pointer-based Menu-driven Implementation] Output result*

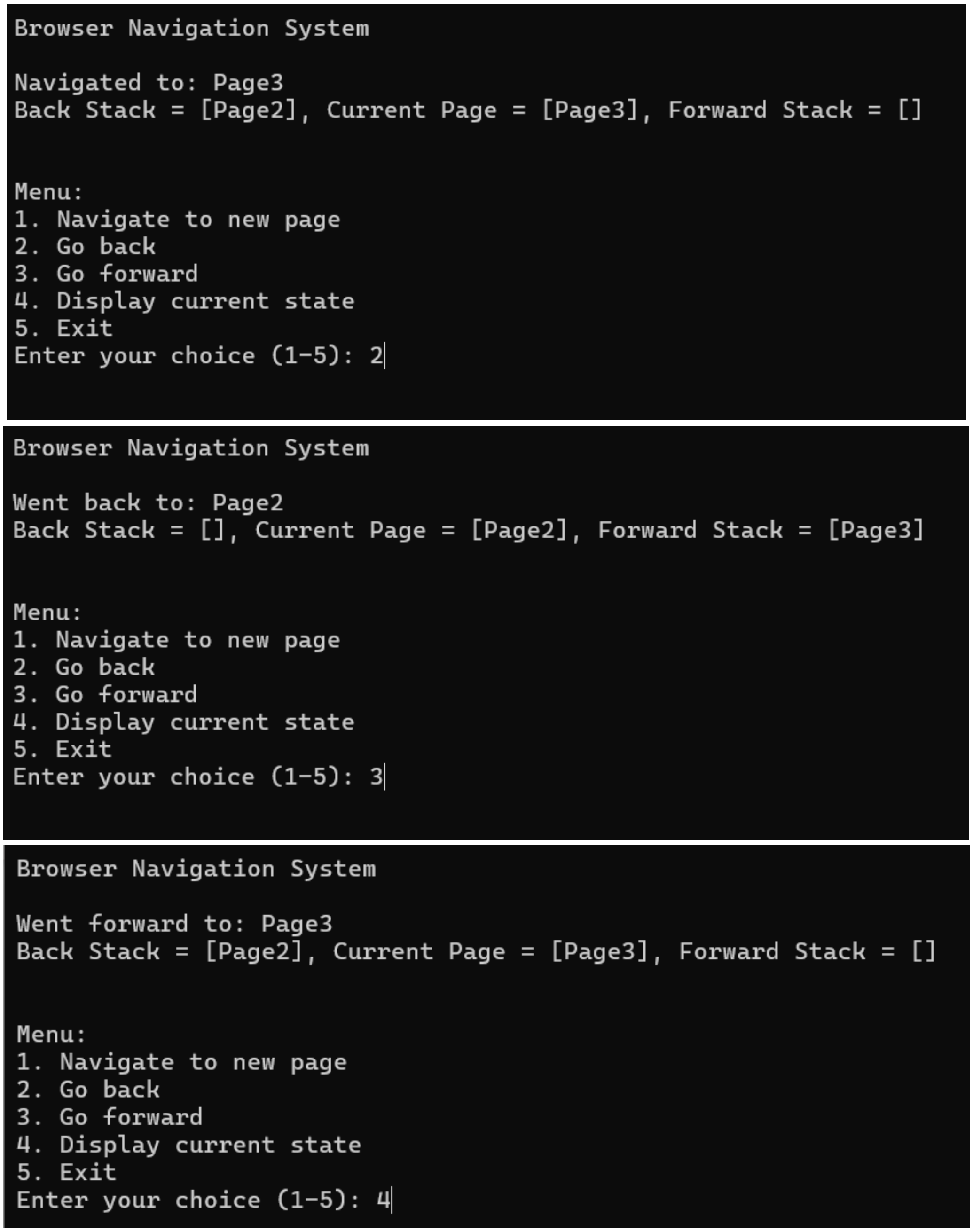


```
Browser Navigation System
Back Stack = [], Current Page = [], Forward Stack = []


Menu:
1. Navigate to new page
2. Go back
3. Go forward
4. Display current state
5. Exit
Enter your choice (1-5): 1
```

```
Browser Navigation System

Enter page to navigate to: 2
```

```
Browser Navigation System

Navigated to: Page2
Back Stack = [], Current Page = [Page2], Forward Stack = []


Menu:
1. Navigate to new page
2. Go back
3. Go forward
4. Display current state
5. Exit
Enter your choice (1-5): 1
```

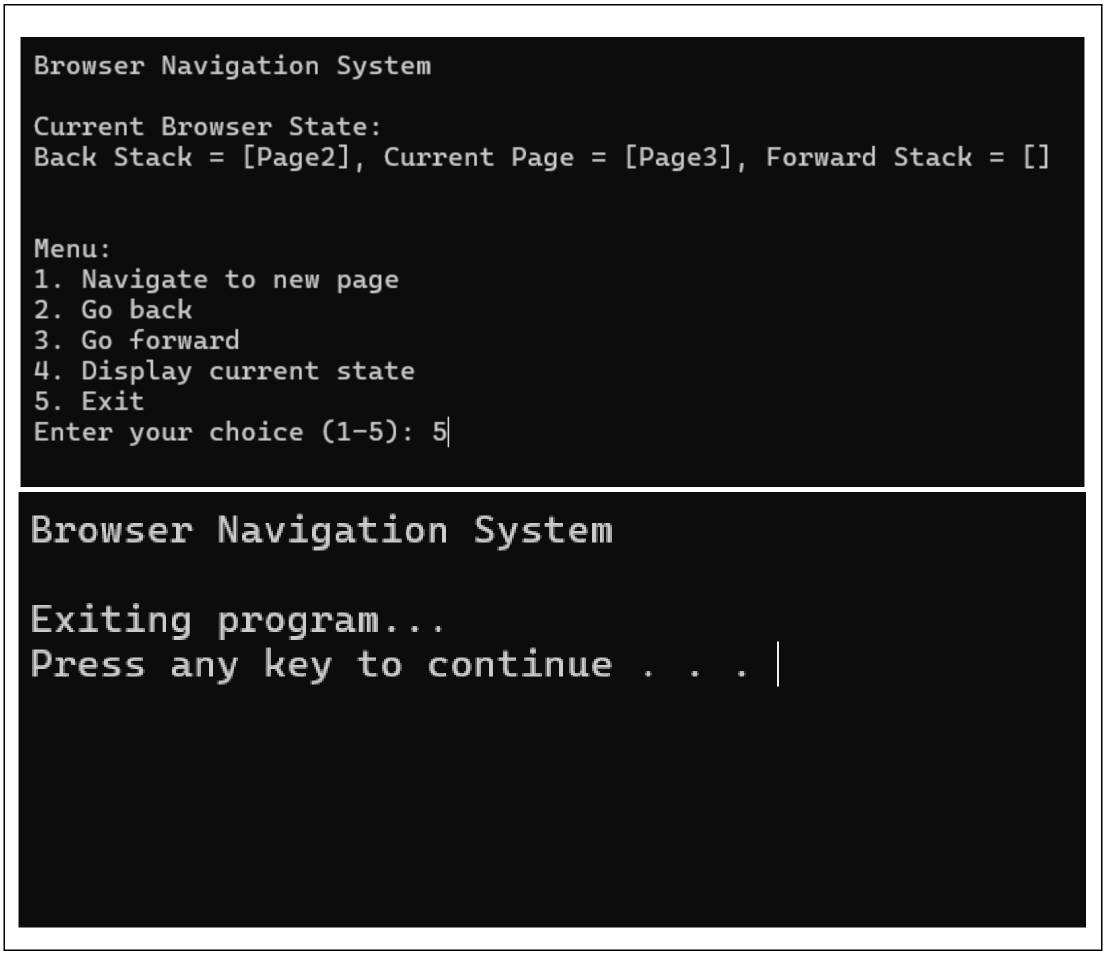
```
Browser Navigation System

Enter page to navigate to: 3
```



```
Browser Navigation System

Navigated to: Page3
Back Stack = [Page2], Current Page = [Page3], Forward Stack = []


Menu:
1. Navigate to new page
2. Go back
3. Go forward
4. Display current state
5. Exit
Enter your choice (1-5): 2
```

```
Browser Navigation System

Went back to: Page2
Back Stack = [], Current Page = [Page2], Forward Stack = [Page3]


Menu:
1. Navigate to new page
2. Go back
3. Go forward
4. Display current state
5. Exit
Enter your choice (1-5): 3
```

```
Browser Navigation System

Went forward to: Page3
Back Stack = [Page2], Current Page = [Page3], Forward Stack = []


Menu:
1. Navigate to new page
2. Go back
3. Go forward
4. Display current state
5. Exit
Enter your choice (1-5): 4
```



```
Browser Navigation System

Current Browser State:
Back Stack = [Page2], Current Page = [Page3], Forward Stack = []


Menu:
1. Navigate to new page
2. Go back
3. Go forward
4. Display current state
5. Exit
Enter your choice (1-5): 5
```

```
Browser Navigation System

Exiting program...
Press any key to continue . . .
```

## Task 4 Comparative Analysis/ Findings

It discusses how a stack data structure is implemented with the use of two techniques – the array-based data structure technique and the pointer based linked list technique. Memory utilization, ease of implementation and scalability are the factors on which the development is evaluated.

### *I. Memory Usage*

Array-based stacks used an array of finite size allocated at build time. This leads to inefficient memory utilization, as arrays may be too large and result in wasted space or too small and cause overflow if the stack depth exceeds the array's depth. Pointer-based stacks allocate memory dynamically by adding nodes on the required basis. Such a memory allocation method allocates memory only when items are appended and releases them upon withdrawal. Thus, pointer-based implementations are memory efficient and flexible.

### *II. Ease of Implementation*

The implementation of array-based systems is generally easier. Basic array use combined with a variable to maintain the top of the stack makes very easy push and pop operations. In contrast, pointer-based implementations are much more complex as they require a higher understanding of pointers and dynamic memory management as well as node topologies and are thus much more difficult to implement.

### *III. Scalability*

An array-based stack is not scalable because it is of fixed size at the point of its declaration. A stack defined this way cannot dynamically change with the number of entries or there will be either underflow caused by lack of items or overflow due to wastages in memory. Conversely, pointer-based stacks are an absolute picture of scalability because they can expand and shrink dynamically based on their needs. Hence, they can vary in size without any constraints. New components may be added if there is memory available in the system.

In short, solution stacks based on an array are easier to implement but are less memory efficient and non-scalable. Pointer-based stacks are harder to develop; however, they provide more efficient memory and scalability improvements. The choice between the two is dependent on the real application's needs: stack solutions that are based on arrays are suitable for small static stacks, whereas tree-based solutions are better for large dynamic stacks.

**--- END OF DOCUMENTATION ---**