

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	3
I.	Basic Operations of a Stack	3
II.	LIFO Principle and Relevance in Real-World Scenarios	4
Task	2 Application of Stack in a Real-world Scenario	4
I.	Browser Back and Forward Navigation	4
II.	Role of Stack in Browser Navigation	5
Task	3 Implementation in C++	5
Arr	ay-based Implementation	5
[Array-based Implementation] Output Result	9
[Array-based Implementation] Menu-driven	10
[Array-based Menu-driven Implementation] Output Result	15
Ро	inter-based Implementation	18
[Pointer-based Implementation] Output Result	22
[Pointer-based Implementation] Menu-driven	23
[Pointer-based Menu-driven Implementation] Output result	28
Task	4 Comparative Analysis/ Findings	30
I.	Memory Usage	30
II.	Ease of Implementation	31
III.	Scalability	31

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;
}</pre>
```

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;
}</pre>
```

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

```
#include <iostream>
 2
  #include <string>
 3 using namespace std;
 4
 5
  const int MAX SIZE = 10;
 6
 7
   class Stack {
 8
       private:
 9
            int top;
10
            string data[MAX SIZE];
11
12
       public:
13
       void createStack() {
14
            top=-1;
15
16
17
       void push(string newitem) {
18
            if(isFull()){
19
                cout << "Sorry, Cannot push item. Stack is now</pre>
20
   full!" << endl;
21
            }
22
            else {
23
                top = top + 1;
24
                data[top] = newitem;
25
```

```
2.6
27
28
        string pop() {
29
            if (isEmpty()) {
30
                 cout << "Sorry, Cannot pop item.Stack is empty!"</pre>
31
   << endl;
32
                 return "";
33
34
            else{
35
                 string item = data[top];
36
                 top = top - 1;
37
                 return item;
38
            }
39
        }
40
41
        string stackTop() {
42
            if (isEmpty()) {
43
                 cout << "Sorry, stack is empty!" << endl;</pre>
44
                 return "";
45
            }
46
47
            else
48
                 return data[top];
49
        }
50
51
       bool isFull() {
52
            return (top == MAX SIZE-1);
53
        }
54
55
       bool isEmpty() {
56
            return (top == -1);
57
58
59
        void display() {
60
            if (isEmpty()) {
61
                 cout << "(empty stack)";</pre>
62
63
            for (int i = 0; i <= top; i++) {</pre>
                 cout << data[i] << " ";</pre>
64
65
            cout << endl;</pre>
66
67
        }
68 };
69
70 class Browser {
71
        private:
72
            Stack backStack;
```

```
Stack forwardStack;
 74
             string currentPage;
 75
 76
        public:
 77
            void initialize() {
 78
                 backStack.createStack();
 79
                 forwardStack.createStack();
 80
                 currentPage = "Home";
 81
             }
 82
 83
            void navigate(string page) {
 84
                 page = "Page " + page;
 85
 86
                 if (!currentPage.empty()) {
 87
                     backStack.push(currentPage);
 88
                 }
 89
                 currentPage = page;
 90
                 while (!forwardStack.isEmpty()) {
 91
                     forwardStack.pop();
 92
 93
                 cout << "Navigated to: " << currentPage << endl;</pre>
 94
                 displayStacks();
 95
             }
 96
 97
            void goBack() {
 98
                 if (backStack.isEmpty()) {
 99
                     cout << "No pages in back history!" << endl;</pre>
100
                     return;
101
102
                 forwardStack.push(currentPage);
103
                 currentPage = backStack.pop();
104
                 cout << "Went back to: " << currentPage << endl;</pre>
105
                 displayStacks();
106
             }
107
108
            void goForward() {
109
                 if (forwardStack.isEmpty()) {
110
                     cout << "No pages in forward history!" <<</pre>
111 endl;
112
                     return;
113
114
                 backStack.push(currentPage);
115
                 currentPage = forwardStack.pop();
116
                 cout << "Went forward to: " << currentPage <<</pre>
117 endl;
118
                 displayStacks();
119
```

```
120
121
             void displayStacks() {
122
                 cout << "\nBack Stack: ";</pre>
123
                 backStack.display();
                 cout << "Current Page: " << currentPage << endl;</pre>
124
125
                 cout << "Forward Stack: ";</pre>
126
                 forwardStack.display();
127
                 cout << endl;</pre>
128
             }
129
130 };
131
132 int main () {
133
        Browser browser;
134
        browser.initialize();
135
136
        cout << "Browser Navigation System" << endl;</pre>
137
        cout << "Initial State:\n";</pre>
138
        browser.displayStacks();
139
        browser.navigate("1");
140
141
        browser.navigate("2");
142
        browser.navigate("3");
143
144
        browser.goBack();
145
        browser.goBack();
146
        browser.goForward();
147
        return 0;
148
```

[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

```
#include <iostream>
 2 #include <string>
 3 using namespace std;
  const int MAX SIZE = 10;
  class Stack {
 8
       private:
 9
           int top;
10
           string data[MAX SIZE];
11
12
       public:
13
       void createStack() {
14
           top=-1;
15
16
17
       void push(string newitem) {
18
           if(isFull()){
19
                cout << "Sorry, Cannot push item. Stack is now</pre>
20 full!" << endl;
21
           }
22
           else {
23
                top = top + 1;
24
                data[top] = newitem;
25
26
       }
27
28
       string pop() {
29
           string item;
30
           if (isEmpty()) {
31
                cout << "Sorry, Cannot pop item.Stack is empty!"</pre>
32 << endl;
33
                return "";
34
35
           else{
```

```
36
                item = data[top];
37
                top = top - 1;
38
                return item;
39
40
       }
41
42
       string stackTop() {
43
            if (isEmpty()) {
44
                cout << "Sorry, stack is empty!" << endl;</pre>
45
                return "";
46
            }
47
48
            else
49
                return data[top];
50
       }
51
52
       bool isFull() {
53
            return (top == MAX SIZE-1);
54
       }
55
56
       bool isEmpty() {
57
            return (top == -1);
58
       }
59
60
       void display() {
61
            if (isEmpty()) {
62
                cout << "(empty stack)" << endl;</pre>
63
                return;
64
65
            for (int i = 0; i <= top; i++) {
66
                cout << data[i] << " ";
67
68
            cout << endl;
69
       }
70 };
71
72 class Browser {
73
       private:
74
            Stack backStack:
75
           Stack forwardStack;
76
            string currentPage;
77
78
       public:
79
            void initialize() {
80
                backStack.createStack();
81
                forwardStack.createStack();
82
                currentPage = "Home";
```

```
83
 84
 85
            void navigate(string page) {
 86
                 page = "Page " + page;
 87
 88
                 if (!currentPage.empty()) {
 89
                     backStack.push(currentPage);
 90
 91
                 currentPage = page;
 92
                 while (!forwardStack.isEmpty()) {
 93
                      forwardStack.pop();
 94
 95
                 cout << "Navigated to: " << currentPage << endl;</pre>
 96
                 displayStacks();
 97
 98
 99
            void goBack() {
100
                 if (backStack.isEmpty()) {
101
                     cout << "No pages in back history!" << endl;</pre>
102
                     return:
103
104
                 forwardStack.push (currentPage);
105
                 currentPage = backStack.pop();
106
                 cout << "Went back to: " << currentPage << endl;</pre>
107
                 displayStacks();
108
             }
109
110
            void goForward() {
111
                 if (forwardStack.isEmpty()) {
112
                     cout << "No pages in forward history!" <<</pre>
113 endl;
114
                     return;
115
                 }
116
                 backStack.push(currentPage);
117
                 currentPage = forwardStack.pop();
118
                 cout << "Went forward to: " << currentPage <<</pre>
119 endl;
120
                 displayStacks();
121
122
123
            void displayStacks() {
124
                 cout << "\nBack Stack: ";</pre>
125
                 backStack.display();
126
                 cout << "Current Page: " << currentPage << endl;</pre>
127
                 cout << "Forward Stack: ";</pre>
128
                 forwardStack.display();
129
                 cout << endl;
```

```
130
131
132 };
133
134 int main () {
135
        Browser browser;
136
137
        browser.initialize();
138
139
        int choice;
140
        string page;
141
142
        cout << "Browser Navigation System\n";</pre>
143
        cout << "Initial State:\n";</pre>
144
145
        browser.displayStacks();
146
147
        do {
148
             cout << "\nMenu:\n";</pre>
149
             cout << "1. Navigate to new page\n"</pre>
150
                   << "2. Go back\n"
151
                   << "3. Go forward\n"
                  << "4. Display current state\n"
152
153
                   << "5. Exit\n"
154
                   << "Enter your choice (1-5): ";
155
             cin >> choice;
156
157
             switch(choice) {
158
                 case 1:
159
                      cout << "Enter page to navigate to: ";</pre>
160
                      cin >> page;
161
                      browser.navigate(page);
162
                      break;
163
164
                 case 2:
165
                      browser.goBack();
166
                      break;
167
168
                 case 3:
169
                      browser.goForward();
170
                      break;
171
172
                 case 4:
173
                      cout << "Current Browser State:\n";</pre>
174
                      browser.displayStacks();
175
                      break;
176
```

```
177
                 case 5:
                     cout << "Exiting program...\n";</pre>
178
179
                     break;
180
181
                 default:
182
                     cout << "Invalid choice! Please enter a</pre>
183 number between 1-5.\n";
184
                     browser.displayStacks();
185
186
        }while (choice != 5);
187
188
189
        return 0;
190 }
```

```
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
Go forward
4. Display current state
5. Exit
Enter your choice (1-5): 5
Exiting program...
Press any key to continue . . .
```

Pointer-based Implementation

```
1 #include <iostream>
 2 #include <string>
 3 using namespace std;
 5 class Node {
 6
       public:
 7
           string data;
 8
           Node *next;
 9
  };
10
11 class Stack{
12
       private:
13
           Node *head;
14
15
       public:
16
           Stack(){
17
                head = nullptr;
18
19
20
           ~Stack(){
21
                while (!isEmpty()) {
22
                    pop();
23
                }
24
            }
25
26
           void push(string newItem) {
27
                Node *newNode = new Node;
28
                newNode->data = newItem;
29
                newNode->next = head;
30
                head = newNode;
31
            }
32
33
            string pop() {
34
                if (isEmpty()) {
35
                    cout << "Sorry, cannot pop item. Stack is</pre>
36 empty." << endl;
37
                    return "";
38
                }
39
40
                Node *delNode = head;
41
                string popData = head->data;
42
                head = head->next;
43
                delete delNode;
44
                return popData;
45
```

```
46
47
           string top() const {
48
                if (isEmpty()) {
49
                    return "";
50
51
                return head->data;
52
            }
53
54
           bool isEmpty() const {
55
                return head == nullptr;
56
           }
57
58
           void display() const {
59
                Stack tempStack; // Create a temporary stack to
60 reverse the order
61
                Node* current = head;
62
63
                while (current != nullptr) { // Copy items to
64
   temp stack
65
                    tempStack.push(current->data);
66
                    current = current->next;
67
                }
68
69
                current = tempStack.head; // Reverse the order
70
                bool isFirst = true;
71
                while (current != nullptr) {
72
                    if (!isFirst) cout << ", ";</pre>
73
                    cout << current->data;
74
                    current = current->next;
75
                    isFirst = false;
76
77
78 };
79
80 class Browser {
81 private:
82
       Stack backStack;
83
       Stack forwardStack;
84
       string currentPage;
85
86 public:
87
       Browser() : currentPage("") {}
88
89
       void navigate(string page) {
90
           if (!currentPage.empty()) {
91
                backStack.push(currentPage);
92
```

```
93
             currentPage = page;
 94
             cout << "Navigated to: " << page << endl;</pre>
 95
             forwardStack = Stack(); // Clear forward stack
 96
             displayStacks();
 97
        }
 98
 99
        void goBack() {
100
             if (backStack.isEmpty()) {
101
                 cout << "No pages in back history!" << endl;</pre>
102
                 return;
103
             }
104
105
             Stack tempForward; // Forward history
106
107
             while (!forwardStack.isEmpty()) { // Add forward
108 history to temp stack
109
                 tempForward.push(forwardStack.pop());
110
111
112
             forwardStack.push(currentPage); // Add current page
113 to forward stack
114
115
             while (!tempForward.isEmpty()) { // Add the rest of
116 forward history
117
                 forwardStack.push(tempForward.pop());
118
119
120
             currentPage = backStack.pop();
121
             cout << "Went back to: " << currentPage << endl;</pre>
122
             displayStacks();
123
        }
124
125
        void goForward() {
126
             if (forwardStack.isEmpty()) {
127
                 cout << "No pages in forward history!" << endl;</pre>
128
                 return;
129
130
             backStack.push(currentPage);
131
             currentPage = forwardStack.pop();
132
             cout << "Went forward to: " << currentPage << endl;</pre>
133
             displayStacks();
134
        }
135
136
        void displayStacks() {
137
             cout << "Back Stack = [";</pre>
138
             backStack.display();
139
             cout << "], Current Page = [" << currentPage;</pre>
```

```
140
            cout << "], Forward Stack = [";</pre>
141
            forwardStack.display();
142
            cout << "]" << "\n\n";
143
144 };
145
146 int main() {
147
        Browser browser;
148
149
        cout << "Initial state:" << endl;</pre>
150
        browser.displayStacks();
151
152
        cout << "\nVisiting Pages:" << endl;</pre>
153
        browser.navigate("Page1");
154
        browser.navigate("Page2");
155
        browser.navigate("Page3");
156
        cout << "\nGoing Back:" << endl;</pre>
157
158
        browser.goBack(); // Back to Page2
159
        browser.goBack(); // Back to Page1
160
161
        cout << "\nGoing Forward:" << endl;</pre>
162
        browser.goForward(); // Forward to Page2
163
        browser.goForward(); // Forward to Page3
164
165
        return 0;
166
```

[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

```
1 #include <iostream>
 2 #include <string>
 3 using namespace std;
 4
 5 class Node {
 6
       public:
 7
           string data;
 8
           Node *next;
 9
  };
10
11 class Stack{
12
       private:
13
           Node *head;
14
15
       public:
16
           Stack(){
17
                head = nullptr;
18
19
20
           ~Stack(){
21
                while (!isEmpty()) {
22
                    pop();
23
                }
24
            }
25
26
           void push(string newItem) {
27
                Node *newNode = new Node;
28
                newNode->data = newItem;
29
                newNode->next = head;
30
                head = newNode;
31
            }
32
33
            string pop() {
34
                if (isEmpty()) {
35
                    cout << "Sorry, cannot pop item. Stack is</pre>
36 empty." << endl;
37
                    return "";
38
                }
39
40
                Node *delNode = head;
41
                string popData = head->data;
42
                head = head->next;
43
                delete delNode;
44
                return popData;
45
```

```
46
47
           string top() const {
48
                if (isEmpty()) {
49
                    return "";
50
51
                return head->data;
52
            }
53
54
           bool isEmpty() const {
55
                return head == nullptr;
56
           }
57
58
           void display() const {
59
                Stack tempStack; // Create a temporary stack to
60 reverse the order
61
                Node* current = head;
62
63
                while (current != nullptr) { // Copy items to
64 temp stack
65
                    tempStack.push(current->data);
66
                    current = current->next;
67
                }
68
69
                current = tempStack.head; // Reverse the order
70
                bool isFirst = true;
71
                while (current != nullptr) {
72
                    if (!isFirst) cout << ", ";</pre>
73
                    cout << current->data;
74
                    current = current->next;
75
                    isFirst = false;
76
77
78 };
79
80 class Browser {
81 private:
82
       Stack backStack;
83
       Stack forwardStack;
84
       string currentPage;
85
86 public:
87
       Browser() : currentPage("") {}
88
89
       void navigate(string page) {
90
           if (!currentPage.empty()) {
91
                backStack.push(currentPage);
92
```

```
93
             currentPage = page;
 94
             cout << "Navigated to: " << page << endl;</pre>
 95
             forwardStack = Stack(); // Clear forward stack
 96
             displayStacks();
 97
        }
 98
 99
        void goBack() {
100
             if (backStack.isEmpty()) {
101
                 cout << "No pages in back history!" << endl;</pre>
102
                 return;
103
             }
104
105
             Stack tempForward; // Forward history
106
107
            while (!forwardStack.isEmpty()) { // Add forward
108 history to temp stack
109
                 tempForward.push(forwardStack.pop());
110
111
112
             forwardStack.push(currentPage); // Add current page
113 to forward stack
114
115
            while (!tempForward.isEmpty()) { // Add the rest of
116 forward history
117
                 forwardStack.push(tempForward.pop());
118
119
120
            currentPage = backStack.pop();
121
             cout << "Went back to: " << currentPage << endl;</pre>
122
            displayStacks();
123
        }
124
125
        void goForward() {
126
             if (forwardStack.isEmpty()) {
127
                 cout << "No pages in forward history!" << endl;</pre>
128
                 return;
129
130
            backStack.push(currentPage);
131
             currentPage = forwardStack.pop();
132
             cout << "Went forward to: " << currentPage << endl;</pre>
133
            displayStacks();
134
        }
135
136
        void displayStacks() {
137
             cout << "Back Stack = [";</pre>
138
            backStack.display();
139
             cout << "], Current Page = [" << currentPage;</pre>
```

```
140
             cout << "], Forward Stack = [";</pre>
141
             forwardStack.display();
142
             cout << "]" << "\n\n";
143
144 };
145
146 int main() {
147
         Browser browser;
148
         int choice;
149
         string page;
150
         int pageNumber;
151
152
         system("cls");
153
         cout << "Browser Navigation System\n";</pre>
154
         browser.displayStacks();
155
156
         do {
157
             cout << "\nMenu:\n";</pre>
158
             cout << "1. Navigate to new page\n";</pre>
159
             cout << "2. Go back\n";</pre>
160
             cout << "3. Go forward\n";</pre>
161
             cout << "4. Display current state\n";</pre>
162
             cout << "5. Exit\n";</pre>
163
             cout << "Enter your choice (1-5): ";</pre>
164
             cin >> choice;
165
166
             system("cls");
167
             cout << "Browser Navigation System\n\n";</pre>
168
169
             switch (choice) {
170
                  case 1:
171
                      cout << "Enter page to navigate to: ";</pre>
172
                      cin >> pageNumber;
173
                      system("cls");
174
                      cout << "Browser Navigation System\n\n";</pre>
175
                      page = "Page" + to string(pageNumber);
176
                      browser.navigate(page);
177
                      break;
178
179
                  case 2:
180
                      browser.goBack();
181
                      break;
182
183
                  case 3:
184
                      browser.goForward();
185
                      break;
186
```

```
187
                 case 4:
188
                     cout << "Current Browser State:\n";</pre>
189
                     browser.displayStacks();
190
                     break;
191
192
                 case 5:
                     cout << "Exiting program...\n";</pre>
193
194
                     break;
195
196
                 default:
197
                     cout << "Invalid choice! Please enter a</pre>
198 number between 1-5.\n";
199
                     browser.displayStacks();
200
201
        } while (choice != 5);
202
203
        return 0;
204
```

[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.