

Course T1Y2: Advanced Algorithms

Lecturer: Bou Channa

Student's name: Chea Ilong

ID: 100022

Group: 1 SE Gen10

Lab 4: Assignment

Exercise1:

```

1 #include <iostream>
2 using namespace std;
3
4 // Class Array {
5     int *data;
6     int size;
7     int length;
8
9 public:
10     Array(int size) {
11         this->size = size;
12         this->length = 0;
13         this->data = new int[size];
14     }
15
16     ~Array() {
17         delete[] data;
18     }
19
20     void element(int data) {
21         if (length == size) {
22             newSize();
23         }
24         this->data[length] = data;
25         length++;
26     }
27
28     int getLength() const {
29         return length;
30     }
31
32     int getSize() const {
33         return size;
34     }
35
36     int getIndex(int index) const {
37         if (index >= 0 && index < length) {
38             return data[index];
39         }
40         return -1;
41     }
42
43     void push(int data) {
44         if (length == size) {
45             newSize();
46         }
47         this->data[length] = data;
48         length++;
49     }
50
51     void insertAt(int index, int value) {
52         if (index < 0 || index > length) {
53             return;
54         }
55         if (length == size) {
56             newSize();
57         }
58         for (int i = length; i > index; i--) {
59             this->data[i] = this->data[i - 1];
60         }
61         this->data[index] = value;
62         length++;
63     }
64
65     void removeAt(int index) {
66         if (index < 0 || index >= length) {
67             return;
68         }
69         for (int i = index; i < length - 1; i++) {
70             this->data[i] = this->data[i + 1];
71         }
72         length--;
73     }
74
75     void print() const {
76         for (int i = 0; i < length; ++i) {
77             cout << data[i] << " ";
78         }
79         cout << endl;
80     }
81
82 private:
83     void newSize() {
84         size = size * 2;
85         int *newData = new int[size];
86         for (int i = 0; i < length; i++) {
87             newData[i] = data[i];
88         }
89         delete[] data;
90         data = newData;
91     }
92 };
93
94 int main() {
95     Array arr(5);
96
97     arr.element(10);
98     arr.element(11);
99     arr.element(12);
100    arr.element(13);
101    arr.element(14);
102
103    arr.print();
104    cout << "Current length: " << arr.getLength() << ", Current size: " << arr.getSize() << endl;
105
106    cout << "The value at index 2: " << arr.getIndex(2) << endl;
107
108    arr.push(5);
109    cout << "After pushing value 5:" << endl;
110    arr.print();
111    cout << "Current length: " << arr.getLength() << ", Current size: " << arr.getSize() << endl;
112
113    arr.insertAt(5, 99);
114    cout << "After inserting 99 at index 5:" << endl;
115    arr.print();
116    cout << "Current length: " << arr.getLength() << ", Current size: " << arr.getSize() << endl;
117
118    arr.removeAt(5);
119    cout << "After removing value at index 5:" << endl;
120    arr.print();
121    cout << "Current length: " << arr.getLength() << ", Current size: " << arr.getSize() << endl;
122
123    return 0;
124 }
125

```

```
PS C:\Users\MSI PC\Desktop\a> cd "c:\Users\MSI PC\Desktop\a"
10 11 12 13 14
Current length: 5, Current size: 5
The value at index 2: 12
After pushing value 5:
10 11 12 13 14 5
Current length: 6, Current size: 10
After inserting 99 at index 5:
10 11 12 13 14 99 5
Current length: 7, Current size: 10
After removing value at index 5:
10 11 12 13 14 5
Current length: 6, Current size: 10
PS C:\Users\MSI PC\Desktop\a> █
```

Exercise2:

```

1 // Linked list structure
2 #include <iostream>
3 using namespace std;
4
5 class Node
6 {
7 public:
8     int data;
9     Node *next;
10    Node(Node*)
11    {
12        data = data;
13        next = NULL;
14    }
15 };
16
17 // Linked list
18 class LinkedList
19 {
20 public:
21     int length;
22     Node *head;
23
24 public:
25     LinkedList()
26     {
27         length = 0;
28         head = NULL;
29     }
30
31 void insertAt(int data)
32 {
33     Node *newNode = new Node(data);
34     newNode->next = head;
35     head = newNode;
36     length++;
37 }
38
39 void deleteAt(int i)
40 {
41     Node *current = head;
42     while (current != NULL)
43     {
44         Node *next = current->next;
45         current = current->next;
46         delete next;
47     }
48 }
49
50 int getLength() const
51 {
52     return length;
53 }
54
55 int getAt(int index) const
56 {
57     if (index <= 0 || index > length)
58     {
59         return -1;
60     }
61     Node *current = head;
62     for (int i = 0; i < index; i++)
63     {
64         current = current->next;
65     }
66     return current->data;
67 }
68
69 void print() const
70 {
71     Node *current = head;
72     while (current != NULL)
73     {
74         cout << current->data << " ";
75         current = current->next;
76     }
77     cout << endl;
78 }
79
80 void insertAt(int index, int value)
81 {
82     if (index <= 0 || index > length)
83     {
84         return;
85     }
86     if (index == 0)
87     {
88         insertAt(head);
89     }
90     else
91     {
92         Node *newNode = new Node(value);
93         Node *current = head;
94         for (int i = 0; i < index - 1; i++)
95         {
96             current = current->next;
97         }
98         newNode->next = current->next;
99         current->next = newNode;
100        length++;
101    }
102 }
103
104 void removeAt(int index)
105 {
106     if (index <= 0 || index > length)
107     {
108         return;
109     }
110     Node *current = head;
111     if (index == 0)
112     {
113         head = current->next;
114         delete current;
115     }
116     else
117     {
118         Node *previous = NULL;
119         for (int i = 0; i < index; i++)
120         {
121             previous = current;
122             current = current->next;
123         }
124         previous->next = current->next;
125         delete current;
126         length--;
127     }
128 }
129
130 void print() const
131 {
132     Node *current = head;
133     while (current != NULL)
134     {
135         cout << current->data << " ";
136         current = current->next;
137     }
138     cout << endl;
139 }
140
141 int main()
142 {
143     LinkedList list;
144     list.insert(10);
145     list.insert(20);
146     list.insert(30);
147     list.insert(40);
148     list.insert(50);
149     list.print();
150     cout << "Length of list: " << list.getLength() << endl;
151     cout << "Value at index 2: " << list.get(2) << endl;
152     list.print();
153     cout << "Value at index 0: " << list.get(0) << endl;
154     list.print();
155     list.insertAt(2, 60);
156     list.print();
157     list.insertAt(5, 70);
158     cout << "Value at index 5: " << list.get(5) << endl;
159     list.print();
160     list.removeAt(2);
161     cout << "Value at index 2: " << list.get(2) << endl;
162     list.print();
163     list.removeAt(0);
164     cout << "Value at index 0: " << list.get(0) << endl;
165     list.print();
166     return 0;
167 }

```

```
8 Node *next;
PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

PS C:\Users\MSI PC\Desktop\a> cd "c:\Users\MSI PC\Desktop\a\" ; if ($?) {
10 11 12 13
Current length: 4
Value at index 1: 11
After pushing value 99:
10 11 12 13 99
Current length: 5
After inserting value 77 at index 1:
10 77 11 12 13 99
Current length: 6
After removing value at index 3:
10 77 11 13 99
Current length: 5
PS C:\Users\MSI PC\Desktop\a> 
```

Exercise3:

1. Analyze the **key limitations** of arrays and single linked lists regarding some **specific cases**:
 - ✓ Identify use cases where limitations can appear
 - ✓ Compare the performances of the 2 data structures regarding each use case

Examples of use cases:

- *Going backward (from the end to the beginning of the list)*
- *Inserting a value in the middle of the list*
- *Deleting a value at the beginning*
- *Sorting a list of numbers.*
- *...*

2. Present your **analysis results** using a table:

	ARRAY	LINKED LIST
<i>Use case 1</i>	Not perform	Perform
<i>Use case 2</i>	Not perform	Not perform
<i>Use case 3</i>	Not perform	Perform
Etc...	Perform	Not perform

3. **Explain your results** in terms of time/space **complexity**.

Array:

- Time Complexity:
Accessing one element: $O(1)$.

Insert and Remove: $O(n)$ it require us to shift the elements;
- Space Complexity: Fixed size;

Linked List:

- Time Complexity:
Accessing an element: $O(n)$.

Insert and remove: $O(1)$ (
- Space Complexity: Dynamic size,

4. Identify 3 Real-World Scenarios. For each scenario, describe which structure is the most suitable

Examples of real scenarios:

- *A music player where you need to go to the previous song.*
- *A round-robin scheduling system that loops through tasks continuously.*

Real-World Scenario	Most Appropriate Data Structure	Reason
Storing and sorting exam scores	Array	Fixed, static list of scores with efficient sorting and access.
Storing a fixed list of employees in an organization	Array	Fixed size, constant-time access to employee data using indices.
Music player's history of recently played songs	Linked List	Dynamic tracking of recently played songs with easy addition/removal of songs.

