Course T1Y2: Advanced Algorithms

Lecturer: Bou Channa

Student's name: Chea Ilong

ID: 100022

Group: 1 SE Gen10

Lab 4: Assignment

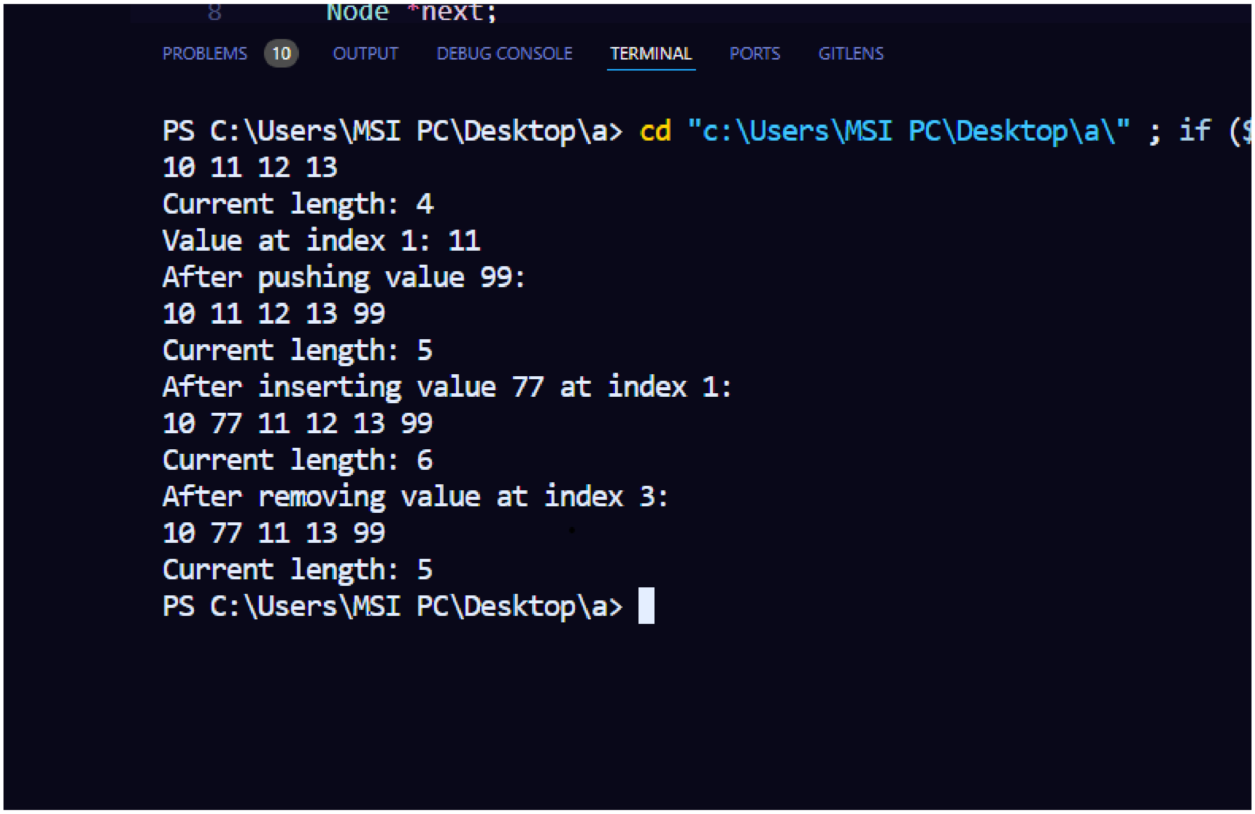
Exercise1:

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

class Array {
    int *data;
    int size;
    int length;

public:
    Array(int size) {
        this->size = size;
        this->length = 0;
        this->data = new int[size];
    }

    ~Array() {
        delete[] data;
    }

    void element(int data) {
        if (length == size) {
            newSize();
        }
        this->data[length] = data;
        length++;
    }

    int getLength() const {
        return length;
    }

    int getSize() const {
        return size;
    }

    int getIndex(int index) const {
        if (index >= 0 && index < length) {
            return data[index];
        }
        return -1;
    }

    void push(int data) {
        if (length == size) {
            newSize();
        }
        this->data[length] = data;
        length++;
    }

    void insertAt(int index, int value) {
        if (index < 0 || index > length) {
            return;
        }
        if (length == size) {
            newSize();
        }
        for (int i = length; i > index; i--) {
            this->data[i] = this->data[i - 1];
        }
        this->data[index] = value;
        length++;
    }

    void removeAt(int index) {
        if (index < 0 || index >= length) {
            return;
        }
        for (int i = index; i < length - 1; i++) {
            this->data[i] = this->data[i + 1];
        }
        length--;
    }

    void print() const {
        for (int i = 0; i < length; ++i) {
            cout << data[i] << " ";
        }
        cout << endl;
    }

private:
    void newSize() {
        size = size * 2;
        int *newData = new int[size];
        for (int i = 0; i < length; i++) {
            newData[i] = data[i];
        }
        delete[] data;
        data = newData;
    }
};

int main() {
    Array arr(5);

    arr.element(10);
    arr.element(11);
    arr.element(12);
    arr.element(13);
    arr.element(14);

    arr.print();
    cout << "Current length: " << arr.getLength() << ", Current size: " << arr.getSize() << endl;

    cout << "The value at index 2: " << arr.getIndex(2) << endl;

    arr.push(5);
    cout << "After pushing value 5:" << endl;
    arr.print();
    cout << "Current length: " << arr.getLength() << ", Current size: " << arr.getSize() << endl;

    arr.insertAt(5, 99);
    cout << "After inserting 99 at index 5:" << endl;
    arr.print();
    cout << "Current length: " << arr.getLength() << ", Current size: " << arr.getSize() << endl;

    arr.removeAt(5);
    cout << "After removing value at index 5:" << endl;
    arr.print();
    cout << "Current length: " << arr.getLength() << ", Current size: " << arr.getSize() << endl;

    return 0;
}
```

```
PS C:\Users\MSI PC\Desktop\a> cd "c:\Users\MS
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:

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

class Node
{
public:
    int data;
    Node *next;
    Node(int data)
    {
        this->data = data;
        this->next = NULL;
    }
};

class linkedlist
{
    int length;
    Node *head;

public:
    linkedlist()
    {
        length = 0;
        head = NULL;
    }

    void insert(int data)
    {
        Node *newNode = new Node(data);
        newNode->next = head;
        head = newNode;
        length++;
    }

    ~linkedlist()
    {
        Node *current = head;
        while (current != NULL)
        {
            Node *temp = current;
            current = current->next;
            delete temp;
        }
    }

    int getLength() const
    {
        return length;
    }

    int get(int index) const
    {
        if (index >= 0 && index < length)
        {
            Node *current = head;
            for (int i = 0; i < index; i++)
            {
                current = current->next;
            }
            return current->data;
        }
        return -1;
    }

    void push(int value)
    {
        Node *newNode = new Node(value);
        if (head == NULL)
        {
            head = newNode;
        }
        else
        {
            Node *current = head;
            while (current->next != NULL)
            {
                current = current->next;
            }
            current->next = newNode;
        }
        length++;
    }

    void insertAt(int index, int value)
    {
        if (index >= 0 && index <= length)
        {
            if (index == 0)
            {
                insert(value);
            }
            else
            {
                Node *newNode = new Node(value);
                Node *current = head;
                for (int i = 0; i < index - 1; i++)
                {
                    current = current->next;
                }
                newNode->next = current->next;
                current->next = newNode;
                length++;
            }
        }
    }

    void removeAt(int index)
    {
        if (index >= 0 && index < length)
        {
            Node *current = head;
            if (index == 0)
            {
                head = current->next;
                delete current;
            }
            else
            {
                Node *previous = NULL;
                for (int i = 0; i < index; i++)
                {
                    previous = current;
                    current = current->next;
                }
                previous->next = current->next;
                delete current;
            }
            length--;
        }
    }

    void print() const
    {
        Node *current = head;
        while (current != NULL)
        {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};

int main()
{
    linkedlist list;
    list.insert(13);
    list.insert(12);
    list.insert(11);
    list.insert(10);
    list.print();
    cout << "Current length: " << list.getLength() << endl;

    cout << "Value at index 1: " << list.get(1) << endl;

    list.push(99);
    cout << "After pushing value 99: " << endl;
    list.print();
    cout << "Current length: " << list.getLength() << endl;

    list.insertAt(1, 77);
    cout << "After inserting value 77 at index 1: " << endl;
    list.print();
    cout << "Current length: " << list.getLength() << endl;

    list.removeAt(3);
    cout << "After removing value at index 3: " << endl;
    list.print();
    cout << "Current length: " << list.getLength() << endl;

    return 0;
}
```



```
    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
Use case 1	Not perform	Perform
Use case 2	Not perform	Not perform
Use case 3	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).

  Insertt 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.