```
#include <iostream>

class MyStack {
    int Stack[100]; // Array to hold stack elements
    int Top;        // Index of the top element in the stack
    int MaxSize;    // Maximum size of the stack

public:
    // Constructor to initialize the stack
    MyStack(int Size = 100) {
        MaxSize = Size;
        Top = 0;
    }

    // Function to check if the stack is empty
    bool isEmpty() {
        return (Top == 0);
    }

    // Function to check if the stack is full
    bool isFull() {
        return (Top == MaxSize);
    }

    // Function to push an element to the stack
    bool push(int Element) {
        if (!isFull()) {
            Stack[Top++] = Element;
```

```
        return true;
    } else {
        std::cout << "Stack is full!" << std::endl;
        return false;
    }
}

// Function to pop an element from the stack
bool pop() {
    if (!isEmpty()) {
        --Top;
        return true;
    } else {
        std::cout << "Stack is empty!" << std::endl;
        return false;
    }
}

// Function to return the top element of the stack
int topElement() {
    if (!isEmpty()) {
        return Stack[Top - 1];
    } else {
        std::cout << "Stack is empty!" << std::endl;
        return -1; // Indicates an error condition
    }
}
```

```
    // Function to print the whole stack from top to bottom
    void show() {
        if (isEmpty()) {
            std::cout << "Stack empty" << std::endl;
            return;
        }
        for (int i = Top - 1; i >= 0; i--) {
            std::cout << Stack[i] << std::endl;
        }
    }
};

// Main function to demonstrate the usage of MyStack
int main() {
    MyStack stack(7); // Create a stack with a maximum size of 7

    // Push some elements into the stack
    stack.push(10);
    stack.push(20);
    stack.push(30);
    stack.push(40);
    stack.push(50);
    stack.push(60);
    stack.push(70);

    std::cout << "Current stack:" << std::endl;
    stack.show(); // Output the stack elements
```

```
    std::cout << "Stack is full: " << stack.isFull() << std::endl; // Output: 1 (true)

    // Pop two elements from the stack
    stack.pop();
    stack.pop();

    std::cout << "Stack after popping 2 elements:" << std::endl;
    stack.show(); // Output the stack elements

    std::cout << "Top element: " << stack.topElement() << std::endl; // Output the top element

    std::cout << "Stack is empty: " << stack.isEmpty() << std::endl; // Output: 0 (false)

    return 0;
}
```

```
stack.push(10); // Stack: [10]
stack.push(20); // Stack: [10, 20]
stack.push(30); // Stack: [10, 20, 30]
stack.push(40); // Stack: [10, 20, 30, 40]
stack.push(50); // Stack: [10, 20, 30, 40, 50]
stack.push(60); // Stack: [10, 20, 30, 40, 50, 60]
stack.push(70); // Stack: [10, 20, 30, 40, 50, 60, 70]

Current stack:
70
60
```

50

40

30

20

10

stack.pop(); // Removes 70

stack.pop(); // Removes 60

Stack: [10, 20, 30, 40, 50]

Stack after popping 2 elements:

50

40

30

20

10

Top element: 50

Stack is empty: 0

Output:

Current stack:

70

60

50

40

30

20

10

Stack is full: 1

Stack after popping 2 elements:

50

40

30

20

10

Top element: 50

Stack is empty: 0

2nd code:

```
#include <iostream>

using namespace std;

class MyStack {
    int *Stack;  // Pointer to dynamically allocated array to hold stack elements
    int Top;     // Index of the top element in the stack
    int MaxSize; // Maximum size of the stack

public:
    // Constructor to initialize the stack
    MyStack(int Size = 100) {
        MaxSize = Size;
        Stack = new int[MaxSize]; // Create array dynamically
        Top = 0;
    }

    // Destructor to release the memory
    ~MyStack() {
```

```
    delete[] Stack; // Release the memory for stack
}

// Function to check if the stack is empty
bool isEmpty() {
    return (Top == 0);
}

// Function to check if the stack is full
bool isFull() {
    return (Top == MaxSize);
}

// Function to push an element to the stack
bool push(int Element) {
    if (!isFull()) {
        Stack[Top++] = Element;
        return true;
    } else {
        cout << "Stack is full!" << endl;
        return false;
    }
}

// Function to pop an element from the stack
bool pop() {
    if (!isEmpty()) {
        --Top;
```

```
        return true;
    } else {
        cout << "Stack is empty!" << endl;
        return false;
    }
}

// Function to return the top element of the stack
int topElement() {
    if (!isEmpty()) {
        return Stack[Top - 1];
    } else {
        cout << "Stack is empty!" << endl;
        return -1; // Indicates an error condition
    }
}

// Function to print the whole stack from top to bottom
void show() {
    if (isEmpty()) {
        cout << "Stack empty" << endl;
        return;
    }
    for (int i = Top - 1; i >= 0; i--) {
        cout << Stack[i] << endl;
    }
}
```

```
    // Function to resize the stack
    void resize(int size) {
        int *newStack = new int[size];
        for (int i = 0; i < Top && i < size; i++) {
            newStack[i] = Stack[i];
        }
        delete[] Stack;
        Stack = newStack;          //the pointer Stack to point to the newly allocated stack
                                        (newStack).
        MaxSize = size;  //it updates the MaxSize member variable to the new size.
    }
};

// Main function to demonstrate the usage of MyStack
int main() {
    MyStack stack(7); // Create a stack with a maximum size of 7

    // Push some elements into the stack
    stack.push(10);
    stack.push(20);
    stack.push(30);
    stack.push(40);
    stack.push(50);
    stack.push(60);
    stack.push(70);

    cout << "Current stack:" << endl;
    stack.show(); // Output the stack elements
```

```
    cout << "Stack is full: " << stack.isFull() << endl; // Output: 1 (true)

    // Resize the stack to a larger size
    stack.resize(10);
    cout << "Resized the stack to a larger size." << endl;

    // Push more elements into the resized stack
    stack.push(80);
    stack.push(90);
    stack.push(100);

    cout << "Stack after resizing and adding more elements:" << endl;
    stack.show(); // Output the stack elements

    // Pop two elements from the stack
    stack.pop();
    stack.pop();

    cout << "Stack after popping 2 elements:" << endl;
    stack.show(); // Output the stack elements

    cout << "Top element: " << stack.topElement() << endl; // Output the top element

    cout << "Stack is empty: " << stack.isEmpty() << endl; // Output: 0 (false)

    return 0;
}
```

Output:

Current stack:

70

60

50

40

30

20

10

Stack is full: 1

Resized the stack to a larger size.

Stack after resizing and adding more elements:

100

90

80

70

60

50

40

30

20

10

Stack after popping 2 elements:

80

70

60

50

40

30

Top element: 80

Stack is empty: 0