DAA EXPERIMENTS UNIT-1

Experiment-1:

Aim: Analyze if the stack is empty or full, and if elements are present, return the top element in the stack using templates. Also, perform push and pop operations on the stack.

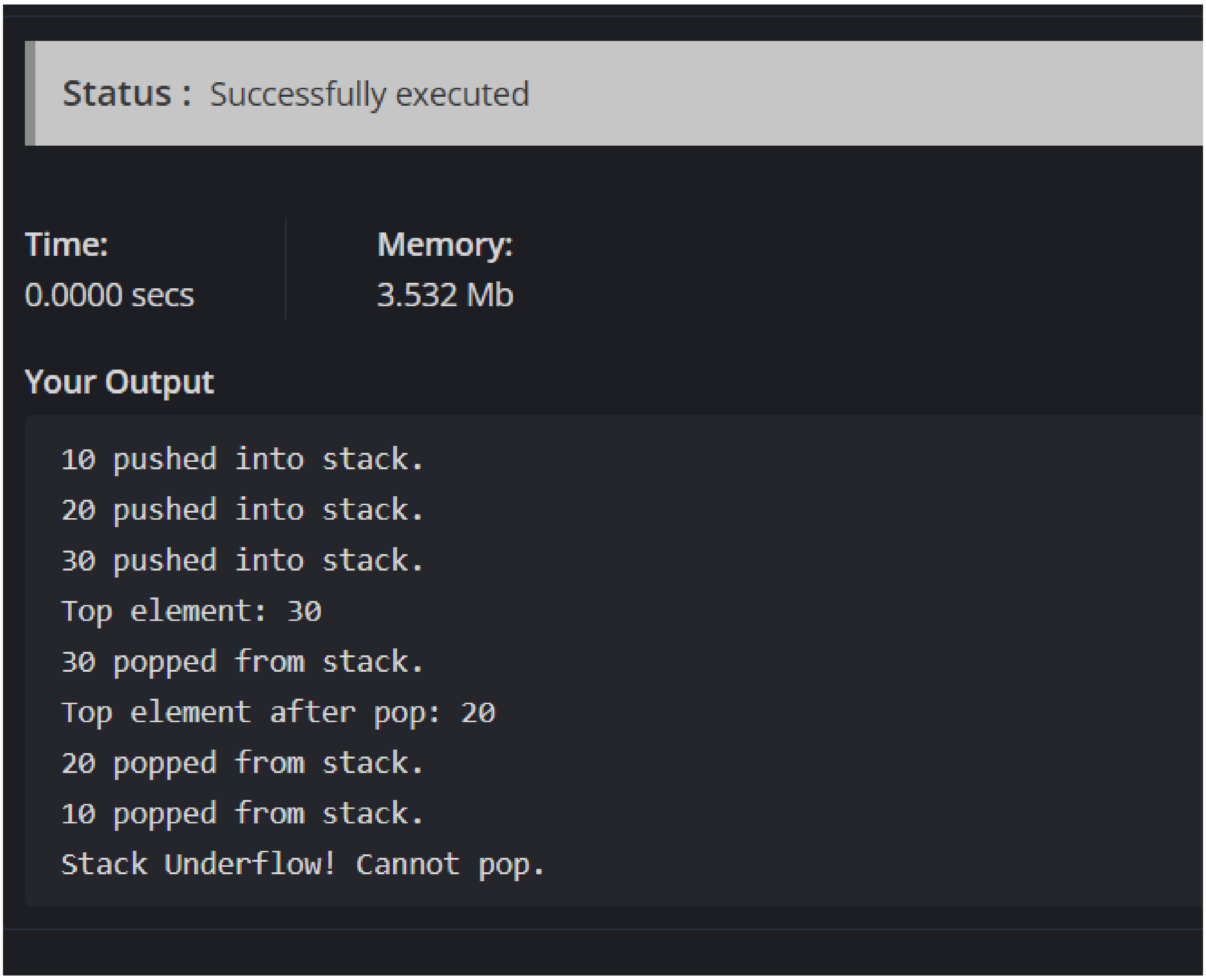
Code:

```
#include <iostream>

using namespace std;

#define SIZE 10

class Stack {

private:

   int arr[SIZE];

   int top;

public:

   // Constructor

   Stack() {

      top = -1;

   }

   // Check if stack is empty

   bool isEmpty() {

      return (top == -1);

   }

   // Check if stack is full

   bool isFull() {

      return (top == SIZE - 1);

   }
```

```
    // Push element into stack
    void push(int value) {
        if (isFull()) {
            cout << "Stack Overflow! Cannot push " << value << endl;
        } else {
            arr[++top] = value;
            cout << value << " pushed into stack." << endl;
        }
    }

    // Pop element from stack
    void pop() {
        if (isEmpty()) {
            cout << "Stack Underflow! Cannot pop." << endl;
        } else {
            cout << arr[top--] << " popped from stack." << endl;
        }
    }

    // Return top element
    int peek() {
        if (isEmpty()) {
            cout << "Stack is empty. No top element." << endl;
            return -1;
        } else {
            return arr[top];
        }
    }
};
```

```
int main() {
    Stack s;
    s.push(10);
    s.push(20);
    s.push(30);
    cout << "Top element: " << s.peek() << endl;
    s.pop();
    cout << "Top element after pop: " << s.peek() << endl;
    s.pop();
    s.pop();
    s.pop();
    return 0;
}
```

OUTPUT:



Status : Successfully executed

Time:
0.0000 secs

Memory:
3.532 Mb

Your Output

```
10 pushed into stack.
20 pushed into stack.
30 pushed into stack.
Top element: 30
30 popped from stack.
Top element after pop: 20
20 popped from stack.
10 popped from stack.
Stack Underflow! Cannot pop.
```

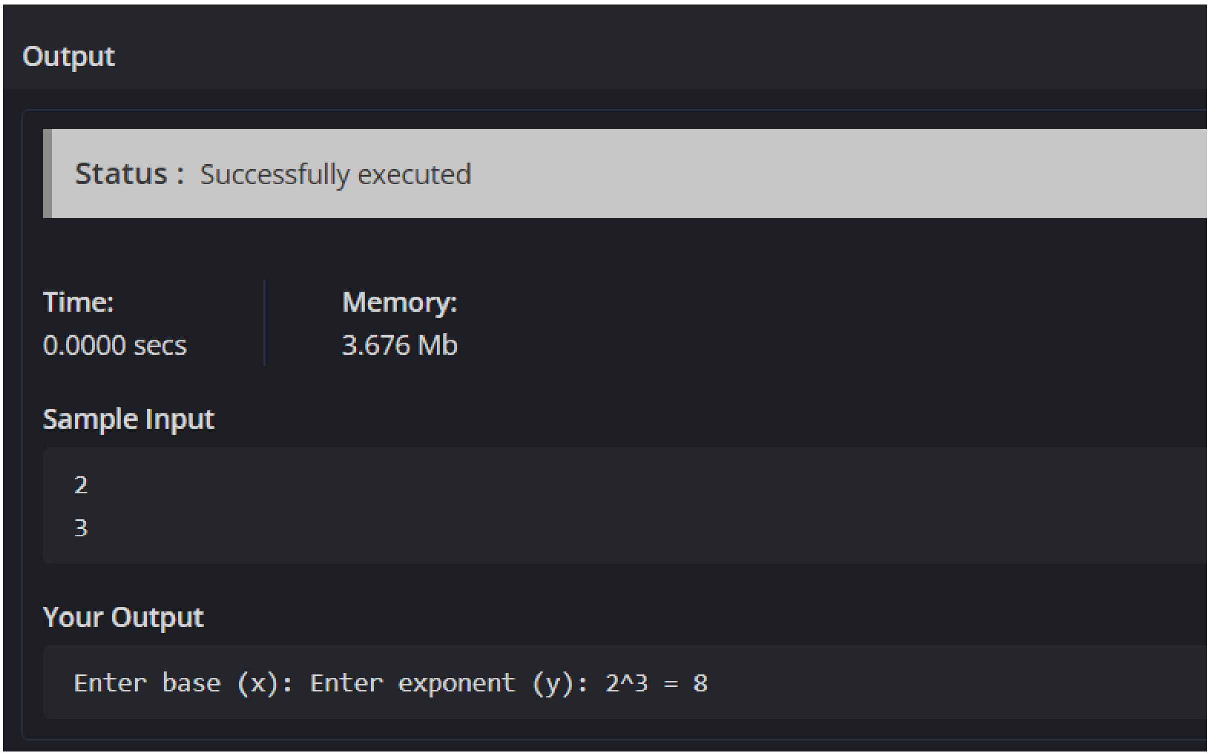
Experiment-2

Aim: Code implement power function in O(logn) time complexity.

Code:

```
#include <iostream>
using namespace std;
double power(double x, int y) {
    if (y == 0)
        return 1;
    double temp = power(x, y / 2);
    if (y % 2 == 0) {
        return temp * temp;
    }
    else if (y > 0) {
        return x * temp * temp;
    }
    else {
        return (temp * temp) / x;
    }
}
int main() {
    double x;
    int y;
    cout << "Enter base (x): ";
    cin >> x;
    cout << "Enter exponent (y): ";
    cin >> y;
    cout << x << "^" << y << " = " << power(x, y) << endl;
    return 0;
}
```

OUTPUT:



```
Status : Successfully executed

Time:
0.0000 secs

Memory:
3.676 Mb

Sample Input
2
3

Your Output
Enter base (x): Enter exponent (y): 2^3 = 8
```

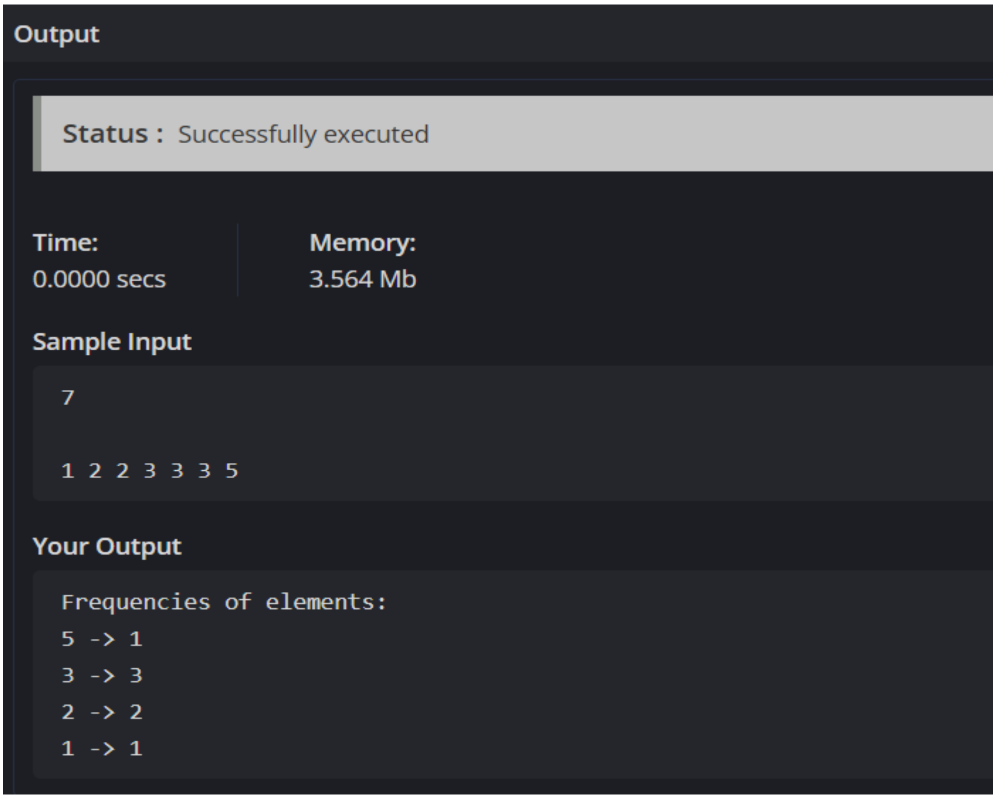
Experiment-3:

Aim: Code to find the frequency of elements in a given array in O(n) Time Complexity

Code:

```
#include <iostream>

#include <unordered_map>

#include<vector>

using namespace std;

void findFrequency(vector<int> arr, int n) {

    unordered_map<int, int> freq;

    for (int i = 0; i < n; i++) {

        freq[arr[i]]++;

    }

     for (auto it : freq) {
        cout << it.first << " -> " << it.second << endl;
    }
}
int main() {
    int n;
    cin>>n;
    vector<int> arr(n);
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    cout << "Frequencies of elements:\n";
    findFrequency(arr, arr.size());
    return 0;
}
```

OUTPUT:



Output

Status : Successfully executed

Time: 0.0000 secs

Memory: 3.564 Mb

Sample Input

```
7

1 2 2 3 3 3 5
```

Your Output

```
Frequencies of elements:
5 -> 1
3 -> 3
2 -> 2
1 -> 1
```

Experiment-4

Aim: Apply the concept of Linked list and write code to Insert and Delete an element at the beginning and atend in Doubly and Circular Linked List.

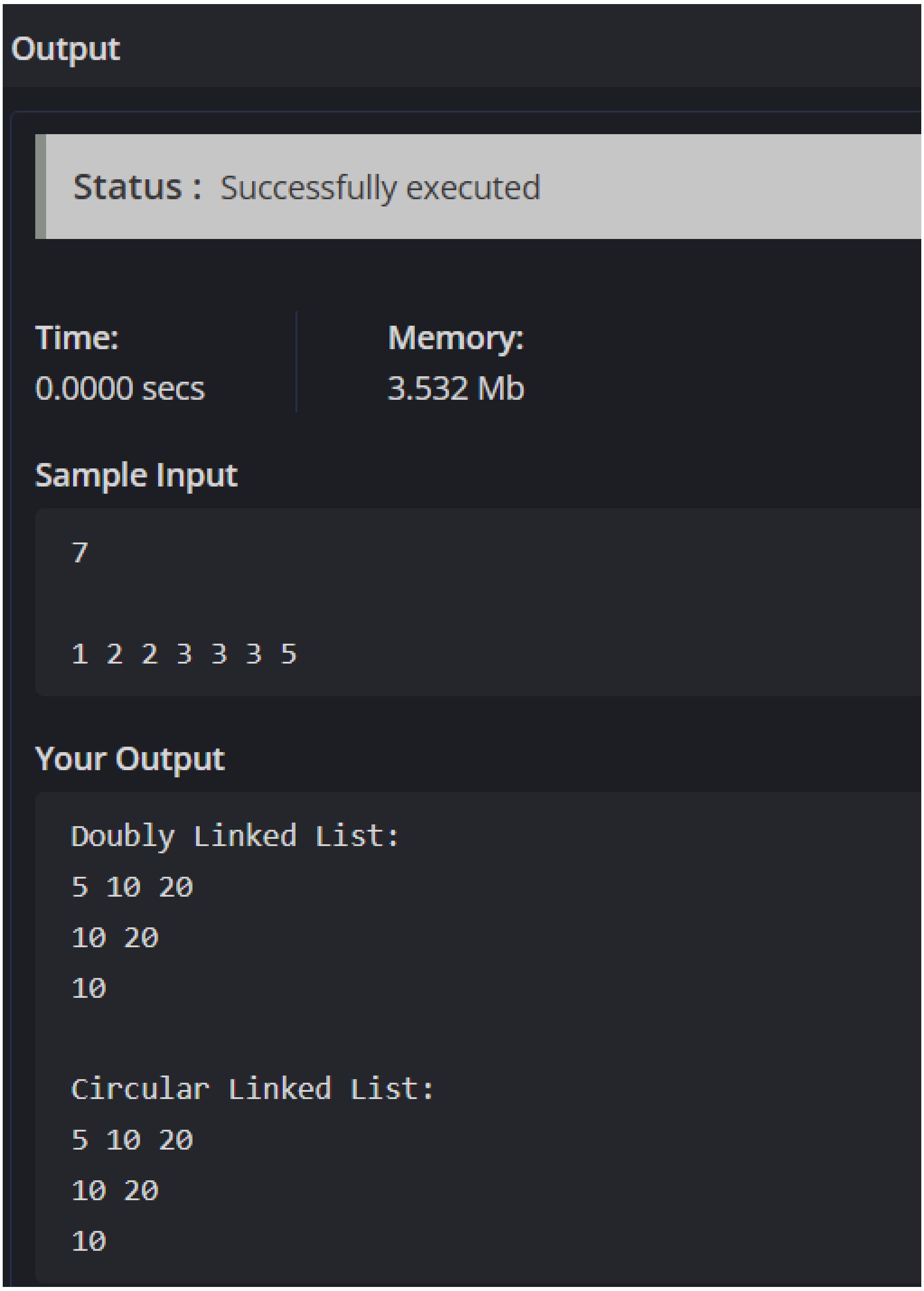
Code:

```
#include <iostream>
using namespace std;
// Doubly Linked List Node
struct DNode {
    int data;
    DNode* prev;
    DNode* next;
    DNode(int val) : data(val), prev(NULL), next(NULL) {}
};
// Circular Linked List Node
struct CNode {
    int data;
    CNode* next;
    CNode(int val) : data(val), next(NULL) {}
};
/// ==================== DOUBLY LINKED LIST ====================
class DoublyLinkedList {
    DNode* head;
public:
    DoublyLinkedList() : head(NULL) {}

    void insertAtBegin(int val) {
        DNode* newNode = new DNode(val);
        if (head != NULL) {
            newNode->next = head;
            head->prev = newNode;
        }
        head = newNode;
    }

    void insertAtEnd(int val) {
        DNode* newNode = new DNode(val);
        if (head == NULL) {
            head = newNode;
            return;
        }
        DNode* temp = head;
        while (temp->next != NULL) temp = temp->next;
        temp->next = newNode;
        newNode->prev = temp;
    }

    void deleteAtBegin() {
        if (head == NULL) return;
        DNode* temp = head;
        head = head->next;
        if (head) head->prev = NULL;
        delete temp;
    }

    void deleteAtEnd() {
        if (head == NULL) return;
        DNode* temp = head;
        while (temp->next != NULL) temp = temp->next;
        if (temp->prev) temp->prev->next = NULL;
        else head = NULL; // only one node
        delete temp;
    }

    void display() {
        DNode* temp = head;
        while (temp != NULL) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};

/// ==================== CIRCULAR LINKED LIST ====================
class CircularLinkedList {
    CNode* head;
public:
    CircularLinkedList() : head(NULL) {}

    void insertAtBegin(int val) {
        CNode* newNode = new CNode(val);
        if (head == NULL) {
            newNode->next = newNode;
            head = newNode;
            return;
        }
        CNode* temp = head;
        while (temp->next != head) temp = temp->next;
        newNode->next = head;
        temp->next = newNode;
        head = newNode;
```

```
}

void insertAtEnd(int val) {
    CNode* newNode = new CNode(val);
    if (head == NULL) {
        newNode->next = newNode;
        head = newNode;
        return;
    }
    CNode* temp = head;
    while (temp->next != head) temp = temp->next;
    temp->next = newNode;
    newNode->next = head;
}

void deleteAtBegin() {
    if (head == NULL) return;
    if (head->next == head) {
        delete head;
        head = NULL;
        return;
    }
    CNode* temp = head;
    CNode* last = head;
    while (last->next != head) last = last->next;
    last->next = head->next;
    head = head->next;
    delete temp;
}
```

```
    void deleteAtEnd() {
        if (head == NULL) return;
        if (head->next == head) {
            delete head;
            head = NULL;
            return;
        }
        CNode* temp = head;
        while (temp->next->next != head) temp = temp->next;
        delete temp->next;
        temp->next = head;
    }

    void display() {
        if (head == NULL) return;
        CNode* temp = head;
        do {
            cout << temp->data << " ";
            temp = temp->next;
        } while (temp != head);
        cout << endl;
    }
};

/// ==================== DRIVER CODE ====================
int main() {
    cout << "Doubly Linked List:\n";
    DoublyLinkedList dll;
    dll.insertAtBegin(10);
    dll.insertAtEnd(20);
    dll.insertAtBegin(5);
    dll.display();
    dll.deleteAtBegin();
    dll.display();
    dll.deleteAtEnd();
    dll.display();

    cout << "\nCircular Linked List:\n";
    CircularLinkedList cll;
    cll.insertAtBegin(10);
    cll.insertAtEnd(20);
    cll.insertAtBegin(5);
    cll.display();
    cll.deleteAtBegin();
    cll.display();
    cll.deleteAtEnd();
    cll.display();

    return 0;
}
```

OUTPUT:



Status : Successfully executed

Time:
0.0000 secs

Memory:
3.532 Mb

Sample Input

```
7

1 2 2 3 3 3 5
```

Your Output

```
Doubly Linked List:
5 10 20
10 20
10

Circular Linked List:
5 10 20
10 20
10
```