Experiment 3

Student Name: Ankush
UID: 23BCS12742
Branch: CSE
Section/Group: KRG-3B
Semester: 5th
Date of Performance: 04/08/25
Subject Name: DAA (23CSH-301)

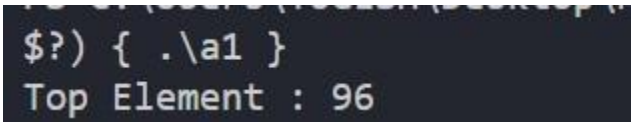
A. Stack Implementation

1. Stack Implementation using array:

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

class Stack {
    int arr[10];
    int top;
public:
    Stack() {
        top = -1;
    }

    void push(int x) {
        if (top >= 9) {
            cout << "OVERFLOW!" << endl;
            return;
        }
        arr[++top] = x;
    }
    void pop() {
```

```
        if (top < 0) {
            cout << "UNDERFLOW!!" << endl;
            return;
        }
        top--;
    }
    int peek() {
        if (top < 0) {
            cout << "UNDERFLOW!!" << endl;
            return -1;
        }
        return arr[top];
    }
};
int main() {
    Stack s;
    s.push(96);
    s.push(100);
    s.pop();
    cout << "Top Element: " << s.peek() << endl;
    return 0;
}
```

Output:



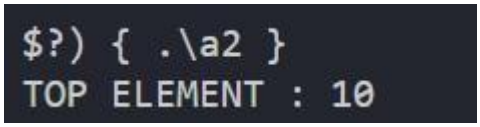
2. Stack implementation using Linked List:

```
#include <iostream>
using namespace std;
struct node {
    int data;
    node* next;
```

```
    node(int x) {
        data = x;
        next = NULL;
    }
};

class stack {
    node* top;
public:
    stack() {
        top = NULL;
    }

    void push(int x) {
        node* temp = new node(x);
        temp->next = top;
        top = temp;
    }

    void pop() {
        if (top == NULL) {
            cout << "UNDERFLOW!!";
            return;
        }
        node* temp = top;
        top = top->next;
        delete temp;
    }

    int peek() {
        if (top == NULL) {
            cout << "UNDERFLOW!!";
            return -1;
        }
        return top->data;
    }
};

int main() {
    stack s;
    s.push(10);
    s.push(20);
    s.push(30);
    s.pop();
    s.pop();
    cout << "TOP ELEMENT : " << s.peek() << endl;
```

```
    return 0;
}
```

Output:



B. Queue Implementation

1. Queue Implementation using array:

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

class queue {
    int arr[10];
    int rear;
    int front;
public:
    queue() {
        rear = front = -1;
    }

    void enqueue(int x) {
        if (rear >= 9) {
            cout << "overflow!!";
            return;
        }
        if (front == -1) {
            front = 0;
        }
        arr[++rear] = x;
    }

    void dequeue() {
        if (front == -1 || front > rear) {
            cout << "Queue Underflow\n";
            return;
        }
        front++;
    }

    int peek() {
```

```
        if (front == -1 || front > rear) {
            cout << "Queue is Empty\n";
            return -1;
        }
        return arr[front];
    }
};

int main() {
    queue q;
    q.enqueue(10);
    q.enqueue(20);
    q.dequeue();
    cout << "Front element: " << q.peek() << endl;
    return 0;
}
```

Output:

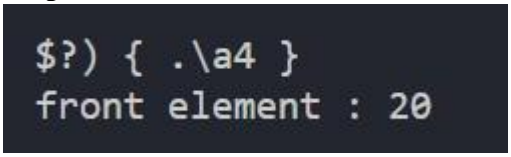
```
$?) { .\a3 }
Front element: 20
```

2. Queue Implementation using Linked List:

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

struct node {
    int data;
    node* next;
    node(int x) {
        data = x;
        next = NULL;
    }
};

class queue {
    node* rear;
    node* front;
public:
    queue() {
        rear = front = NULL;
    }

    void enqueue(int x) {
        node* temp = new node(x);
```

```
        if (rear == NULL) {
            front = rear = temp;
            return;
        }
        rear->next = temp;
        rear = temp;
    }

    void dequeue() {
        if (front == NULL) {
            cout << "UNDERFLOW!!";
            return;
        }
        node* temp = front;
        front = front->next;
        delete temp;
        if (front == NULL) {
            rear = NULL;
        }
    }

    int peek() {
        if (front == NULL) {
            cout << "UNDERFLOW!!";
            return -1;
        }
        return front->data;
    }
};

int main() {
    queue q;
    q.enqueue(10);
    q.enqueue(20);
    q.dequeue();
    cout << "front element : " << q.peek() << endl;
    return 0;
}
```

Output:

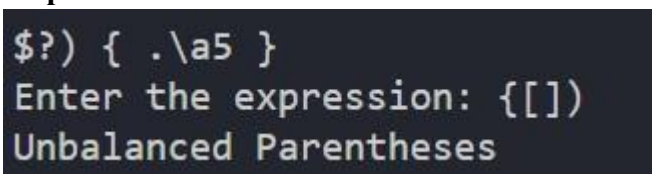


C. Parentheses Checker

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

class stack {
   int arr[10];
   int top;
public:
   stack() {
      top = -1;
   }

   void push(int x) {
      if (top >= 9) {
         cout << "OVERFLOW!!\n";
         return;
      }
      arr[++top] = x;
   }

   void pop() {
      if (top == -1) {
         cout << "UNDERFLOW!!\n";
         return;
      }
      top--;
   }

   int peek() {
      if (top == -1) {
         cout << "UNDERFLOW!!\n";
         return -1;
      }
      return arr[top];
   }

   bool isEmpty() {
      return (top == -1);
   }
};

int main() {
```

```
    string e;
    cout << "Enter the expression: ";
    cin >> e;
    stack s;

    for (int i = 0; i < e.length(); i++) {
        if (e[i] == '(' || e[i] == '{' || e[i] == '[') {
            s.push(e[i]);
        } else if (e[i] == ')' || e[i] == '}' || e[i] == ']') {
            if (s.isEmpty()) {
                cout << "Unbalanced Parentheses\n";
                return 0;
            }
            char topChar = s.peek();
            if ((e[i] == ')' && topChar != '(') ||
                (e[i] == '}' && topChar != '{') ||
                (e[i] == ']' && topChar != '[')) {
                cout << "Unbalanced Parentheses\n";
                return 0;
            }
            s.pop();
        }
    }

    if (s.isEmpty()) {
        cout << "Balanced Parentheses\n";
    } else {
        cout << "Unbalanced Parentheses\n";
    }

    return 0;
}
```

Output:



D. Circular Queue using Array

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

class CircularQueue {
   int *arr;
   int front, rear, size;
public:
   CircularQueue(int n) {
      size = n;
      arr = new int[size];
      front = -1;
      rear = -1;
   }

   // Check if queue is empty
   bool isEmpty() {
      return (front == -1);
   }
   // Check if queue is full
   bool isFull() {
      return ((rear + 1) % size == front);
   }
   // Insert element
   void enqueue(int value) {
      if (isFull()) {
         cout << "Queue is Full\n";
         return;
      }
      if (isEmpty()) {
```

```
        front = rear = 0;
    } else {
        rear = (rear + 1) % size;
    }
    arr[rear] = value;
}
// Remove element
void dequeue() {
    if (isEmpty()) {
        cout << "Queue is Empty\n";
        return;
    }
    if (front == rear) { // only one element
        front = rear = -1;
    } else {
        front = (front + 1) % size;
    }
}
// Get front element
int peek() {
    if (isEmpty()) {
        cout << "Queue is Empty\n";
        return -1;
    }
    return arr[front];
}
// Display queue elements
void display() {
    if (isEmpty()) {
        cout << "Queue is Empty\n";
        return;
```

```
        }
        int i = front;
        while (true) {
            cout << arr[i] << " ";
            if (i == rear) break;
            i = (i + 1) % size;
        }
        cout << endl;
    }
};

int main() {
    CircularQueue q(5);

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.enqueue(40);
    q.display();

    q.dequeue();
    q.display();

    q.enqueue(50);
    q.enqueue(60);
    q.display();

    cout << "Front element: " << q.peek() << endl;

    return 0;
}
```

E. Priority Queue using Array

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

class SortedPriorityQueue {
   int *arr;
   int size;
   int count; // number of elements
public:
   SortedPriorityQueue(int n) {
      size = n;
      arr = new int[size];
      count = 0;
   }

   bool isEmpty() {
      return count == 0;
   }

   bool isFull() {
      return count == size;
   }

   // Insert in sorted order (descending: highest priority first)
   void enqueue(int value) {
      if (isFull()) {
         cout << "Queue is Full\n";
         return;
      }
```

```
    int i;
    // Shift elements to maintain descending order
    for (i = count - 1; (i >= 0 && arr[i] < value); i--) {
        arr[i + 1] = arr[i];
    }
    arr[i + 1] = value;
    count++;
}

// Remove highest priority element (front)
void dequeue() {
    if (isEmpty()) {
        cout << "Queue is Empty\n";
        return;
    }
    // Just reduce count (since highest priority is at front)
    for (int i = 0; i < count - 1; i++) {
        arr[i] = arr[i + 1];
    }
    count--;
}

// Peek highest priority
int peek() {
    if (isEmpty()) {
        cout << "Queue is Empty\n";
        return -1;
    }
    return arr[0]; // first element has highest priority
}
```

```
    void display() {
        if (isEmpty()) {
            cout << "Queue is Empty\n";
            return;
        }
        for (int i = 0; i < count; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    SortedPriorityQueue pq(5);

    pq.enqueue(30);
    pq.enqueue(50);
    pq.enqueue(10);
    pq.enqueue(40);

    pq.display(); // Should be sorted: 50 40 30 10

    cout << "Highest priority: " << pq.peek() << endl;

    pq.dequeue();
    pq.display();

    return 0;
}
```