1.Implementing Stacks and Queues: o Implement a stack using both array-based and linked list-based approaches.

```
#include <iostream>
using namespace std;
class ArrayStack {
private:
   int* arr;
   int capacity;
   int top;
public:
   ArrayStack(int size) {
      capacity = size;
      arr = new int[capacity];
      top = -1;
   }
 void push(int x) {
      if (top == capacity - 1) return;
      arr[++top] = x;
   }
 int pop() {
      if (top == -1) return -1;
      return arr[top--];
   }

   int peek() {
      if (top == -1) return -1;
      return arr[top];
```

```
    }

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

    ~ArrayStack() {
        delete[] arr;
    }
};

class Node {
public:
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

class LinkedListStack {
private:
    Node* top;

public:
    LinkedListStack() : top(nullptr) {}

    void push(int x) {
        Node* newNode = new Node(x);
```

```
    newNode->next = top;
    top = newNode;
}

int pop() {
    if (top == nullptr) return -1;
    int poppedValue = top->data;
    Node* temp = top;
    top = top->next;
    delete temp;
    return poppedValue;
}

int peek() {
    if (top == nullptr) return -1;
    return top->data;
}

bool isEmpty() {
    return top == nullptr;
}

~LinkedListStack() {
    while (top != nullptr) {
        pop();
    }
}
```

```
};
```

OUTPUT

1

2. Implement a queue using both array-based and linked list-based approaches.

```
#include <iostream>
using namespace std;
class ArrayQueue {
private:
  int* arr;
  int capacity;
  int front;
  int rear;
public:
  ArrayQueue(int size) {
    capacity = size;
    arr = new int[capacity];
    front = -1;
    rear = -1;
  }

 void enqueue(int x) {
    if ((rear + 1) % capacity == front) return;
    if (front == -1) front = 0;
    rear = (rear + 1) % capacity;
    arr[rear] = x;
  }
```

```
    int dequeue() {
        if (front == -1) return -1;
        int dequeuedValue = arr[front];
        if (front == rear) {
            front = rear = -1;
        } else {
            front = (front + 1) % capacity;
        }
        return dequeuedValue;
    }

    int peek() {
        if (front == -1) return -1;
        return arr[front];
    }

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

    ~ArrayQueue() {
        delete[] arr;
    }
};

class LinkedListQueue {
```

```
private:
    class Node {
    public:
        int data;
        Node* next;
        Node(int val) : data(val), next(nullptr) {}
    };

    Node* front;
    Node* rear;

public:
    LinkedListQueue() : front(nullptr), rear(nullptr) {}

    void enqueue(int x) {
        Node* newNode = new Node(x);
        if (rear == nullptr) {
            front = rear = newNode;
            return;
        }
        rear->next = newNode;
        rear = newNode;
    }

    int dequeue() {
        if (front == nullptr) return -1;
        int dequeuedValue = front->data;
```

```
        Node* temp = front;
        front = front->next;
        if (front == nullptr) rear = nullptr;
        delete temp;
        return dequeuedValue;
    }

    int peek() {
        if (front == nullptr) return -1;
        return front->data;
    }

    bool isEmpty() {
        return front == nullptr;
    }

    ~LinkedListQueue() {
        while (front != nullptr) {
            dequeue();
        }
    }
};

int main() {
    ArrayQueue queue(5);
    queue.enqueue(1);
    queue.enqueue(2);
```

```
  queue.dequeue();
  cout << queue.peek() << endl;
  LinkedListQueue linkedQueue;
  linkedQueue.enqueue(1);
  linkedQueue.enqueue(2);
  linkedQueue.dequeue();
 return 0;
}
```

OUTPUT

2

3.System Stack: Simulate the call stack of a program using the stack implementation. Write a function to display the state of the stack at any point.

```
#include <stack>
#include <string>
#include <vector>
using namespace std;
class CallStackSimulator {
private:
stack<string> callStack;
public:
  void pushFunction(const string& functionName) {
    callStack.push(functionName);
    displayStack();
  }
  void popFunction() {
    if (!callStack.empty()) {
```

```
        cout << "Returning from function: " << callStack.top() << endl;
        callStack.pop();
        displayStack();
    } else {
        cout << "Call stack is empty!" << endl;
    }
}
void displayStack() {
    cout << "Current Call Stack: ";
    if (callStack.empty()) {
        cout << "Empty" << endl;
    } else {
        stack<string> tempStack = callStack;
        vector<string> stackContents;
        while (!tempStack.empty()) {
            stackContents.push_back(tempStack.top());
            tempStack.pop();
        }
        for (int i = stackContents.size() - 1; i >= 0; --i) {
            cout << stackContents[i] << " ";
        }
        cout << endl;
    }
}
};
void functionA(CallStackSimulator& stackSimulator);
void functionB(CallStackSimulator& stackSimulator);
```

```
void functionC(CallStackSimulator& stackSimulator);
void functionA(CallStackSimulator& stackSimulator) {
   stackSimulator.pushFunction("functionA");
   functionB(stackSimulator);
   stackSimulator.popFunction();
}
void functionB(CallStackSimulator& stackSimulator) {
   stackSimulator.pushFunction("functionB");
   functionC(stackSimulator);
   stackSimulator.popFunction();
}
void functionC(CallStackSimulator& stackSimulator) {
   stackSimulator.pushFunction("functionC");
   stackSimulator.popFunction();
}
int main() {
   CallStackSimulator stackSimulator;
   functionA(stackSimulator);
 return 0;
}
```

OUTPUT

Current Call Stack: functionA

Current Call Stack: functionA functionB

Current Call Stack: functionA functionB functionC

Returning from function: functionC

Current Call Stack: functionA functionB

Returning from function: functionB

Current Call Stack: functionA

Returning from function: functionA

Current Call Stack: Empty

4. Reversing Data: Use a stack to reverse the contents of a given string

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;
string reverseString(const string& str) {
   stack<char> s;
   for (char ch : str) {
      s.push(ch);
   }
   string reversed;
   while (!s.empty()) {
      reversed += s.top();
      s.pop();
   }
 return reversed;
}
int main() {
   string input = "hello";
   string output = reverseString(input);
 cout << "Reversed string: " << output << endl;
  return 0;
}
```

OUTPUT

Reversed string: olleh

5. Queue Management in Systems: Simulate a basic print queue system where documents are enqueued and dequeued for printing.

```
#include <iostream>
#include <queue>
#include <string>
using namespace std;
class PrintQueue {
private:
  queue<string> q;
public:
  void enqueue(const string& document) {
    q.push(document);
    cout << "Enqueue: " << document << endl;
  }
  void dequeue() {
    if (!q.empty()) {
      string doc = q.front();
      q.pop();
      cout << "Dequeue: " << doc << endl;
    } else {
      cout << "The print queue is empty!" << endl;
    }
  }
  bool isEmpty() {
    return q.empty();
```

```
    }
};

int main() {
    PrintQueue pq;
    pq.enqueue("doc1");
    pq.enqueue("doc2");
    pq.dequeue();
 return 0;
}
```

OUTPUT

Enqueue: doc1