Lab - 2 Stack and Queue

Name: Akash kafle
Roll No: 27
GitHub -> Akash-kafle

Stack:

➢ A stack is a linear data structure where all insertions and deletions are restricted to one end, called the top. It is also known as a Last-In-First-Out (LIFO) list because the last element inserted into a stack is the first element removed. Principal operations on a stack are push (adding an element into a stack) and pop(removing an element from a stack).

Few implementation of Stack are:

- **Expression Evaluation**: Convert and evaluate infix to post-fix expressions.
- **Reversing a List or String:** Reverse the order of elements in a list or characters in a string using a stack.
- **Balancing Symbols:** Check for balanced parentheses and other symbols.
- **Reverse Words in a Sentence**: Use a stack to reverse the order of words or characters in a sentence.

Queue:

➢ A Queue is a First-In-First-Out (FIFO) data structure where the first element inserted is the first element removed. Unlike stacks, insertions and deletions are made at different ends. New elements are added at one end, called the rear, and elements are removed from the other end, called the front. Principal operations on a queue are enqueue (adding an element into a queue) and dequeue (removing an element from a queue).

Few implementation of Queue are:

- **Web Server Requests**: Manage incoming HTTP requests in web servers.
- **I/O Buffering:** Queue input/output operations in operating systems.
- **CPU Task Management:** Queue tasks for the CPU in multitasking environments.
- **Email Queueing:** Queue outgoing emails for orderly sending.

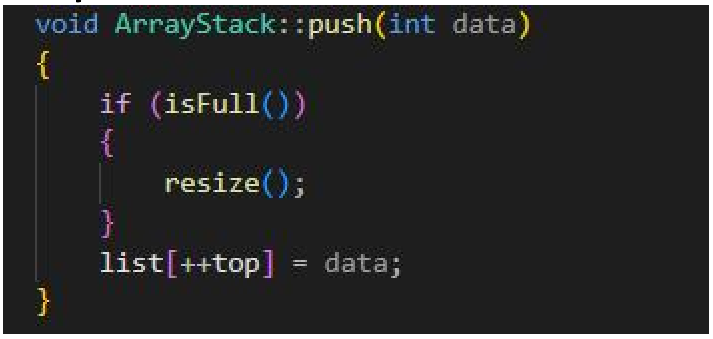
Lab - 2 Stack and Queue

1. Implement Stack data structure using an array as well as a linked list.

(a) push(element):

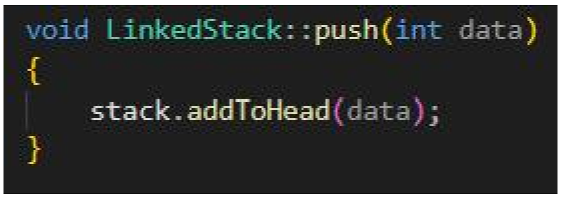
- Adds an element into the stack

Array:



```
void ArrayStack::push(int data)
{
    if (isFull())
    {
        resize();
    }
    list[++top] = data;
}
```

Linked List:

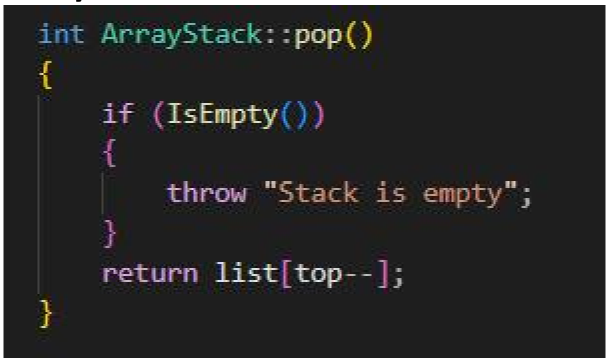


```
void LinkedStack::push(int data)
{
    stack.addToHead(data);
}
```

(b) pop():

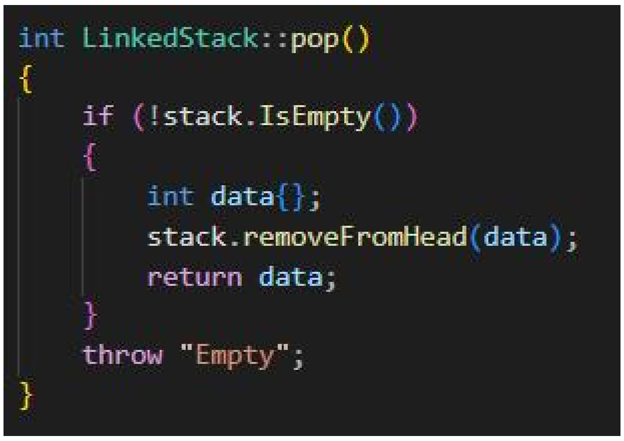
- Removes an element from the stack

Array:



```
int ArrayStack::pop()
{
    if (IsEmpty())
    {
        throw "Stack is empty";
    }
    return list[top--];
}
```

Linked List:

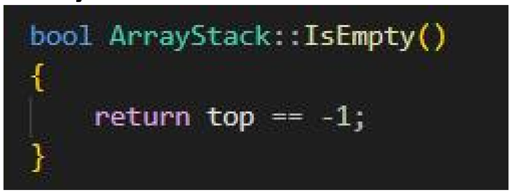


```
int LinkedStack::pop()
{
    if (!stack.IsEmpty())
    {
        int data{};
        stack.removeFromHead(data);
        return data;
    }
    throw "Empty";
}
```

(c) isEmpty():

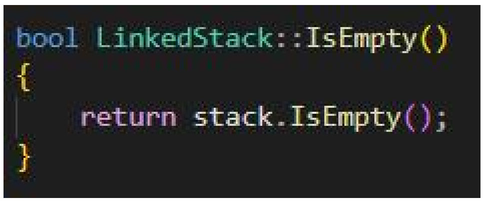
- Checks if the stack is empty

Array:



```
bool ArrayStack::IsEmpty()
{
    return top == -1;
}
```

Linked List:

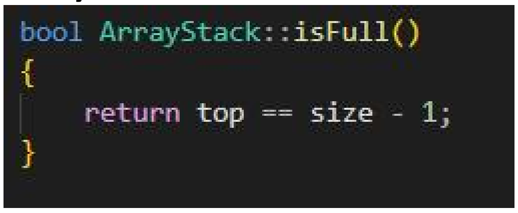


```
bool LinkedStack::IsEmpty()
{
    return stack.IsEmpty();
}
```

(d) isFull():

- Checks if the stack is full

Array:



```
bool ArrayStack::isFull()
{
    return top == size - 1;
}
```

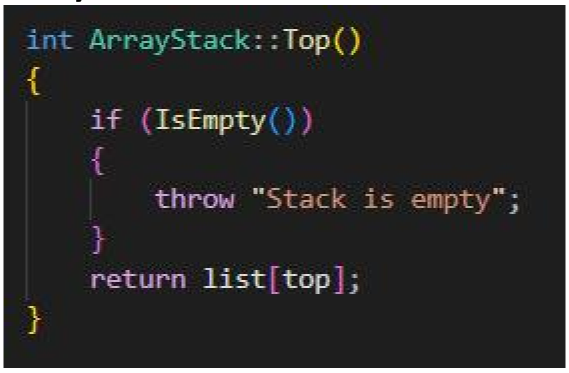
Linked List:

- Stack implementation using linked list is never full and is fully dynamic.

(e) top():

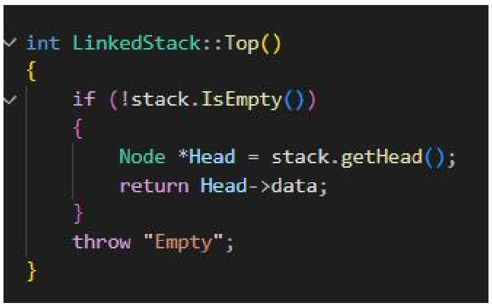
- Gives the element at the top

Array:



```
int ArrayStack::Top()
{
    if (IsEmpty())
    {
        throw "Stack is empty";
    }
    return list[top];
}
```

Linked List:



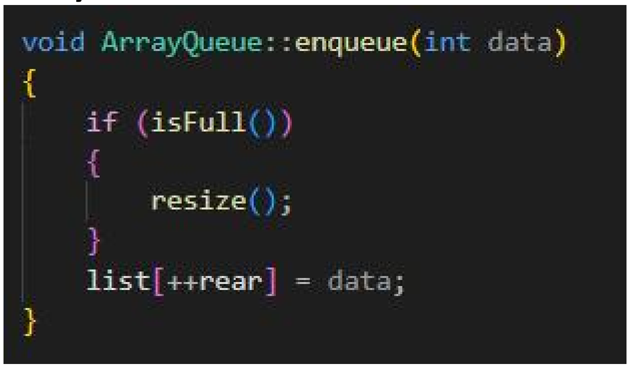
```cpp
int LinkedStack::Top()
{
    if (!stack.IsEmpty())
    {
        Node *Head = stack.getHead();
        return Head->data;
    }
    throw "Empty";
}
```

2. Implement Queue data structure using an array as well as a linked list.

(a) enqueue(element):

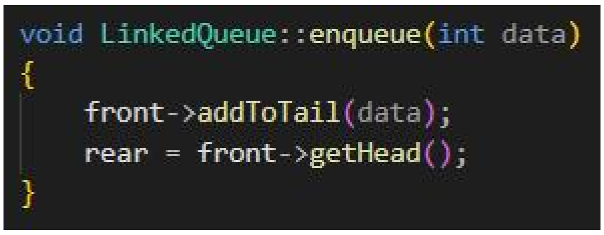
- Adds an element into the queue

Array:



```cpp
void ArrayQueue::enqueue(int data)
{
    if (isFull())
    {
        resize();
    }
    list[++rear] = data;
}
```

Linked List:

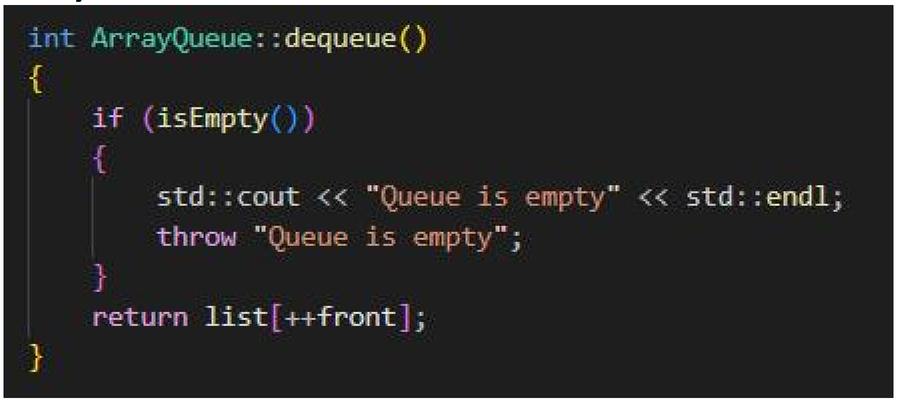


```cpp
void LinkedQueue::enqueue(int data)
{
    front->addToTail(data);
    rear = front->getHead();
}
```

(b) dequeue():

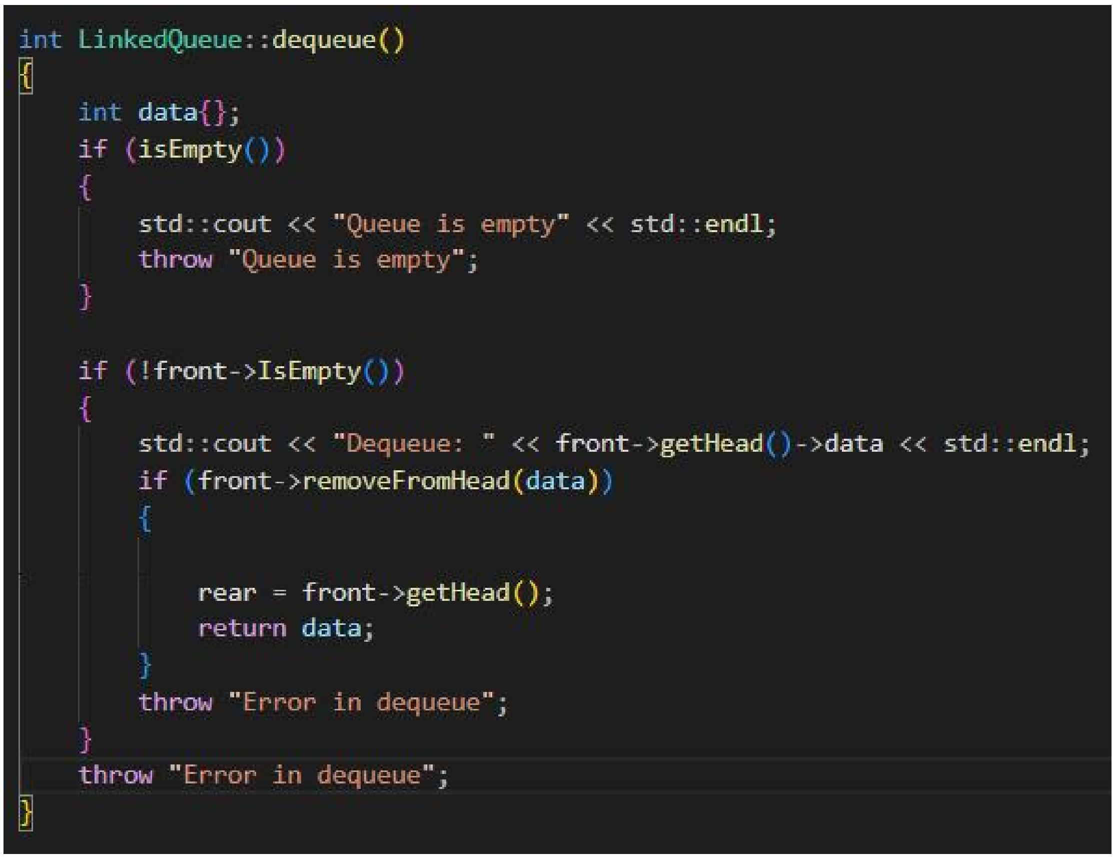
- Removes an element from the queue

Array:



```cpp
int ArrayQueue::dequeue()
{
    if (isEmpty())
    {
        std::cout << "Queue is empty" << std::endl;
        throw "Queue is empty";
    }
    return list[++front];
}
```

Linked List:

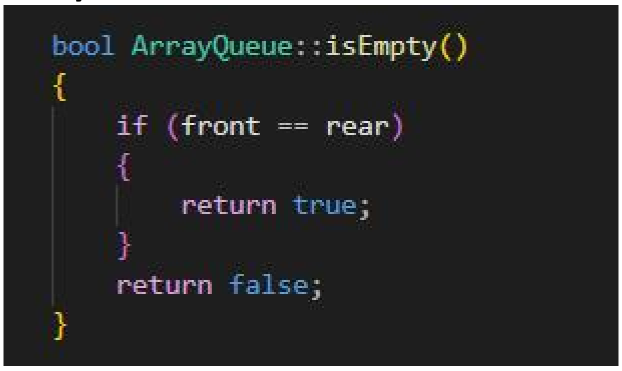


```cpp
int LinkedQueue::dequeue()
{
    int data{};
    if (isEmpty())
    {
        std::cout << "Queue is empty" << std::endl;
        throw "Queue is empty";
    }

    if (!front->IsEmpty())
    {
        std::cout << "Dequeue: " << front->getHead()->data << std::endl;
        if (front->removeFromHead(data))
        {

            rear = front->getHead();
            return data;
        }
        throw "Error in dequeue";
    }
    throw "Error in dequeue";
}
```

(c) isEmpty():

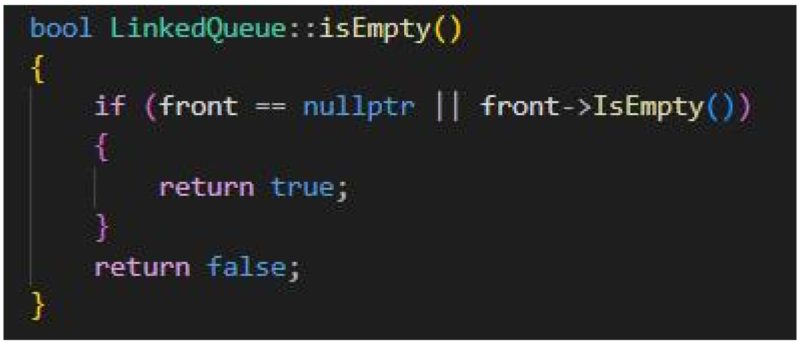
- Checks if the queue is empty

Array:



```cpp
bool ArrayQueue::isEmpty()
{
    if (front == rear)
    {
        return true;
    }
    return false;
}
```

Linked List:

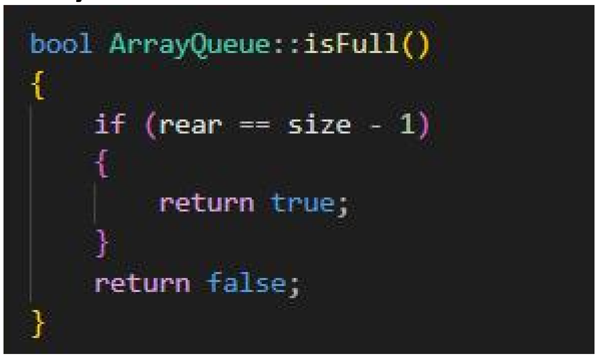


```cpp
bool LinkedQueue::isEmpty()
{
    if (front == nullptr || front->IsEmpty())
    {
        return true;
    }
    return false;
}
```

(d) isFull():

- Checks if the queue is full

Array:



```
bool ArrayQueue::isFull()
{
    if (rear == size - 1)
    {
        return true;
    }
    return false;
}
```

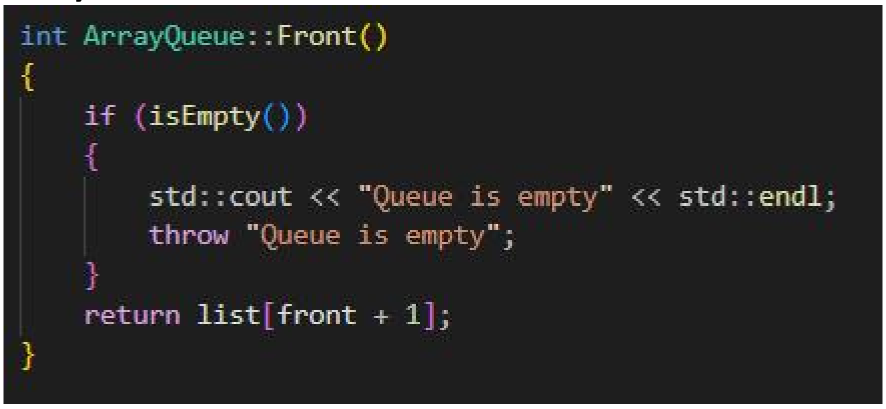
Linked List:

- Queue implemented from linked list is never full and is dynamic.

(e) front():

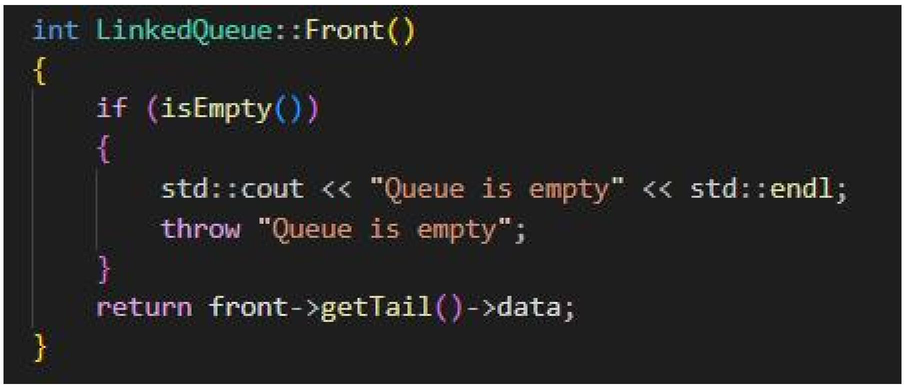
- Gives the element at the front

Array:



```
int ArrayQueue::Front()
{
    if (isEmpty())
    {
        std::cout << "Queue is empty" << std::endl;
        throw "Queue is empty";
    }
    return list[front + 1];
}
```

Linked List:

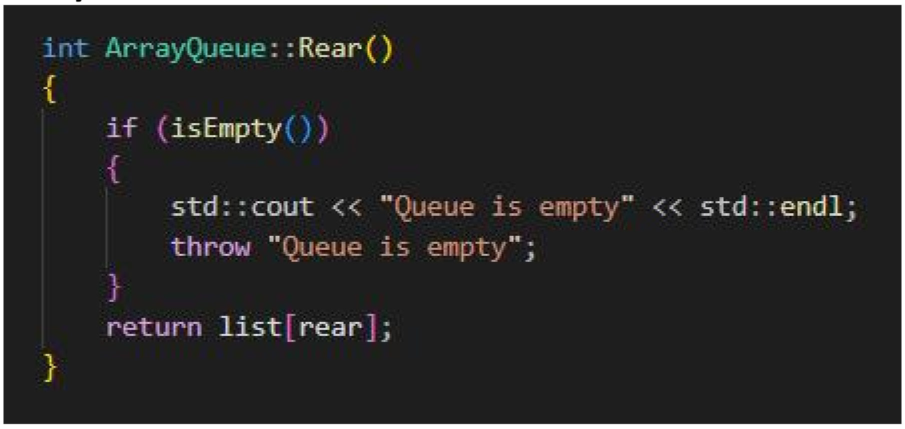


```
int LinkedQueue::Front()
{
    if (isEmpty())
    {
        std::cout << "Queue is empty" << std::endl;
        throw "Queue is empty";
    }
    return front->getTail()->data;
}
```

(f) back():

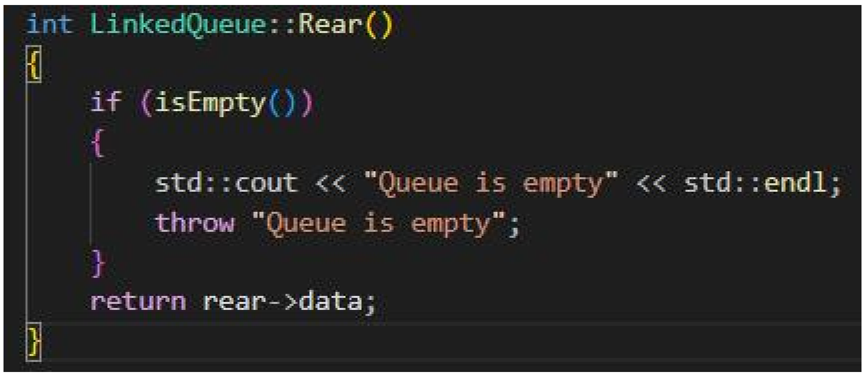
- Gives the element at the rear

Array:



```cpp
int ArrayQueue::Rear()
{
    if (isEmpty())
    {
        std::cout << "Queue is empty" << std::endl;
        throw "Queue is empty";
    }
    return list[rear];
}
```

Linked List:

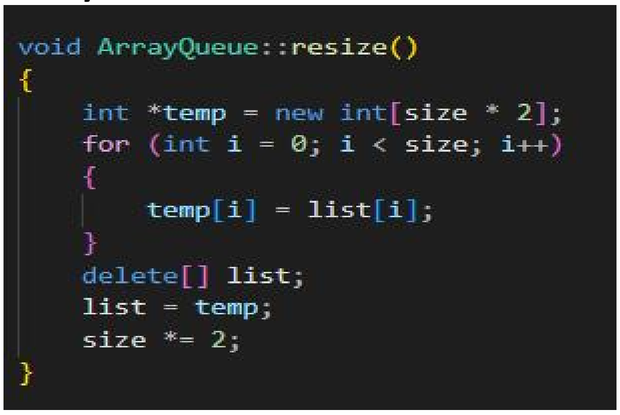


```cpp
int LinkedQueue::Rear()
{
    if (isEmpty())
    {
        std::cout << "Queue is empty" << std::endl;
        throw "Queue is empty";
    }
    return rear->data;
}
```

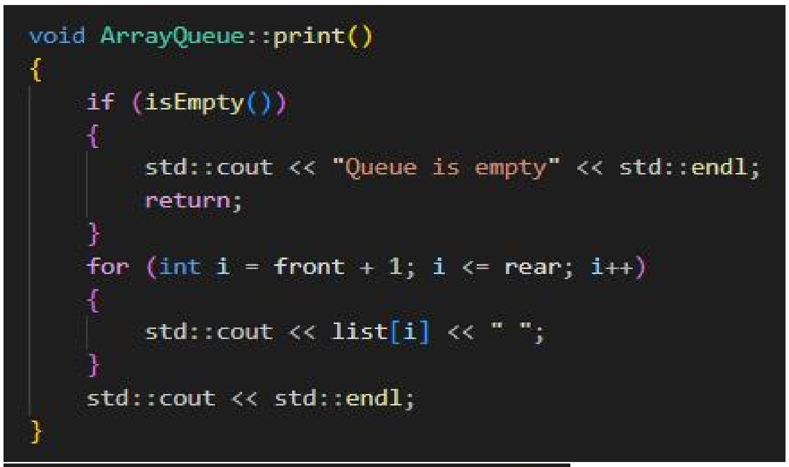
(g) Other functions:

- These are additional functions used in the code respectively for “array” and “linked list” implementations.

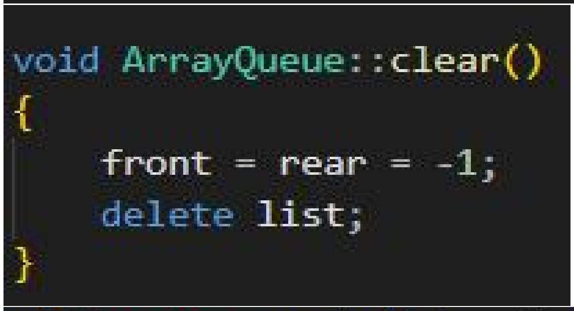
Array:



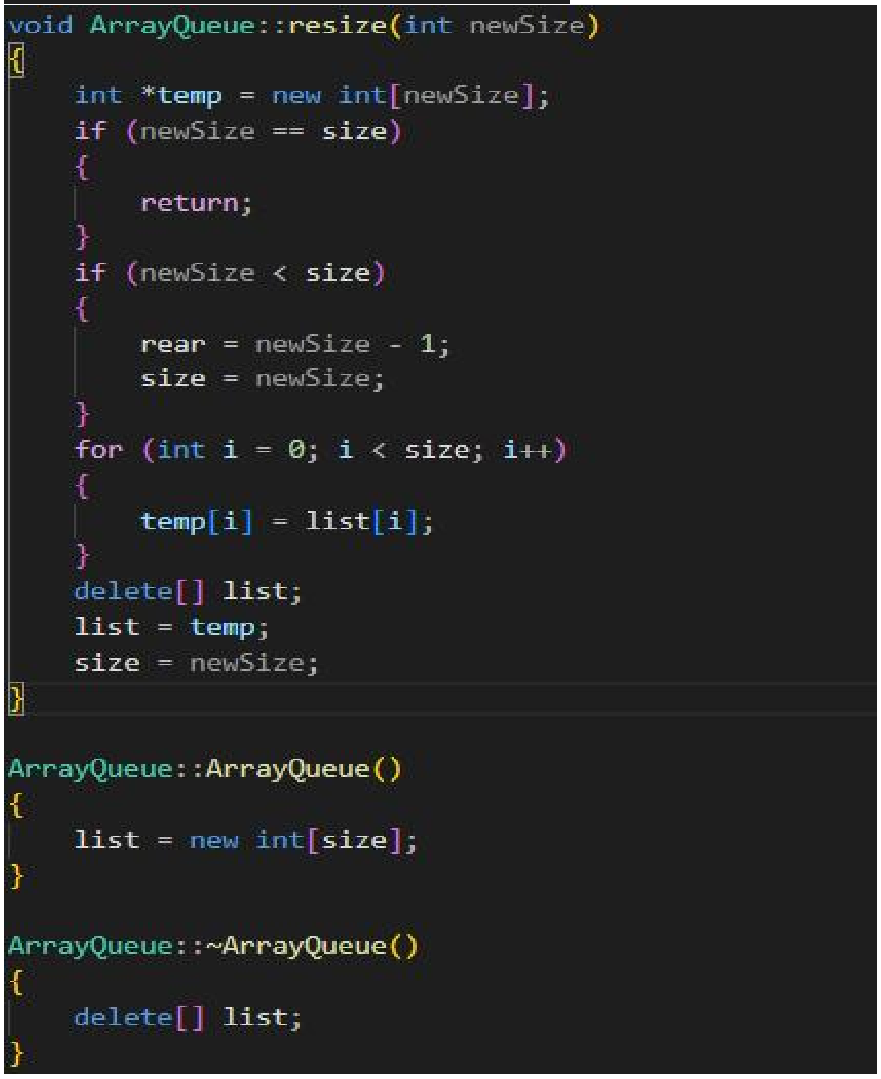
```cpp
void ArrayQueue::resize()
{
    int *temp = new int[size * 2];
    for (int i = 0; i < size; i++)
    {
        temp[i] = list[i];
    }
    delete[] list;
    list = temp;
    size *= 2;
}
```



```cpp
void ArrayQueue::print()
{
    if (isEmpty())
    {
        std::cout << "Queue is empty" << std::endl;
        return;
    }
    for (int i = front + 1; i <= rear; i++)
    {
        std::cout << list[i] << " ";
    }
    std::cout << std::endl;
}
```



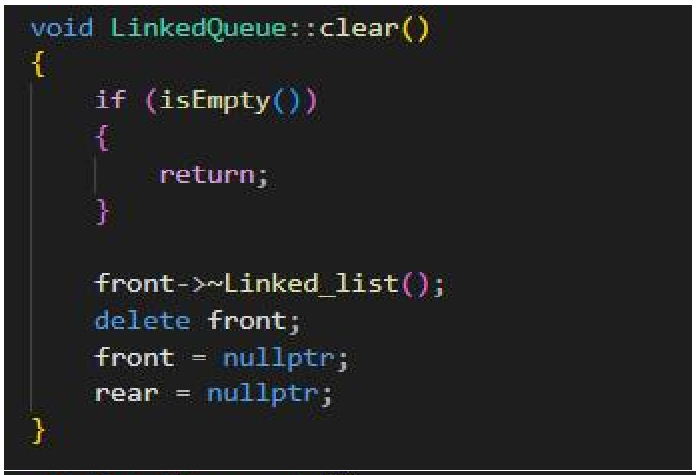
```cpp
void ArrayQueue::clear()
{
    front = rear = -1;
    delete list;
}
```



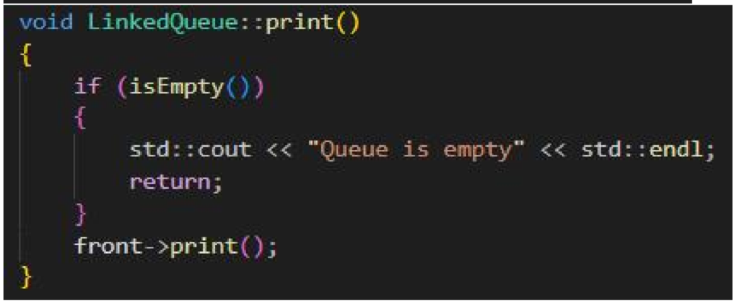
```cpp
void ArrayQueue::resize(int newSize)
{
    int *temp = new int[newSize];
    if (newSize == size)
    {
        return;
    }
    if (newSize < size)
    {
        rear = newSize - 1;
        size = newSize;
    }
    for (int i = 0; i < size; i++)
    {
        temp[i] = list[i];
    }
    delete[] list;
    list = temp;
    size = newSize;
}

ArrayQueue::ArrayQueue()
{
    list = new int[size];
}

ArrayQueue::~ArrayQueue()
{
    delete[] list;
}
```

Lab - 2 Stack and Queue

Linked List:

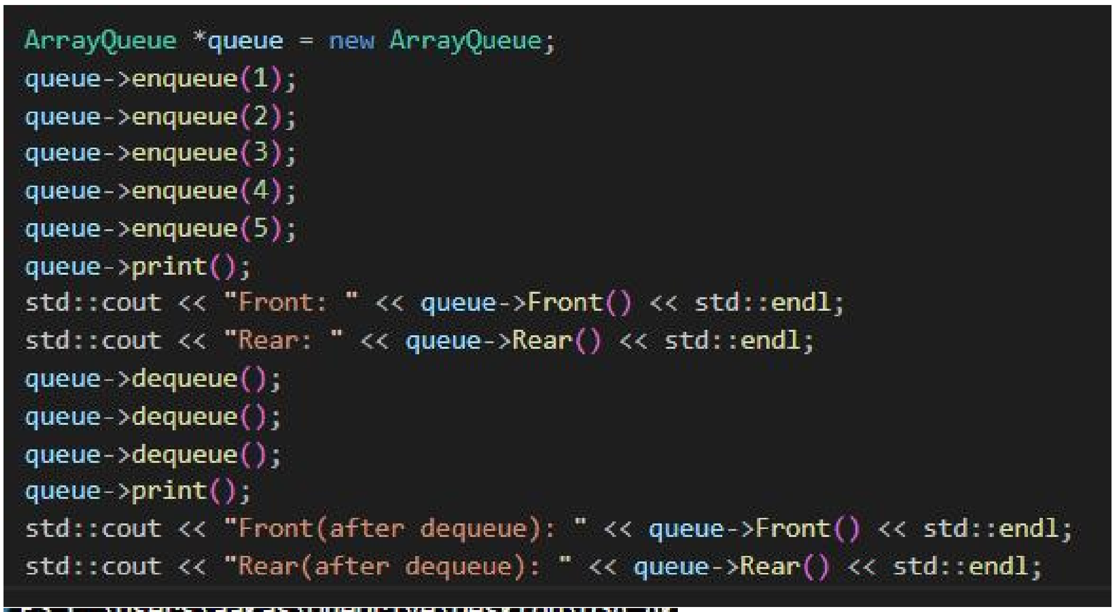


```cpp
void LinkedQueue::clear()
{
    if (isEmpty())
    {
        return;
    }

    front->~Linked_list();
    delete front;
    front = nullptr;
    rear = nullptr;
}
```

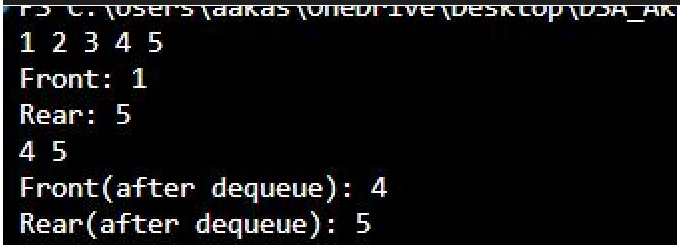


```cpp
void LinkedQueue::print()
{
    if (isEmpty())
    {
        std::cout << "Queue is empty" << std::endl;
        return;
    }
    front->print();
}
```

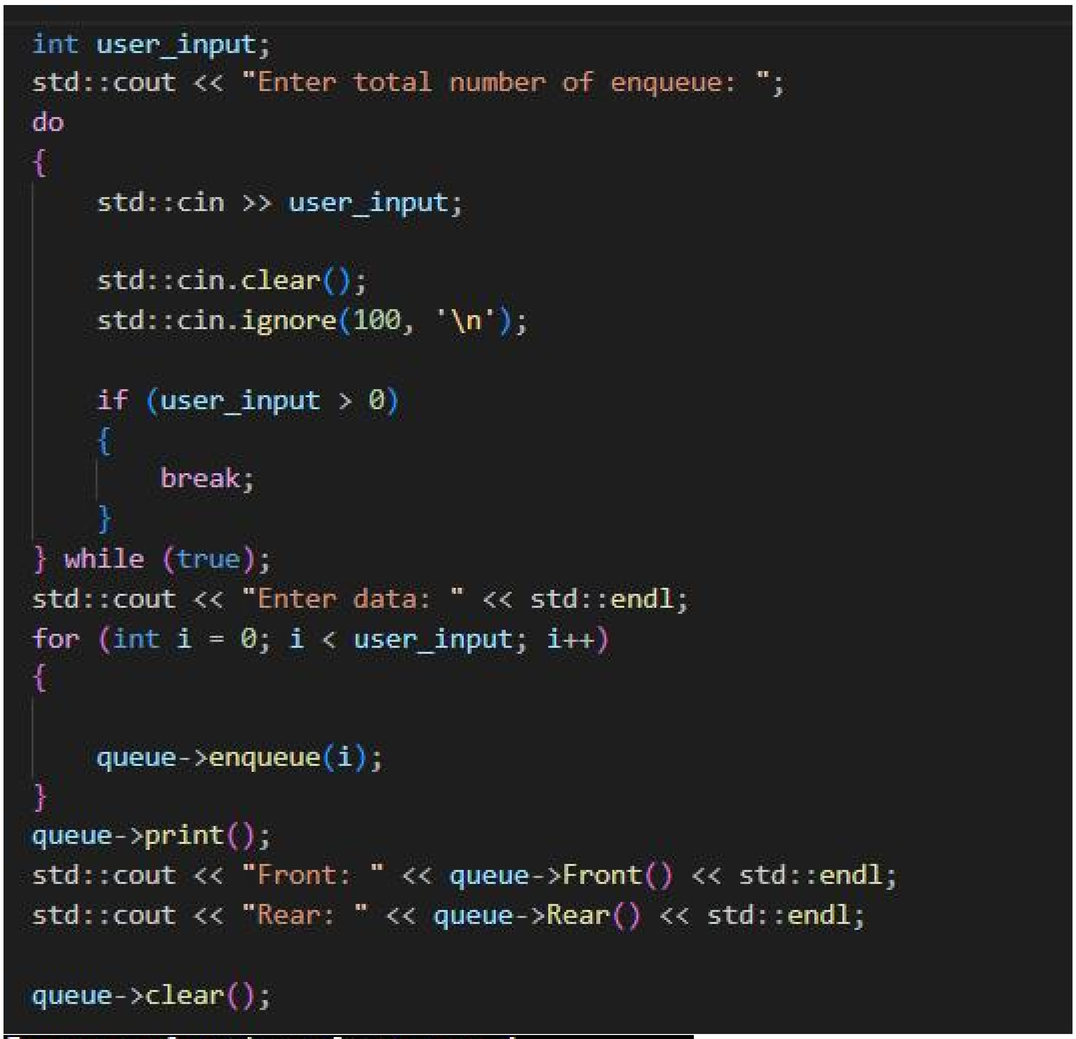
#MAIN Functions used for all:
Queue:



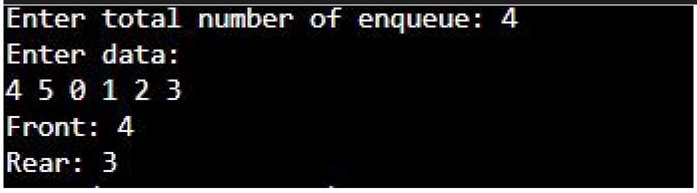
```cpp
ArrayQueue *queue = new ArrayQueue;
queue->enqueue(1);
queue->enqueue(2);
queue->enqueue(3);
queue->enqueue(4);
queue->enqueue(5);
queue->print();
std::cout << "Front: " << queue->Front() << std::endl;
std::cout << "Rear: " << queue->Rear() << std::endl;
queue->dequeue();
queue->dequeue();
queue->dequeue();
queue->print();
std::cout << "Front(after dequeue): " << queue->Front() << std::endl;
std::cout << "Rear(after dequeue): " << queue->Rear() << std::endl;
```



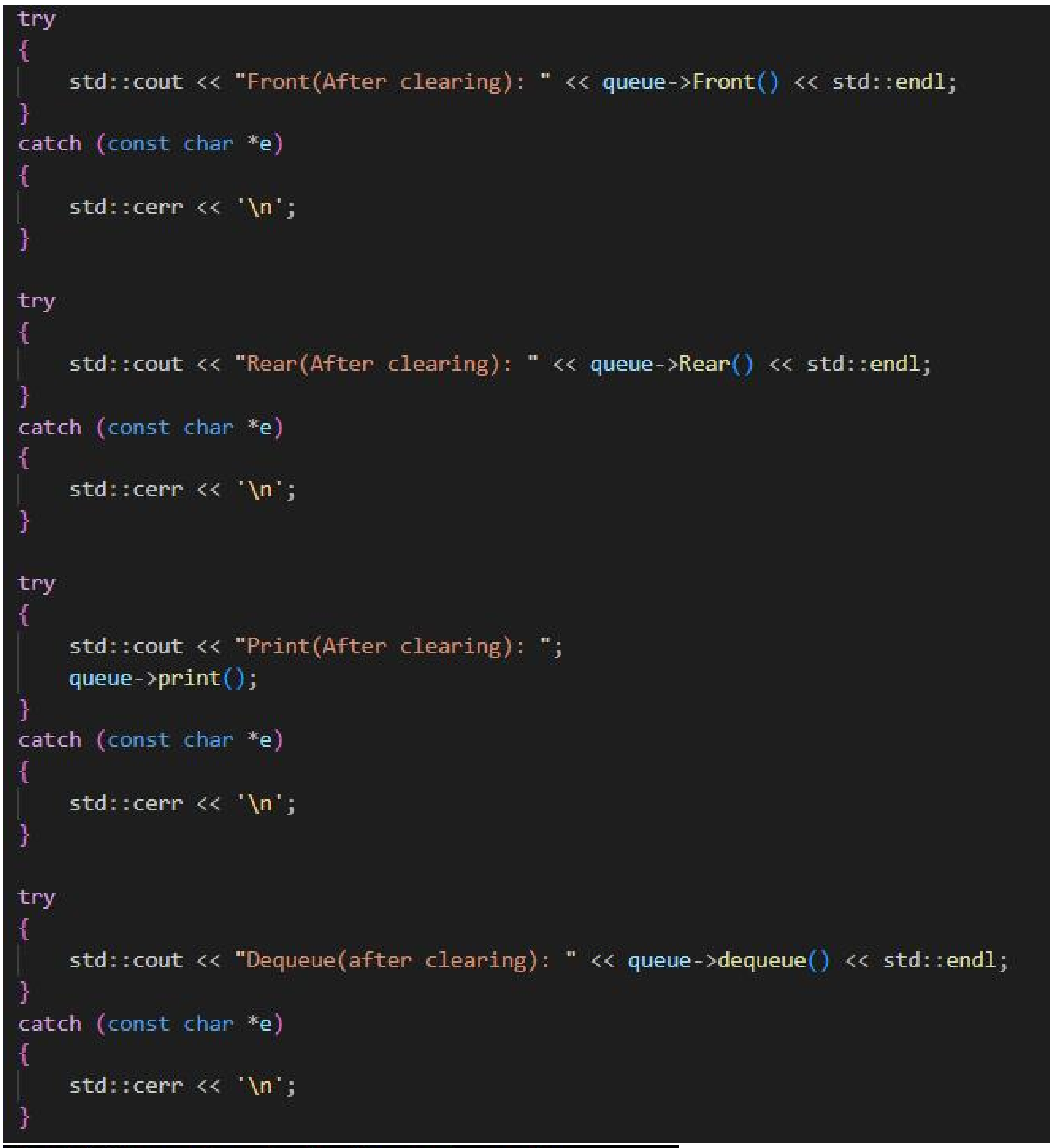
```
PS C:\Users\aakas\OneDrive\Desktop\DSA_AK
1 2 3 4 5
Front: 1
Rear: 5
4 5
Front(after dequeue): 4
Rear(after dequeue): 5
```



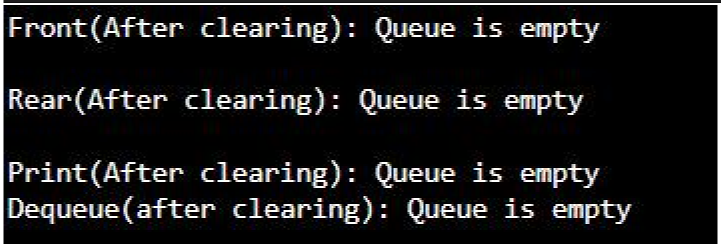
```cpp
int user_input;
std::cout << "Enter total number of enqueue: ";
do
{
    std::cin >> user_input;

    std::cin.clear();
    std::cin.ignore(100, '\n');

    if (user_input > 0)
    {
        break;
    }
} while (true);
std::cout << "Enter data: " << std::endl;
for (int i = 0; i < user_input; i++)
{

    queue->enqueue(i);
}
queue->print();
std::cout << "Front: " << queue->Front() << std::endl;
std::cout << "Rear: " << queue->Rear() << std::endl;

queue->clear();
```



```
Enter total number of enqueue: 4
Enter data:
4 5 0 1 2 3
Front: 4
Rear: 3
```



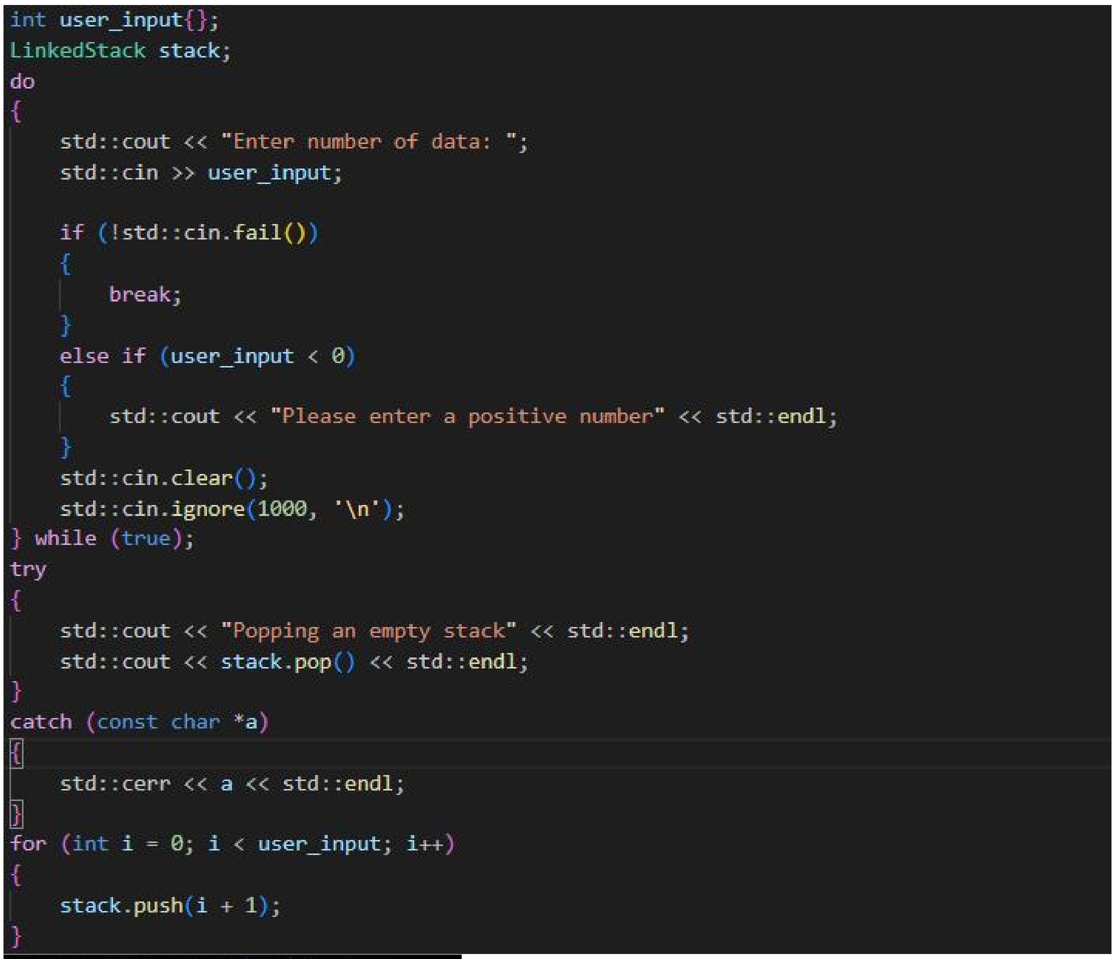
```cpp
try
{
    std::cout << "Front(After clearing): " << queue->Front() << std::endl;
}
catch (const char *e)
{
    std::cerr << '\n';
}

try
{
    std::cout << "Rear(After clearing): " << queue->Rear() << std::endl;
}
catch (const char *e)
{
    std::cerr << '\n';
}

try
{
    std::cout << "Print(After clearing): ";
    queue->print();
}
catch (const char *e)
{
    std::cerr << '\n';
}

try
{
    std::cout << "Dequeue(after clearing): " << queue->dequeue() << std::endl;
}
catch (const char *e)
{
    std::cerr << '\n';
}
```



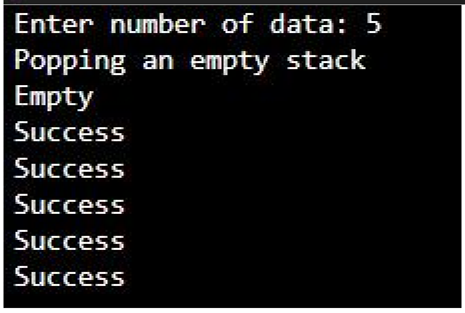
```
Front(After clearing): Queue is empty

Rear(After clearing): Queue is empty

Print(After clearing): Queue is empty
Dequeue(after clearing): Queue is empty
```

Lab - 2 Stack and Queue

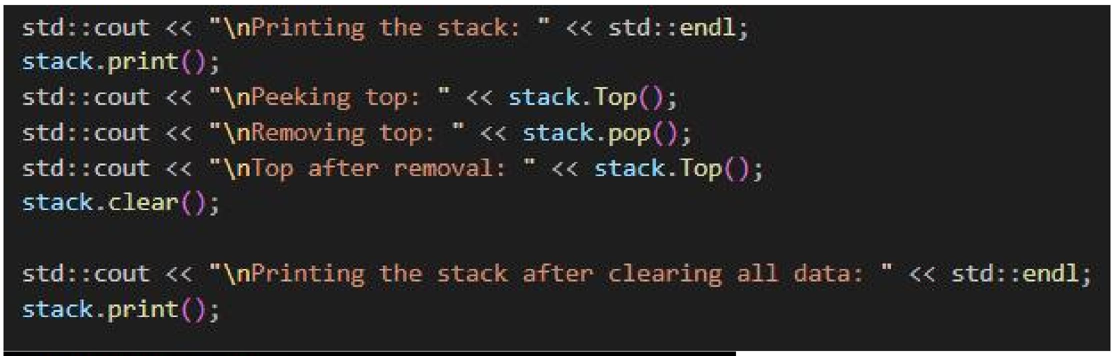
Stack:



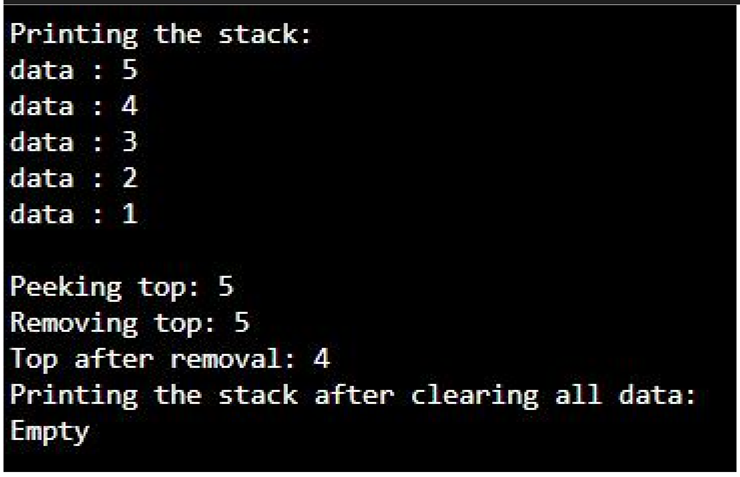
```cpp
int user_input{};
LinkedStack stack;
do
{
    std::cout << "Enter number of data: ";
    std::cin >> user_input;

    if (!std::cin.fail())
    {
        break;
    }
    else if (user_input < 0)
    {
        std::cout << "Please enter a positive number" << std::endl;
    }
    std::cin.clear();
    std::cin.ignore(1000, '\n');
} while (true);
try
{
    std::cout << "Popping an empty stack" << std::endl;
    std::cout << stack.pop() << std::endl;
}
catch (const char *a)
{
    std::cerr << a << std::endl;
}
for (int i = 0; i < user_input; i++)
{
    stack.push(i + 1);
}
```



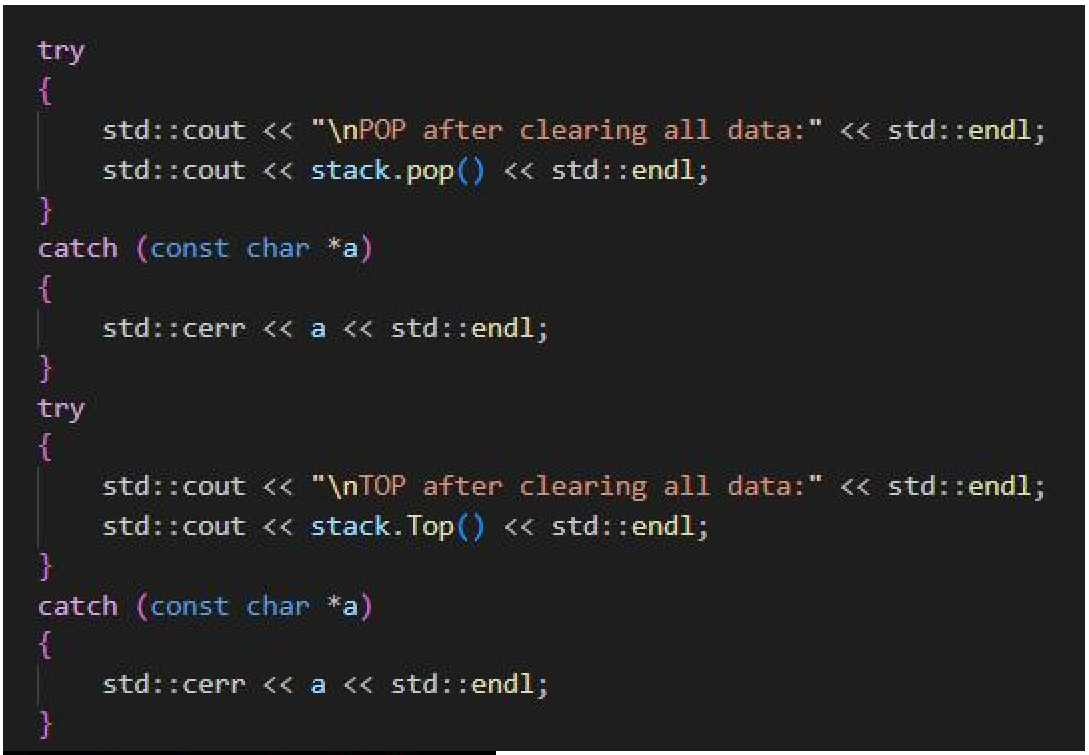
```
Enter number of data: 5
Popping an empty stack
Empty
Success
Success
Success
Success
Success
```



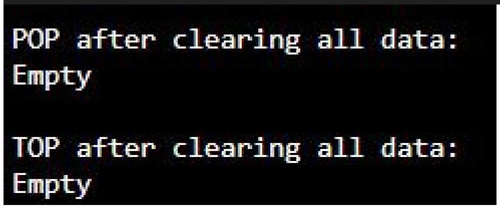
```cpp
std::cout << "\nPrinting the stack: " << std::endl;
stack.print();
std::cout << "\nPeeking top: " << stack.Top();
std::cout << "\nRemoving top: " << stack.pop();
std::cout << "\nTop after removal: " << stack.Top();
stack.clear();

std::cout << "\nPrinting the stack after clearing all data: " << std::endl;
stack.print();
```



```
Printing the stack:
data : 5
data : 4
data : 3
data : 2
data : 1

Peeking top: 5
Removing top: 5
Top after removal: 4
Printing the stack after clearing all data:
Empty
```



```cpp
try
{
    std::cout << "\nPOP after clearing all data:" << std::endl;
    std::cout << stack.pop() << std::endl;
}
catch (const char *a)
{
    std::cerr << a << std::endl;
}
try
{
    std::cout << "\nTOP after clearing all data:" << std::endl;
    std::cout << stack.Top() << std::endl;
}
catch (const char *a)
{
    std::cerr << a << std::endl;
}
```



```
POP after clearing all data:
Empty

TOP after clearing all data:
Empty
```