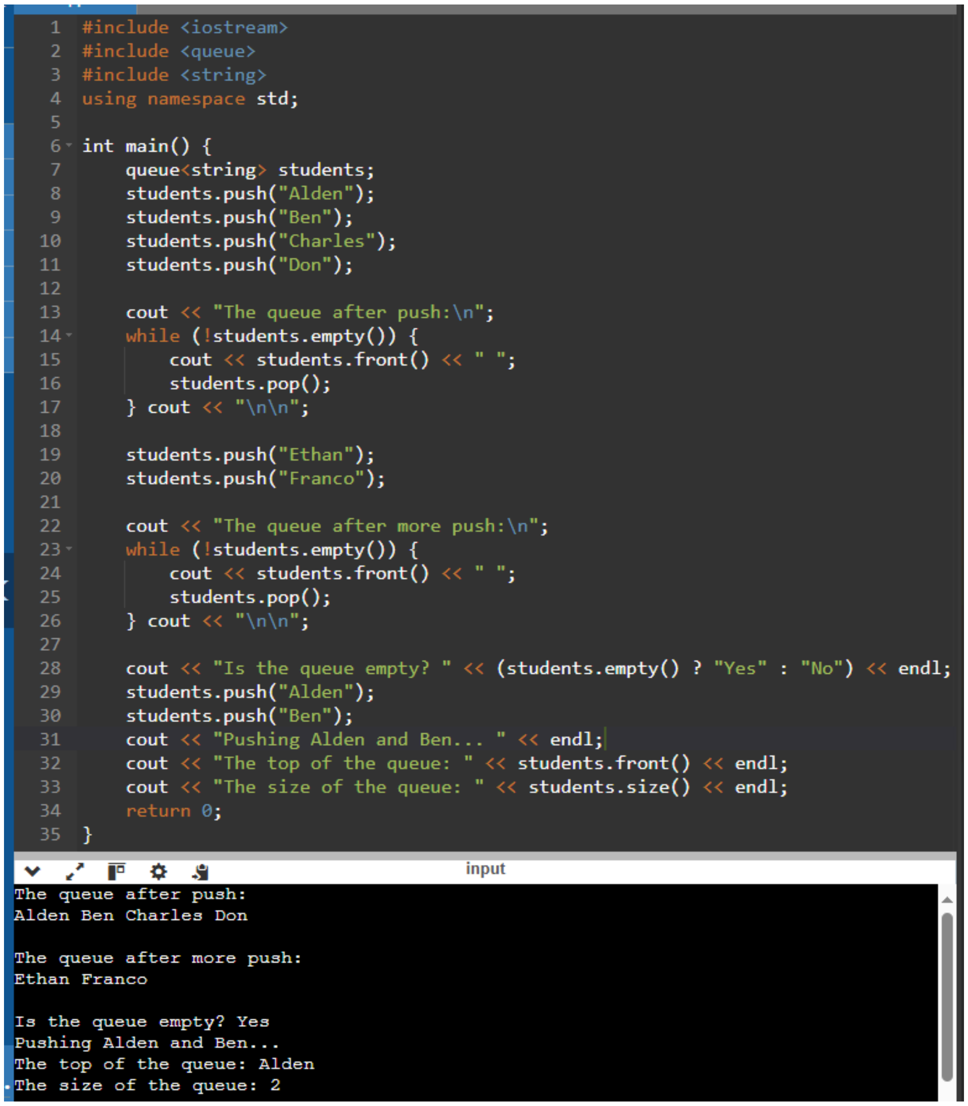
Activity No. 3	
Hands-on Activity 5.1 Queues	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** October 07, 2024
Section: CPE21S4	**Date Submitted:** October 08, 2024
Name(s): Don Eleazar T. Fernandez	**Instructor:** Maria Rizette Sayo

6. Output

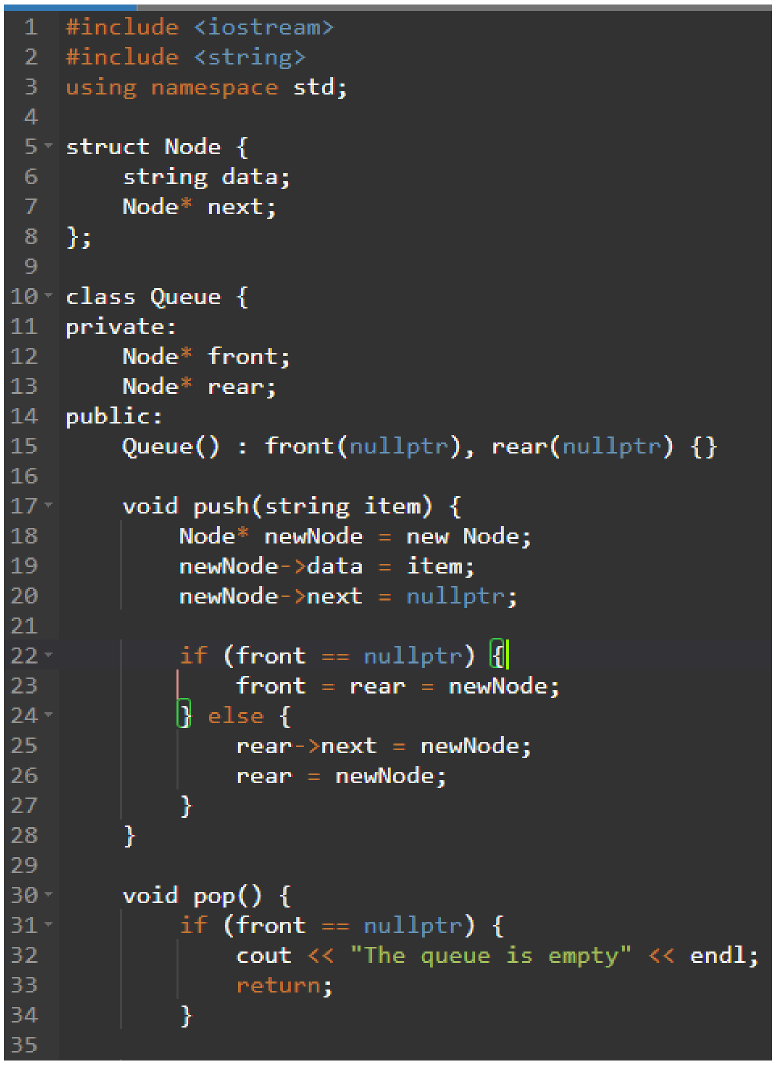


```cpp
#include <iostream>
#include <queue>
#include <string>
using namespace std;

int main() {
    queue<string> students;
    students.push("Alden");
    students.push("Ben");
    students.push("Charles");
    students.push("Don");

    cout << "The queue after push:\n";
    while (!students.empty()) {
        cout << students.front() << " ";
        students.pop();
    } cout << "\n\n";

    students.push("Ethan");
    students.push("Franco");

    cout << "The queue after more push:\n";
    while (!students.empty()) {
        cout << students.front() << " ";
        students.pop();
    } cout << "\n\n";

    cout << "Is the queue empty? " << (students.empty() ? "Yes" : "No") << endl;
    students.push("Alden");
    students.push("Ben");
    cout << "Pushing Alden and Ben... " << endl;
    cout << "The top of the queue: " << students.front() << endl;
    cout << "The size of the queue: " << students.size() << endl;
    return 0;
}
```

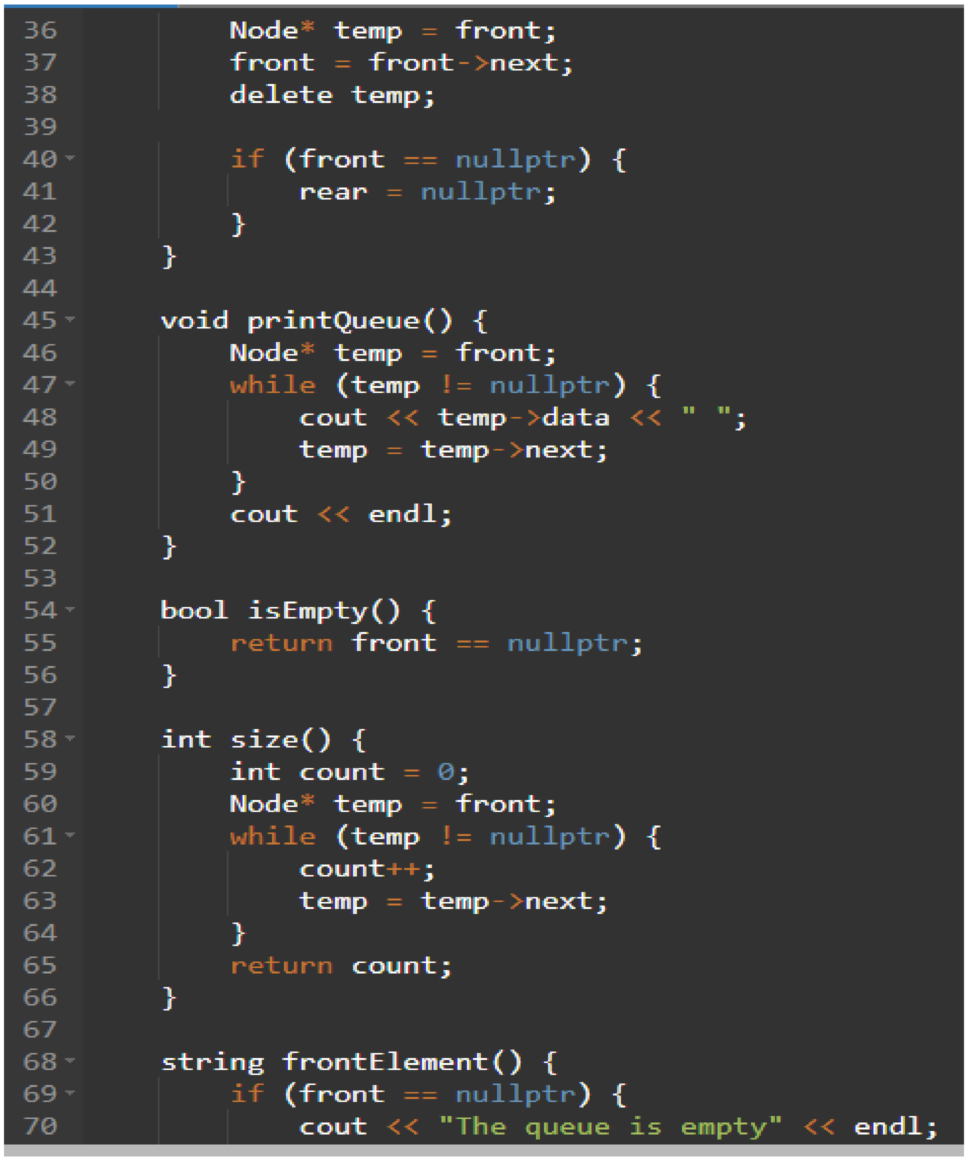
```
The queue after push:
Alden Ben Charles Don

The queue after more push:
Ethan Franco

Is the queue empty? Yes
Pushing Alden and Ben...
The top of the queue: Alden
The size of the queue: 2
```

Observation: The operations used in the queue were .pop(), .push(), .front(), .size(), and .empty(). The pop was used to remove data from the queue. The push was used to add data to the queue. The front was used to output the front of the queue line up. The size was used to determine the number of the data on the queue. The empty was used to clear out the whole data from the queue.

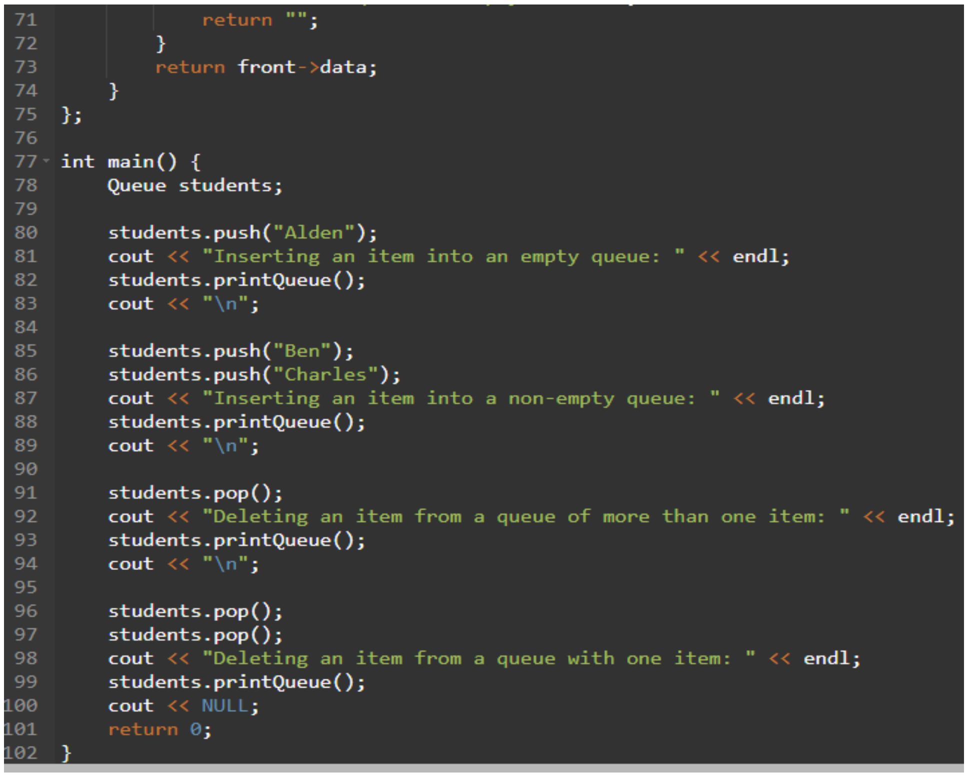
Table 5 - 1. Queues using C++ STL



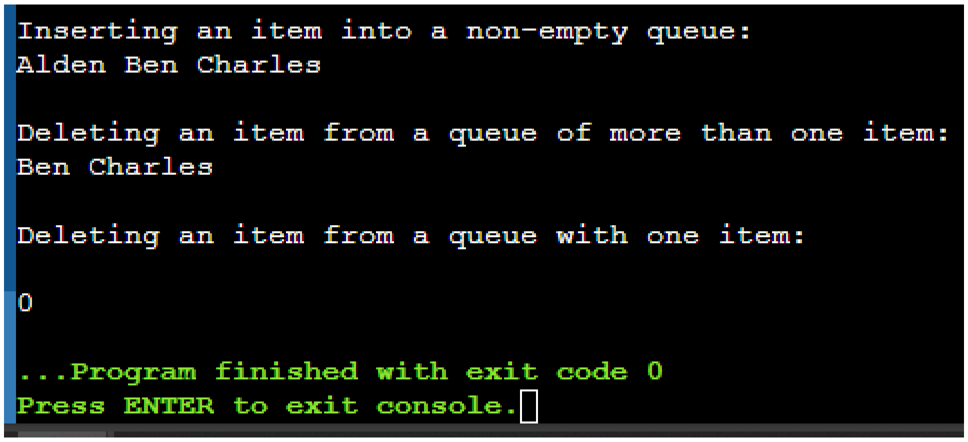
```cpp
#include <iostream>
#include <string>
using namespace std;

struct Node {
    string data;
    Node* next;
};

class Queue {
private:
    Node* front;
    Node* rear;
public:
    Queue() : front(nullptr), rear(nullptr) {}

    void push(string item) {
        Node* newNode = new Node;
        newNode->data = item;
        newNode->next = nullptr;

        if (front == nullptr) {
            front = rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
    }

    void pop() {
        if (front == nullptr) {
            cout << "The queue is empty" << endl;
            return;
        }

```



```
        Node* temp = front;
        front = front->next;
        delete temp;

        if (front == nullptr) {
            rear = nullptr;
        }
    }

    void printQueue() {
        Node* temp = front;
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }

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

    int size() {
        int count = 0;
        Node* temp = front;
        while (temp != nullptr) {
            count++;
            temp = temp->next;
        }
        return count;
    }

    string frontElement() {
        if (front == nullptr) {
            cout << "The queue is empty" << endl;
```

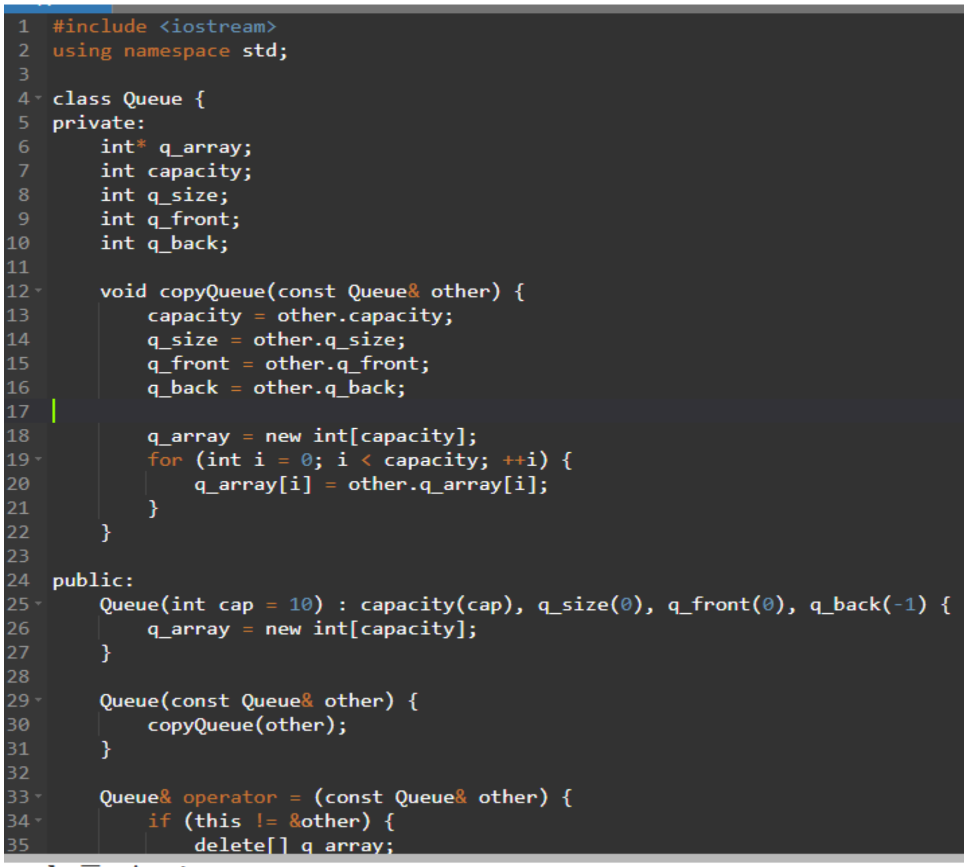


```
            return "";
        }
        return front->data;
    }
};

int main() {
    Queue students;

    students.push("Alden");
    cout << "Inserting an item into an empty queue: " << endl;
    students.printQueue();
    cout << "\n";

    students.push("Ben");
    students.push("Charles");
    cout << "Inserting an item into a non-empty queue: " << endl;
    students.printQueue();
    cout << "\n";

    students.pop();
    cout << "Deleting an item from a queue of more than one item: " << endl;
    students.printQueue();
    cout << "\n";

    students.pop();
    students.pop();
    cout << "Deleting an item from a queue with one item: " << endl;
    students.printQueue();
    cout << NULL;
    return 0;
}
```

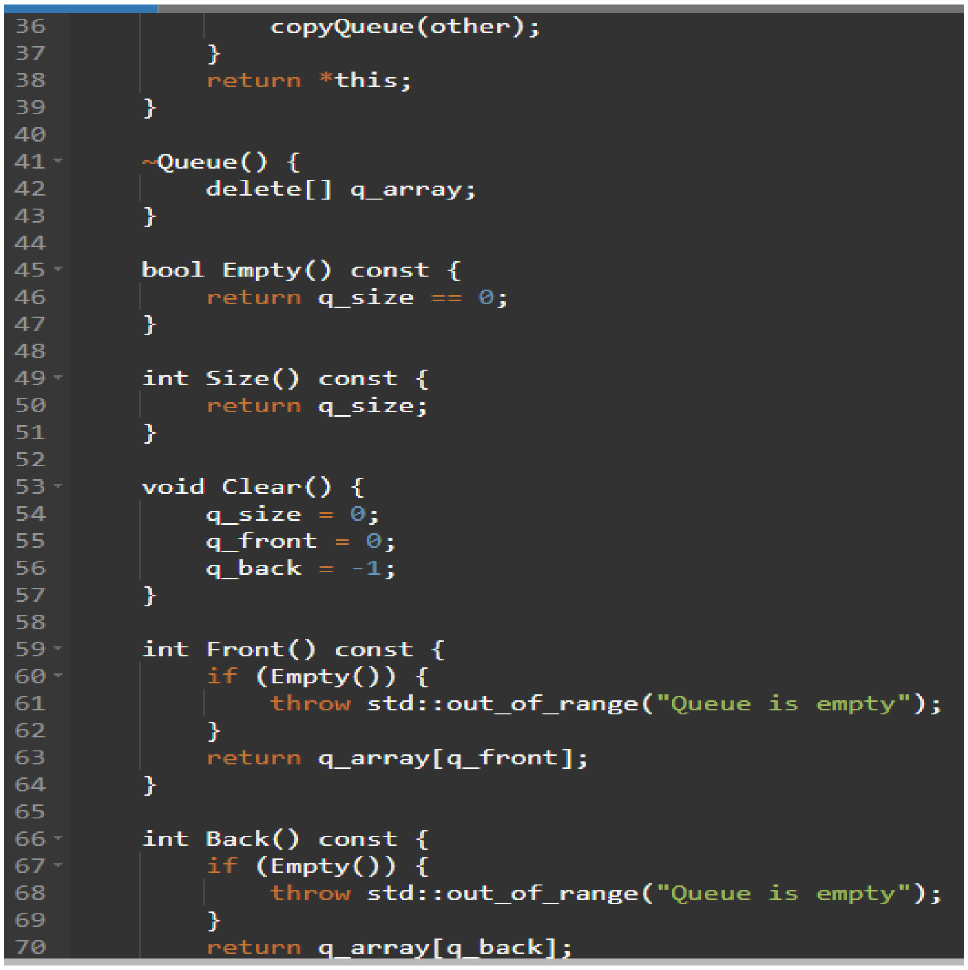


```
Inserting an item into a non-empty queue:
Alden Ben Charles

Deleting an item from a queue of more than one item:
Ben Charles

Deleting an item from a queue with one item:

0

...Program finished with exit code 0
Press ENTER to exit console.
```

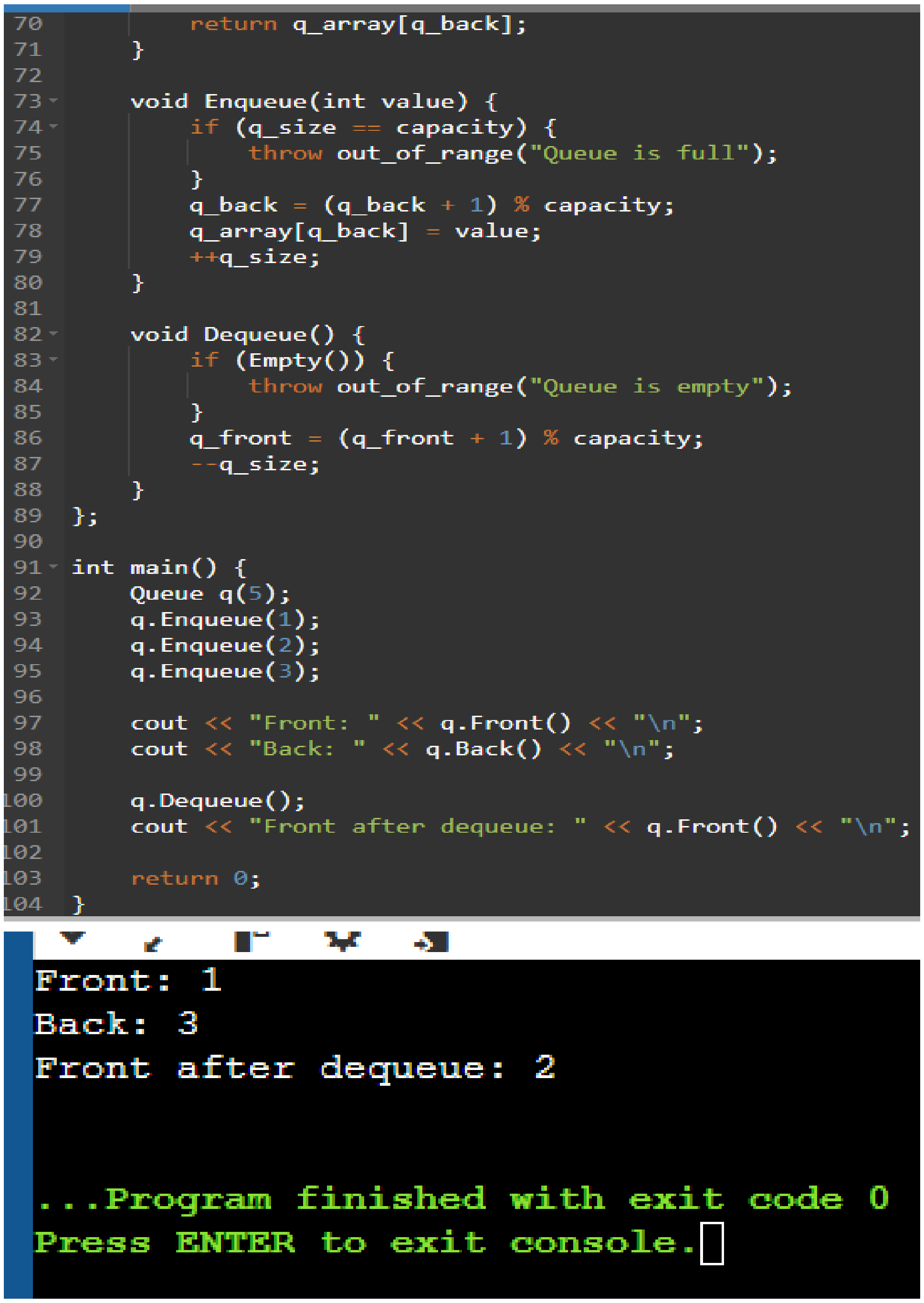
Table 5 - 2. Queues using Linked List Implementation



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

class Queue {
private:
    int* q_array;
    int capacity;
    int q_size;
    int q_front;
    int q_back;

    void copyQueue(const Queue& other) {
        capacity = other.capacity;
        q_size = other.q_size;
        q_front = other.q_front;
        q_back = other.q_back;

        q_array = new int[capacity];
        for (int i = 0; i < capacity; ++i) {
            q_array[i] = other.q_array[i];
        }
    }

public:
    Queue(int cap = 10) : capacity(cap), q_size(0), q_front(0), q_back(-1) {
        q_array = new int[capacity];
    }

    Queue(const Queue& other) {
        copyQueue(other);
    }

    Queue& operator = (const Queue& other) {
        if (this != &other) {
            delete[] q_array;
```



```cpp
            copyQueue(other);
        }
        return *this;
    }

    ~Queue() {
        delete[] q_array;
    }

    bool Empty() const {
        return q_size == 0;
    }

    int Size() const {
        return q_size;
    }

    void Clear() {
        q_size = 0;
        q_front = 0;
        q_back = -1;
    }

    int Front() const {
        if (Empty()) {
            throw std::out_of_range("Queue is empty");
        }
        return q_array[q_front];
    }

    int Back() const {
        if (Empty()) {
            throw std::out_of_range("Queue is empty");
        }
        return q_array[q_back];
```



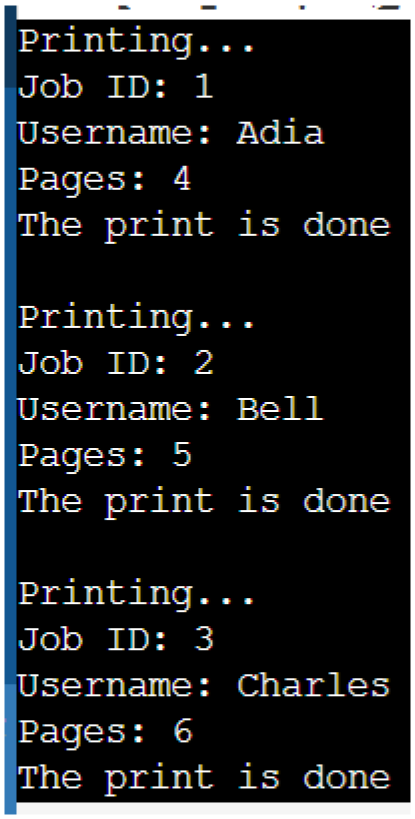
```cpp
        return q_array[q_back];
    }

    void Enqueue(int value) {
        if (q_size == capacity) {
            throw out_of_range("Queue is full");
        }
        q_back = (q_back + 1) % capacity;
        q_array[q_back] = value;
        ++q_size;
    }

    void Dequeue() {
        if (Empty()) {
            throw out_of_range("Queue is empty");
        }
        q_front = (q_front + 1) % capacity;
        --q_size;
    }
};

int main() {
    Queue q(5);
    q.Enqueue(1);
    q.Enqueue(2);
    q.Enqueue(3);

    cout << "Front: " << q.Front() << "\n";
    cout << "Back: " << q.Back() << "\n";

    q.Dequeue();
    cout << "Front after dequeue: " << q.Front() << "\n";

    return 0;
}
```

```
Front: 1
Back: 3
Front after dequeue: 2


...Program finished with exit code 0
Press ENTER to exit console.
```

Table 5 - 3. Queues using Array Implementation

7. Supplementary Activity

```cpp
#include <iostream>
#include <string>
#include <vector>
using namespace std;

class Job {
public:
    int id;
    string username;
    int pages;

    Job(int id, string username, int pages) {
        this->id = id;
        this->username = username;
        this->pages = pages;
    }

    void displayJob() {
        cout << "Job ID: " << id << endl;
        cout << "Username: " << username << endl;
        cout << "Pages: " << pages << endl;
    }
};

class Printer {
private:
    vector<Job> jobs;
    int capacity;

public:
    Printer(int capacity) {
        this->capacity = capacity;
    }

    void addJob(Job job) {
        if (jobs.size() >= capacity) {
            cout << "The queue is full" << endl;
            return;
        }
        jobs.push_back(job);
    }

    void processJobs() {
        while (!jobs.empty()) {
            Job job = jobs.front();
            cout << "Printing..." << endl;
            job.displayJob();
            cout << "The print is done" << endl;
            cout << endl;
            jobs.erase(jobs.begin());
        }
    }
};

int main() {
    Printer printer(5);

    printer.addJob(Job(1, "Adia", 4));
    printer.addJob(Job(2, "Bell", 5));
    printer.addJob(Job(3, "Charles", 6));
    printer.processJobs();
    return 0;
}
```



```
Printing...
Job ID: 1
Username: Adia
Pages: 4
The print is done

Printing...
Job ID: 2
Username: Bell
Pages: 5
The print is done

Printing...
Job ID: 3
Username: Charles
Pages: 6
The print is done
```

Defend your choice of internal representation: Why did you use arrays or linked list?

- I chose a linked list because it has an indefinite size, and it can easily insert and delete elements.

Output Analysis:

- The output displays that the jobs are processed in the same order they were added, which is the fundamental concept of a queue, the "first in and first out". And, each details of the job, including the ID, username, and number of pages, were printed correctly.

8. Conclusion

In this activity, I learned how to use different queue operations like .pop(), .push(), .front(), .size(), and .empty(). The implementation of linked lists and arrays on queues helped me understand how queues work and how to manage data in an organized way. And, a simple printer queue program was done in this activity, where jobs are added to a printer with a indefinite capacity and processed in the order they were received which is the fundamental concept of a queue, the "first in and first out". In my opinion, I did well enough in this activity, however, there are times that I mix all up the program key functions.

9. Assessment Rubric

I affirm that I will not give or receive any unauthorized help on this activity/exam and that all work will be my own.