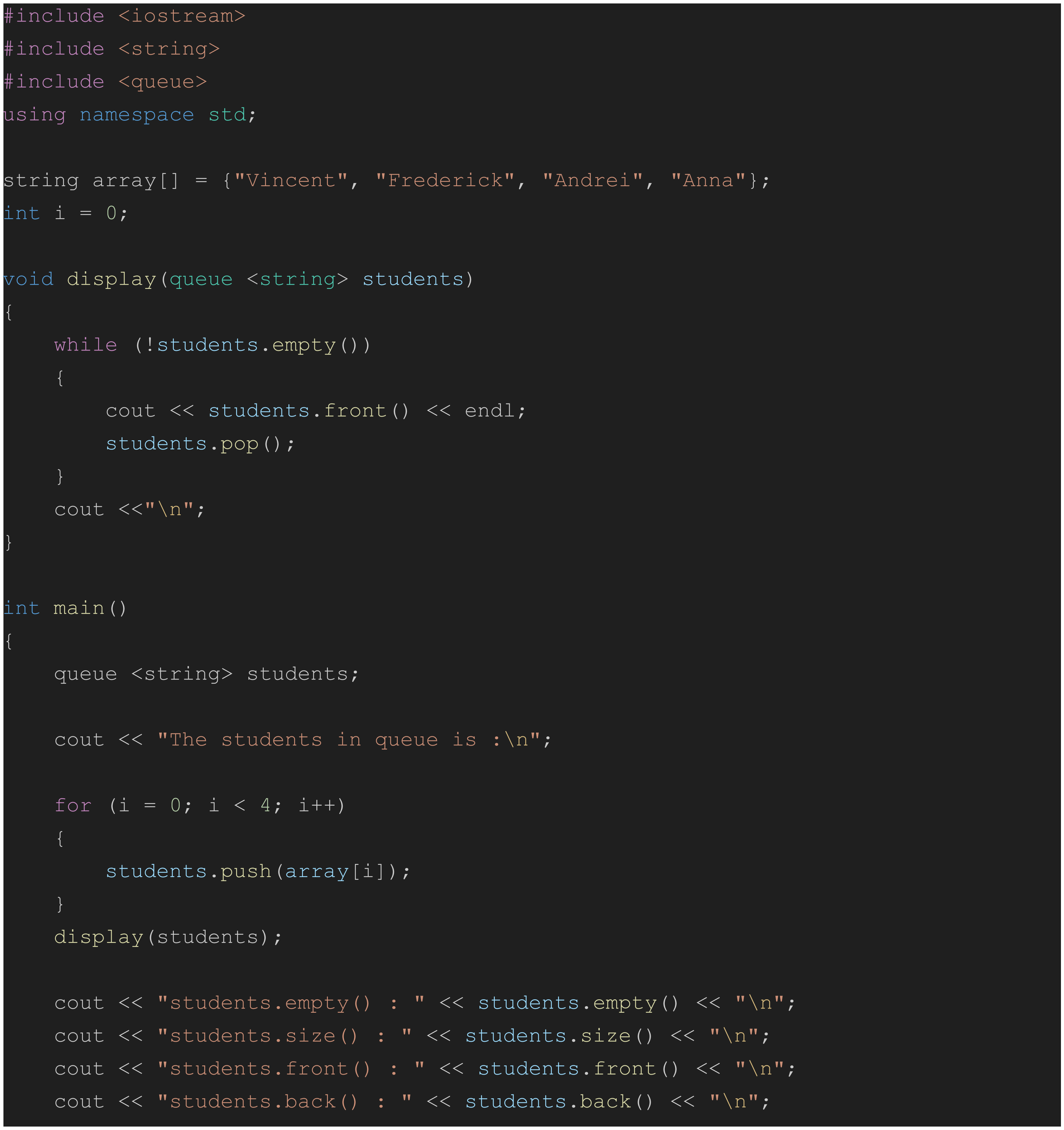
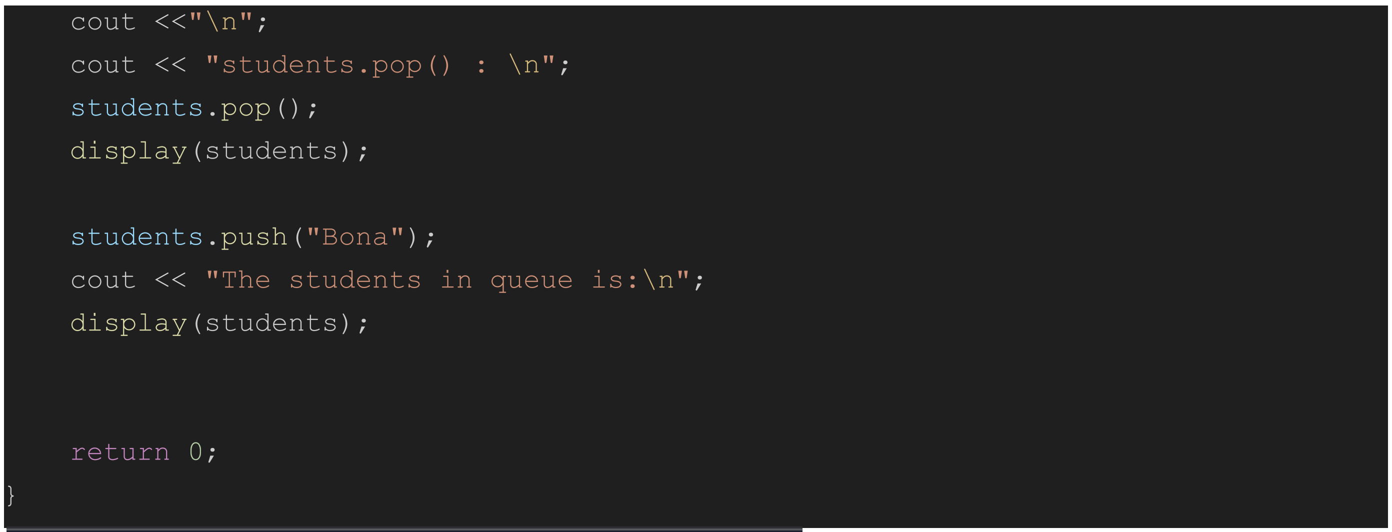
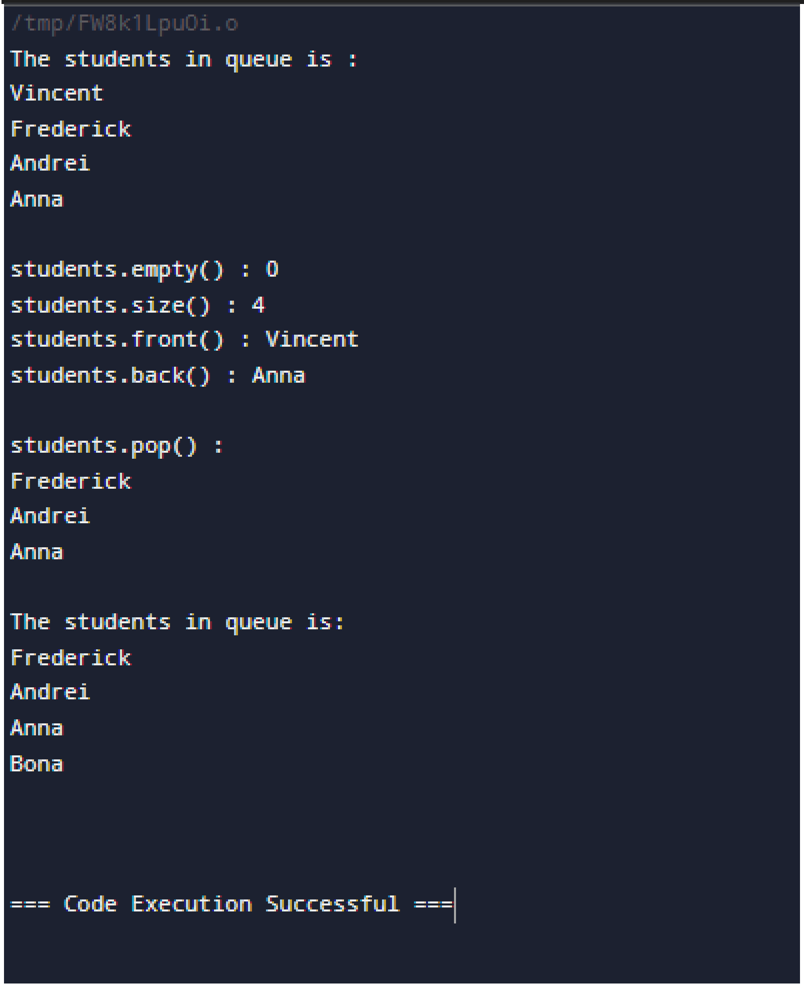
Activity No. 5	
Queues	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** October 7, 2024
Section: CPE21S4	**Date Submitted:** October 8, 2024
Name(s): Bona, Andrei Nycole So	**Instructor:** Prof. Maria Rizette Sayo
6. Output	



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

string array[] = {"Vincent", "Frederick", "Andrei", "Anna"};
int i = 0;

void display(queue <string> students)
{
    while (!students.empty())
    {
        cout << students.front() << endl;
        students.pop();
    }
    cout <<"\n";
}

int main()
{
    queue <string> students;

    cout << "The students in queue is :\n";

    for (i = 0; i < 4; i++)
    {
        students.push(array[i]);
    }
    display(students);

    cout << "students.empty() : " << students.empty() << "\n";
    cout << "students.size() : " << students.size() << "\n";
    cout << "students.front() : " << students.front() << "\n";
    cout << "students.back() : " << students.back() << "\n";
```

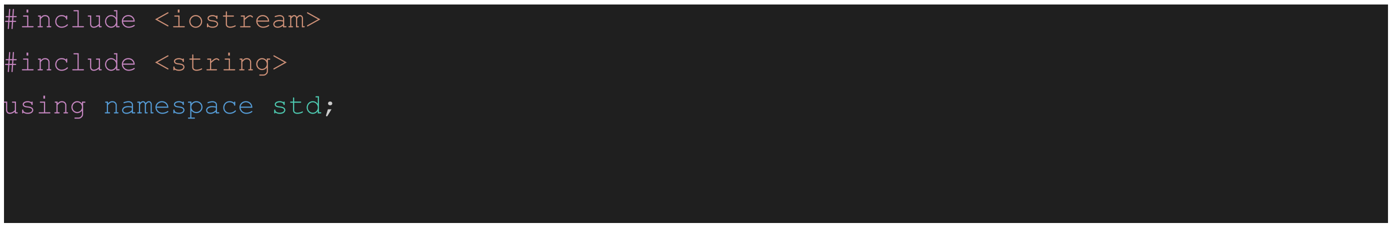


```cpp
    cout <<"\n";
    cout << "students.pop() : \n";
    students.pop();
    display(students);

    students.push("Bona");
    cout << "The students in queue is:\n";
    display(students);


    return 0;
}
```



```
/tmp/FW8k1Lpu0i.o
The students in queue is :
Vincent
Frederick
Andrei
Anna

students.empty() : 0
students.size() : 4
students.front() : Vincent
students.back() : Anna

students.pop() :
Frederick
Andrei
Anna

The students in queue is:
Frederick
Andrei
Anna
Bona



=== Code Execution Successful ===
```

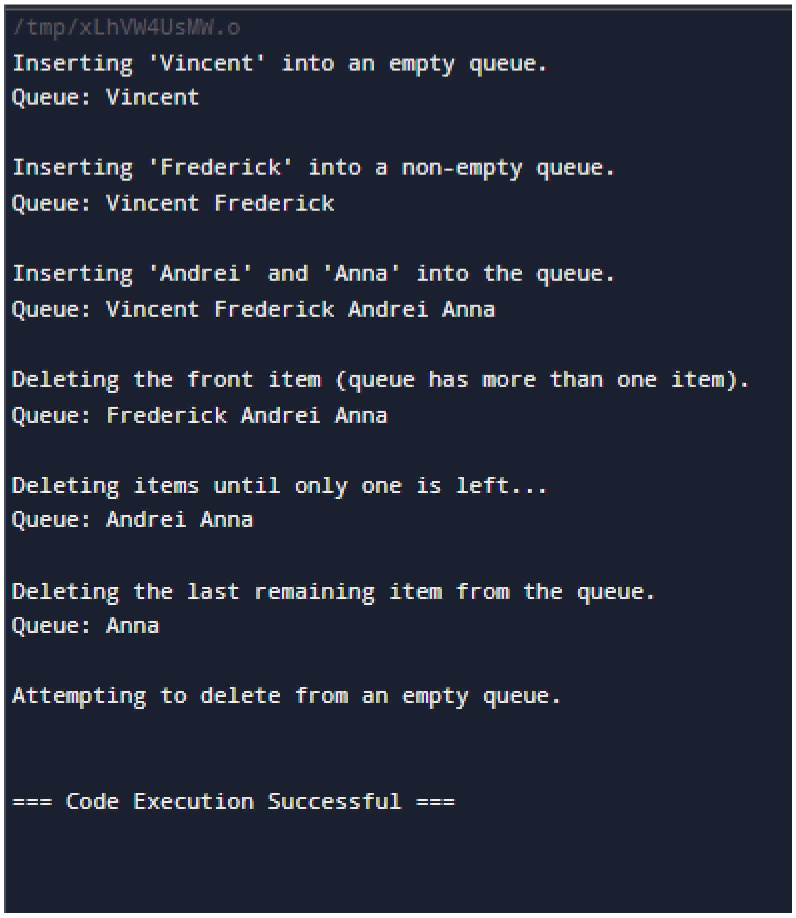
Table 5-1. Queues using C++ STL



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

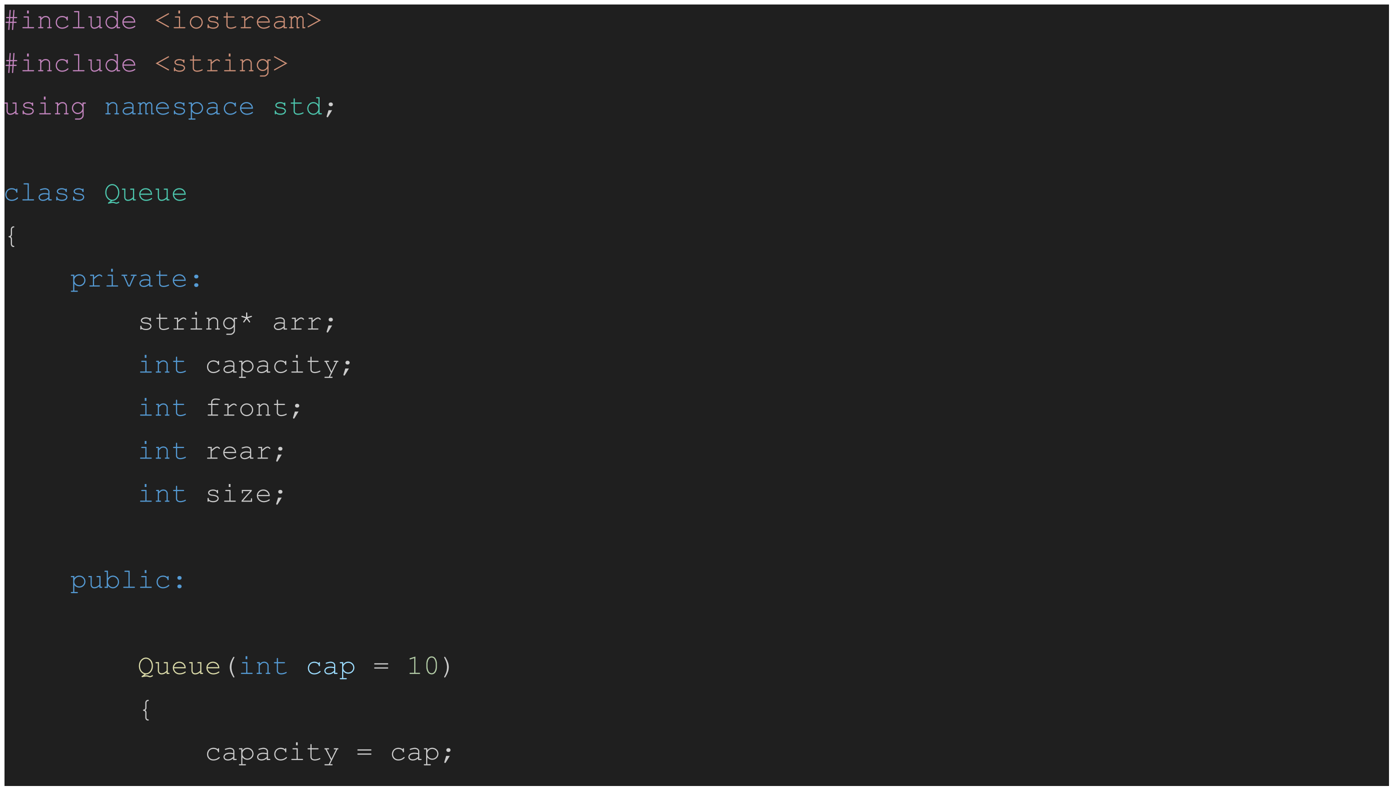
```
class Node
{
    public:
        string name;
        Node* next;

        Node(string val)
        {
            name = val;
            next = nullptr;
        }
};


class Queue
{
    private:
        Node* front;
        Node* rear;

    public:
        Queue()
        {
            front = nullptr;
            rear = nullptr;
        }


    void enqueue(string name)
    {
        Node* newNode = new Node(name);
        if (rear == nullptr)
        {
            front = rear = newNode;
        } else
        {
            rear->next = newNode;
            rear = newNode;
        }
    }


    void dequeue()
    {
        if (front == nullptr)
        {
            cout << "Queue is empty. Cannot dequeue.\n";
            return;
        }
        Node* temp = front;
        front = front->next;


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


        delete temp;
    }


    void display()
    {
        if (front == nullptr)
        {
            cout << "Queue is empty." << endl;
            return;
        }
        Node* temp = front;
        cout << "Queue: ";
        while (temp != nullptr)
        {
            cout << temp->name << " ";
            temp = temp->next;
        }
        cout << endl;
    }


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

```cpp
int main()
{
    Queue q;


    cout << "Inserting 'Vincent' into an empty queue." << endl;
    q.enqueue("Vincent");
    q.display();


    cout << "\nInserting 'Frederick' into a non-empty queue." << endl;
    q.enqueue("Frederick");
    q.display();


    cout << "\nInserting 'Andrei' and 'Anna' into the queue." << endl;
    q.enqueue("Andrei");
    q.enqueue("Anna");
    q.display();


    cout << "\nDeleting the front item (queue has more than one item)." << endl;
    q.dequeue();
    q.display();


    cout << "\nDeleting items until only one is left." << endl;
    q.dequeue();
    q.display();


    cout << "\nDeleting the last remaining item from the queue." << endl;
    q.dequeue();
    q.display();


    cout << "\nAttempting to delete from an empty queue." << endl;
    q.dequeue();
    return 0;
}
```



```
/tmp/xLhVW4UsMW.o
Inserting 'Vincent' into an empty queue.
Queue: Vincent

Inserting 'Frederick' into a non-empty queue.
Queue: Vincent Frederick

Inserting 'Andrei' and 'Anna' into the queue.
Queue: Vincent Frederick Andrei Anna

Deleting the front item (queue has more than one item).
Queue: Frederick Andrei Anna

Deleting items until only one is left...
Queue: Andrei Anna

Deleting the last remaining item from the queue.
Queue: Anna

Attempting to delete from an empty queue.


=== Code Execution Successful ===
```

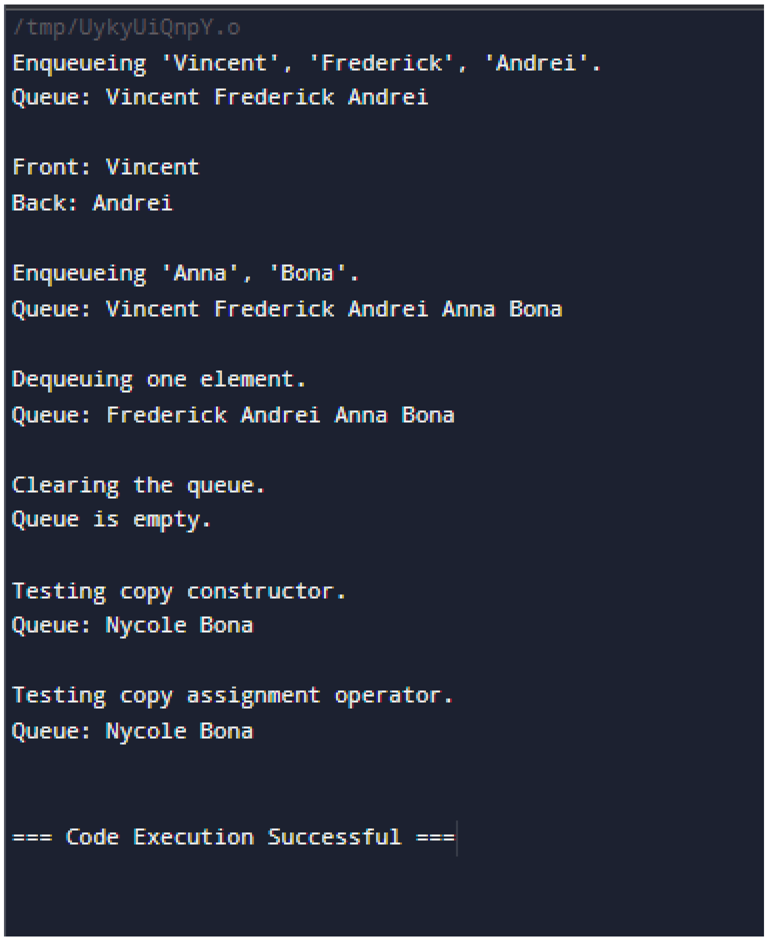
Table 5-2. Queues using Linked List Implementation



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

class Queue
{
    private:
        string* arr;
        int capacity;
        int front;
        int rear;
        int size;

    public:

        Queue(int cap = 10)
        {
            capacity = cap;
            arr = new string[capacity];
            front = 0;
            rear = -1;
            size = 0;
        }


    ~Queue()
    {
        delete[] arr;
    }


    Queue(const Queue& other)
    {
        capacity = other.capacity;
        front = other.front;
        rear = other.rear;
        size = other.size;
        arr = new string[capacity];
        for (int i = 0; i < capacity; i++)
        {
            arr[i] = other.arr[i];
        }
    }


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

            capacity = other.capacity;
            front = other.front;
            rear = other.rear;
            size = other.size;
            arr = new string[capacity];
            for (int i = 0; i < capacity; i++)
            {
                arr[i] = other.arr[i];
            }
        }
```

```cpp
        return *this;
    }

    bool isEmpty() const
    {
        return size == 0;
    }


    int getSize() const
    {
        return size;
    }

    void clear()
    {
        front = 0;
        rear = -1;
        size = 0;
    }


    string getFront() const
    {
        if (isEmpty())
        {
            throw out_of_range("Queue is empty");
        }
        return arr[front];
    }


    string getBack() const
    {
        if (isEmpty())
        {
            throw out_of_range("Queue is empty");
        }
        return arr[rear];
    }


    void enqueue(const string& name)
    {
        if (size == capacity)
        {
            throw overflow_error("Queue overflow");
        }
        rear = (rear + 1) % capacity;
        arr[rear] = name;
        size++;
    }


    void dequeue()
    {
        if (isEmpty())
        {
            throw underflow_error("Queue underflow");
        }
        front = (front + 1) % capacity;
        size--;
    }


    void display() const
    {
        if (isEmpty())
        {
            cout << "Queue is empty." << endl;
            return;
        }
        cout << "Queue: ";
        for (int i = 0; i < size; i++)
        {
            cout << arr[(front + i) % capacity] << " ";
        }
        cout << endl;
    }
};

int main()
{

    Queue q(5);
```

```cpp
    cout << "Enqueueing 'Vincent', 'Frederick', 'Andrei'." << endl;
    q.enqueue("Vincent");
    q.enqueue("Frederick");
    q.enqueue("Andrei");
    q.display();


    cout << "\nFront: " << q.getFront() << endl;
    cout << "Back: " << q.getBack() << endl;

    cout << "\nEnqueueing 'Anna', 'Bona'." << endl;
    q.enqueue("Anna");
    q.enqueue("Bona");
    q.display();

    cout << "\nDequeuing one element." << endl;
    q.dequeue();
    q.display();

    cout << "\nClearing the queue." << endl;
    q.clear();
    q.display();

    cout << "\nTesting copy constructor." << endl;
    q.enqueue("Nycole");
    q.enqueue("Bona");
    Queue q2 = q;
    q2.display();

    cout << "\nTesting copy assignment operator." << endl;
    Queue q3;
    q3 = q2;
    q3.display();

    return 0;
}
```



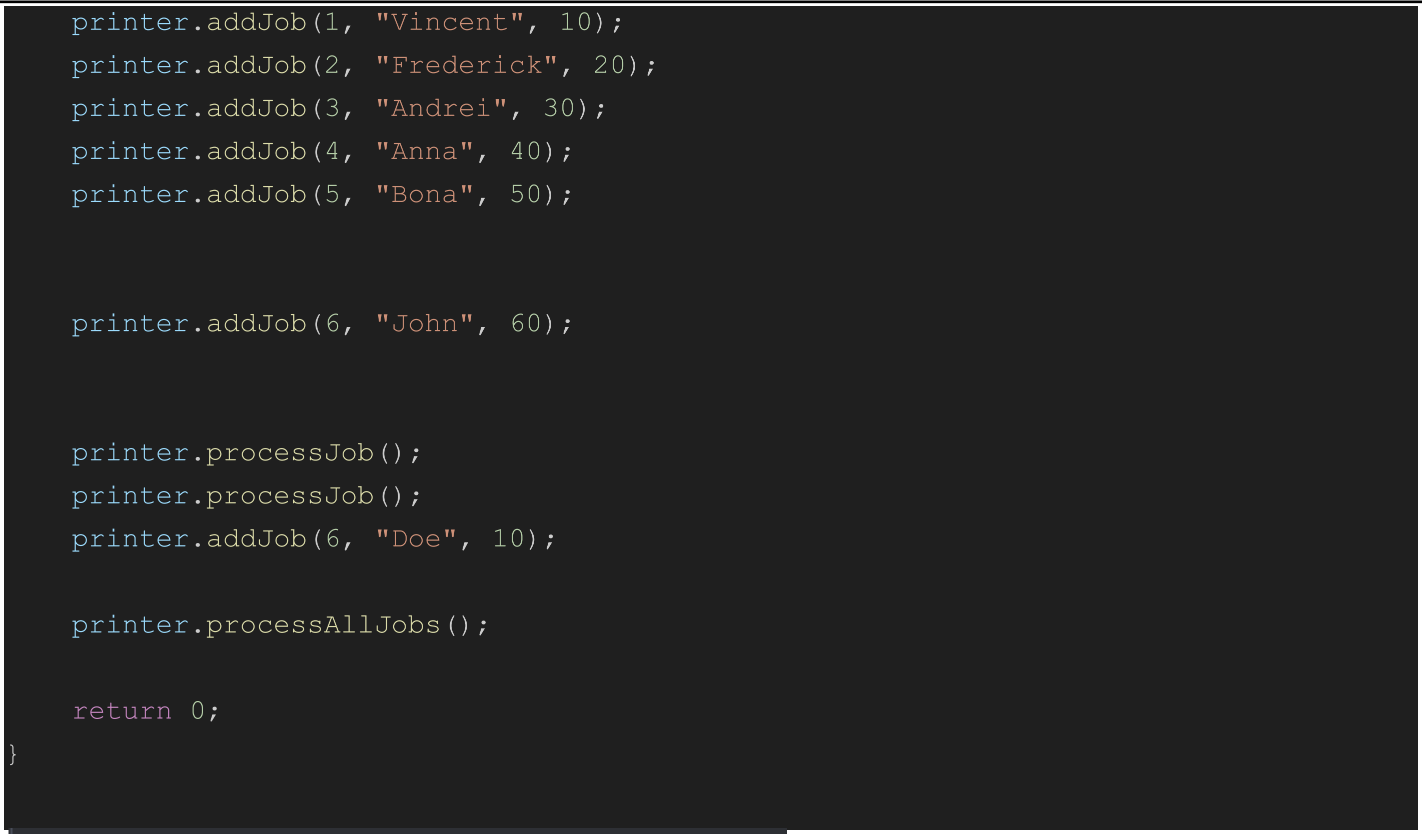
```
/tmp/UykyUiQnpY.o
Enqueueing 'Vincent', 'Frederick', 'Andrei'.
Queue: Vincent Frederick Andrei

Front: Vincent
Back: Andrei

Enqueueing 'Anna', 'Bona'.
Queue: Vincent Frederick Andrei Anna Bona

Dequeuing one element.
Queue: Frederick Andrei Anna Bona

Clearing the queue.
Queue is empty.

Testing copy constructor.
Queue: Nycole Bona

Testing copy assignment operator.
Queue: Nycole Bona


=== Code Execution Successful ===
```

Table 5-3. Queues using Array Implementation

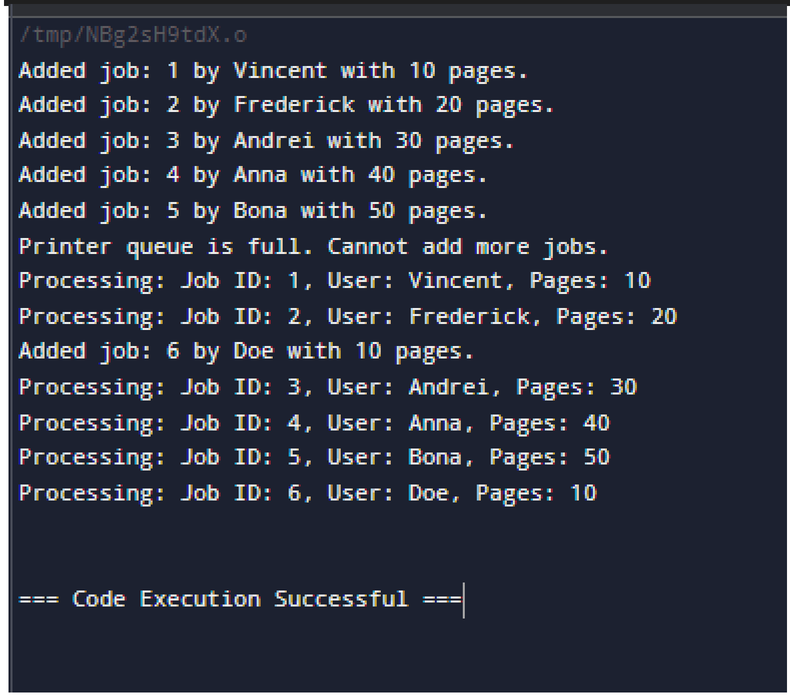
7. Supplementary Activity

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

class Job
{
    public:
        int jobID;
        string userName;
        int numPages;

    Job(int id, string user, int pages)
    {
        jobID = id;
        userName = user;
        numPages = pages;
    }
    void displayJob() {
        cout << "Job ID: " << jobID << ", User: " << userName << ", Pages: " <<
numPages << endl;
    }
};

class Printer
{
    private:
        Job** jobQueue;
        int capacity;
        int front;
        int rear;
        int size;

public:
    Printer(int cap = 10)
    {
        capacity = cap;
        jobQueue = new Job*[capacity];
        front = 0;
        rear = -1;
        size = 0;
    }

    ~Printer()
    {
        for (int i = 0; i < size; ++i)
        {
            delete jobQueue[(front + i) % capacity];
        }
        delete[] jobQueue;
    }

    bool isFull() const
    {
        return size == capacity;
    }

    bool isEmpty() const
    {
        return size == 0;
    }
```

```cpp
    void addJob(int jobID, string userName, int numPages)
    {
        if (isFull())
        {
            cout << "Printer queue is full. Cannot add more jobs.\n";
            return;
        }
        rear = (rear + 1) % capacity;
        jobQueue[rear] = new Job(jobID, userName, numPages);
        size++;
        cout << "Added job: " << jobID << " by " << userName << " with " << numPages
<< " pages." << endl;
    }

    void processJob()
    {
        if (isEmpty())
        {
            cout << "No jobs to process.\n";
            return;
        }
        Job* currentJob = jobQueue[front];
        front = (front + 1) % capacity;
        size--;

        cout << "Processing: ";
        currentJob->displayJob();
        delete currentJob;
    }

    void processAllJobs()
    {
        while (!isEmpty())
        {
            processJob();
        }
    }
};

int main()
{
    Printer printer(5);
```



```
    printer.addJob(1, "Vincent", 10);
    printer.addJob(2, "Frederick", 20);
    printer.addJob(3, "Andrei", 30);
    printer.addJob(4, "Anna", 40);
    printer.addJob(5, "Bona", 50);


    printer.addJob(6, "John", 60);


    printer.processJob();
    printer.processJob();
    printer.addJob(6, "Doe", 10);

    printer.processAllJobs();

    return 0;
}
```



```
/tmp/NBg2sH9tdX.o
Added job: 1 by Vincent with 10 pages.
Added job: 2 by Frederick with 20 pages.
Added job: 3 by Andrei with 30 pages.
Added job: 4 by Anna with 40 pages.
Added job: 5 by Bona with 50 pages.
Printer queue is full. Cannot add more jobs.
Processing: Job ID: 1, User: Vincent, Pages: 10
Processing: Job ID: 2, User: Frederick, Pages: 20
Added job: 6 by Doe with 10 pages.
Processing: Job ID: 3, User: Andrei, Pages: 30
Processing: Job ID: 4, User: Anna, Pages: 40
Processing: Job ID: 5, User: Bona, Pages: 50
Processing: Job ID: 6, User: Doe, Pages: 10


=== Code Execution Successful ===
```

8. Conclusion

During this laboratory, I learned about queues, which follow the First-In-First-Out (FIFO) principle. I explored different implementations: C++ STL queues, linked lists, and arrays, each with its advantages. Using STL provided a convenient way to manage queues without manual memory management, while linked lists demonstrated dynamic resizing. The array implementation was efficient for fixed sizes and suitable for applications like a job printer simulation.

In the supplementary activity, I effectively demonstrated how queues manage tasks on a first-come, first-served basis. I followed a systematic approach to implement various queue types, ensuring the handling of edge cases like overflow and underflow. Testing confirmed that each implementation behaved as expected, highlighting performance trade-offs between them.

I chose an array-based implementation for the job printer because it was simple and efficient, providing quick access times for the limited number of jobs the printer could handle. This choice allowed for effective job management.

Reflecting on the laboratory activity, I believe I successfully grasped key concepts about queues. However, I still acknowledge that my knowledge is still very basic because there are still methods and techniques of working with queues that are quite advanced which I have not mastered yet. I use different resources online to answer this laboratory and to improve my understanding and programming capability.

9. Assessment Rubric