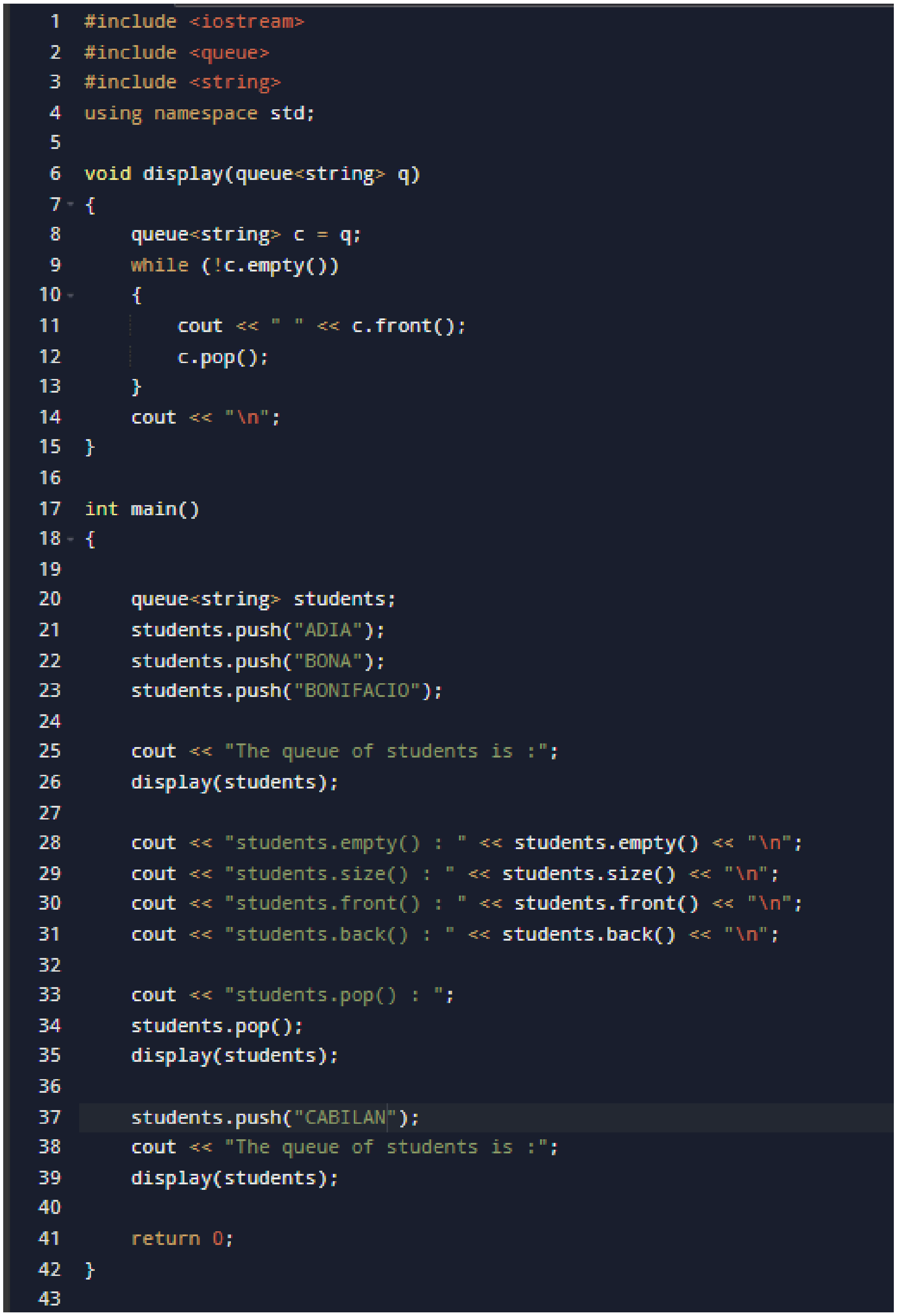
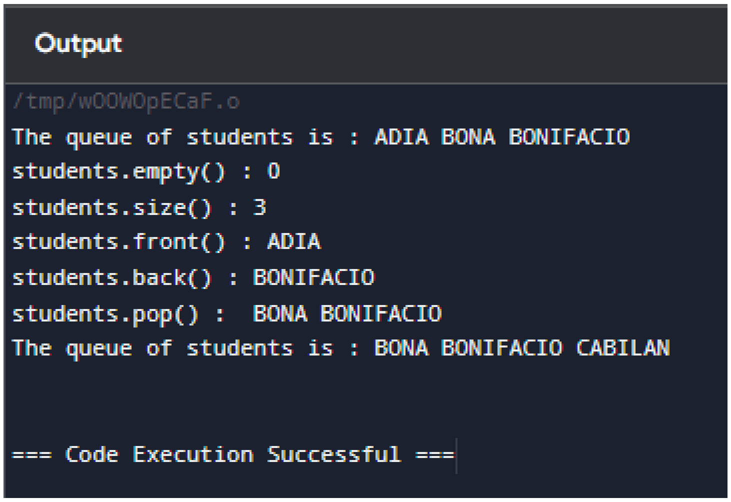
Activity No. 5.1	
Hands-on Activity 5.1 Queues	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 10-07-2024**
Section: CpE21 S4	**Date Submitted: 10-07-2024**
Name(s):BONIFACIO, NYKO ADREIN L.	**Instructor: Prof. Sayo**
6. Output	

CODE



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

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

int main()
{

    queue<string> students;
    students.push("ADIA");
    students.push("BONA");
    students.push("BONIFACIO");

    cout << "The queue of students is :";
    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";

    cout << "students.pop() : ";
    students.pop();
    display(students);

    students.push("CABILAN");
    cout << "The queue of students is :";
    display(students);

    return 0;
}

```

OUTPUT

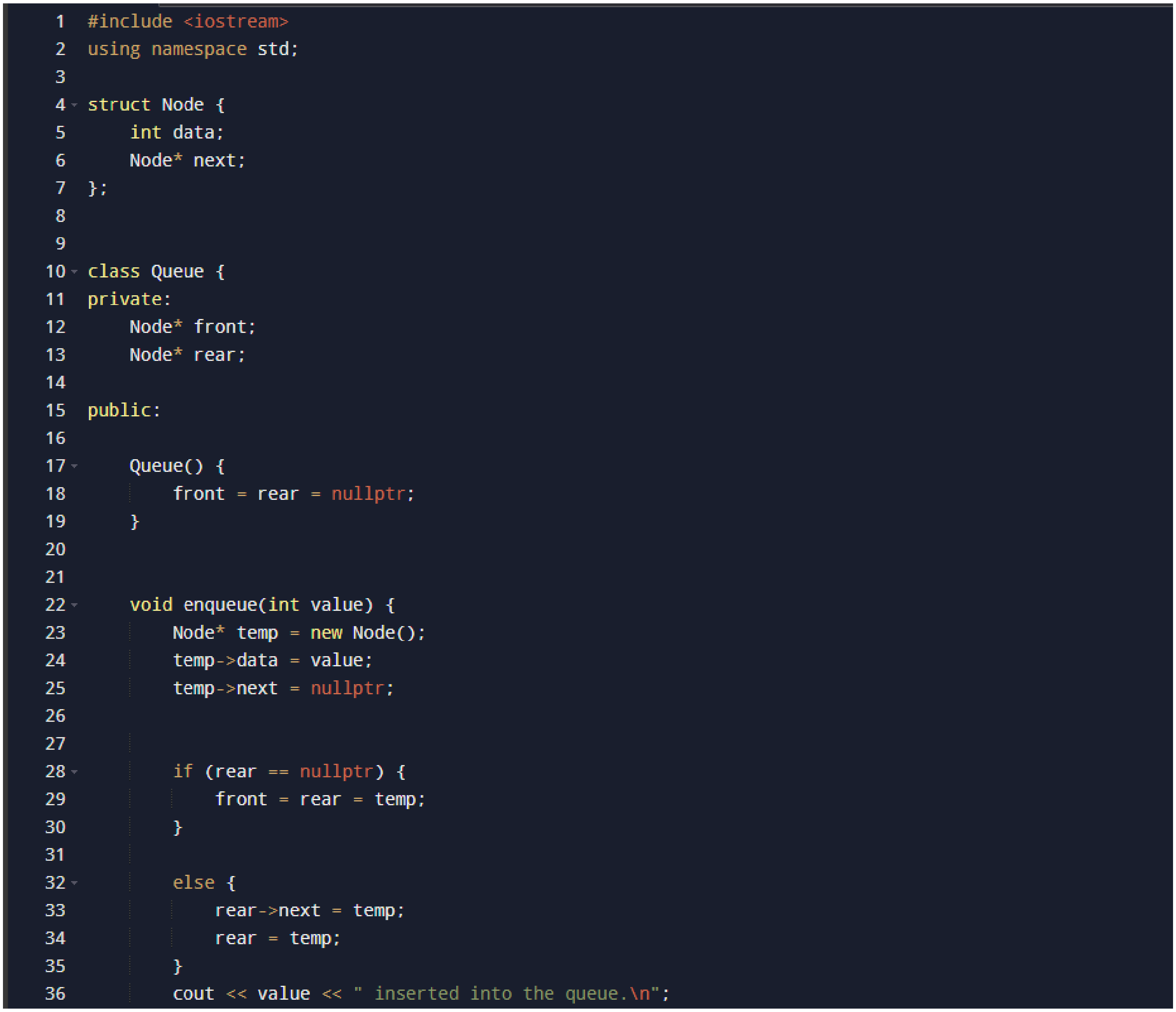


Output

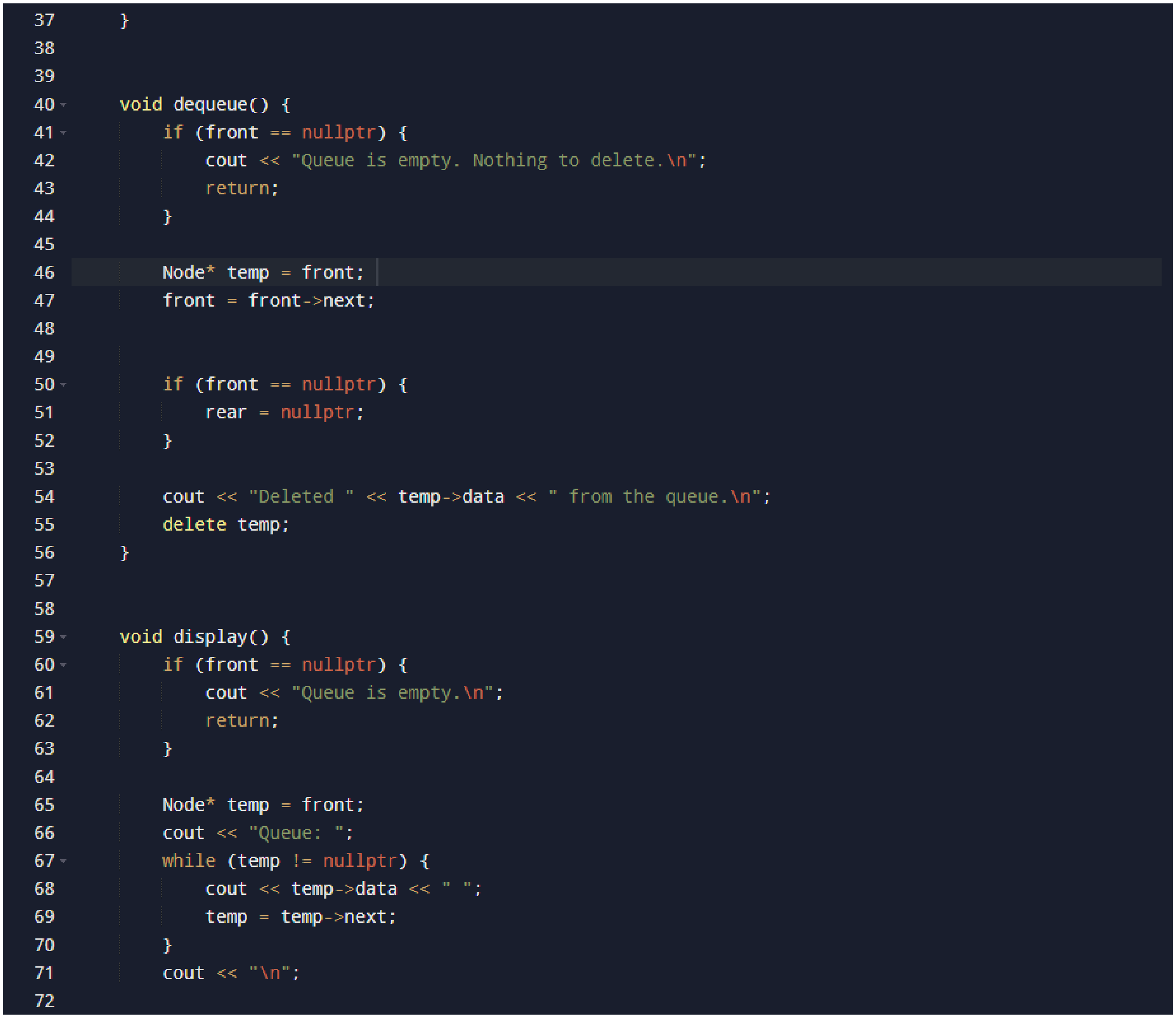
```
/tmp/wOOWOpECaF.o
The queue of students is : ADIA BONA BONIFACIO
students.empty() : 0
students.size() : 3
students.front() : ADIA
students.back() : BONIFACIO
students.pop() :  BONA BONIFACIO
The queue of students is : BONA BONIFACIO CABILAN


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

Table 5-1. Queues Using C++ STL

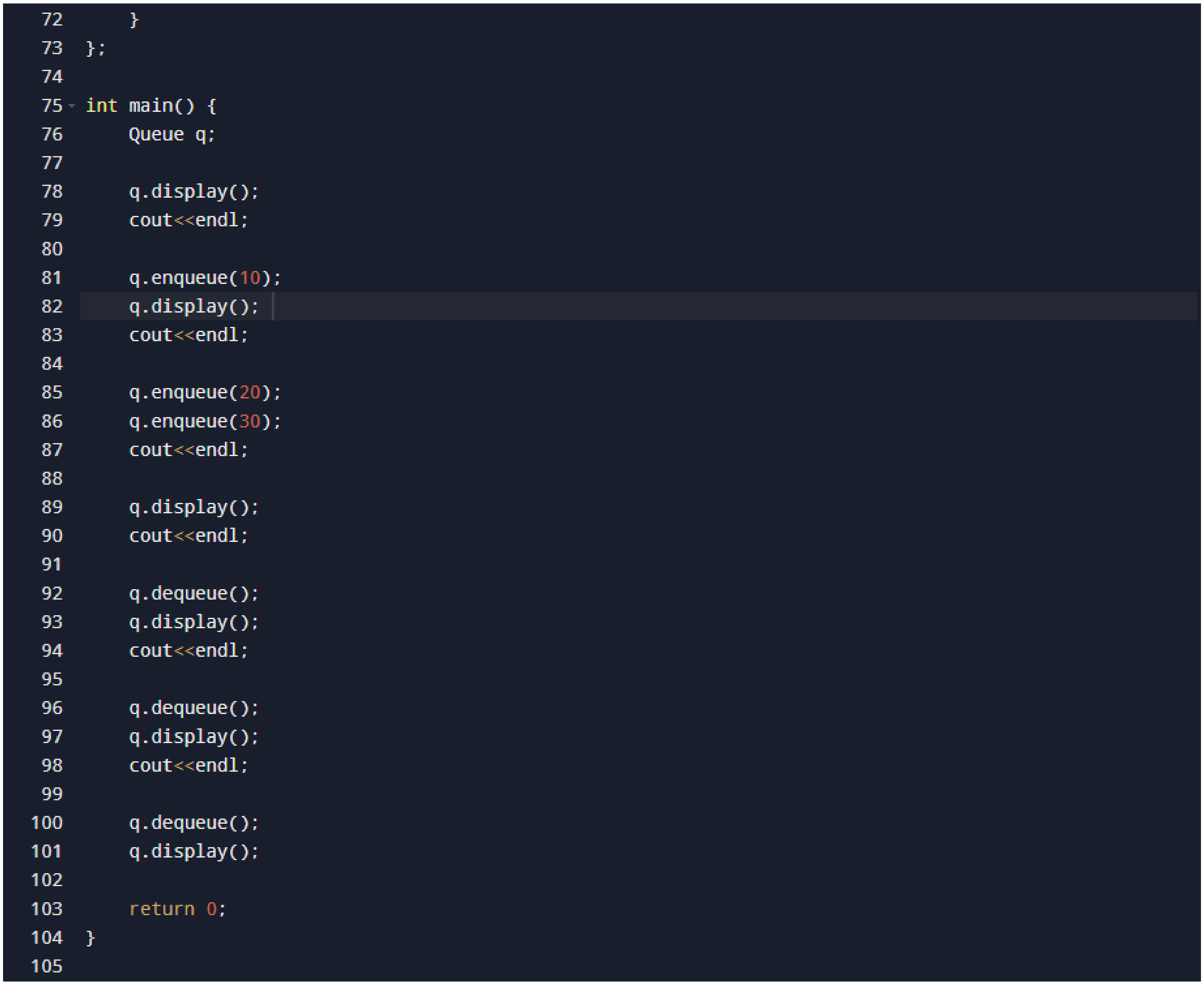
CODE



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

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


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

public:

    Queue() {
        front = rear = nullptr;
    }


    void enqueue(int value) {
        Node* temp = new Node();
        temp->data = value;
        temp->next = nullptr;


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

        else {
            rear->next = temp;
            rear = temp;
        }
        cout << value << " inserted into the queue.\n";
```

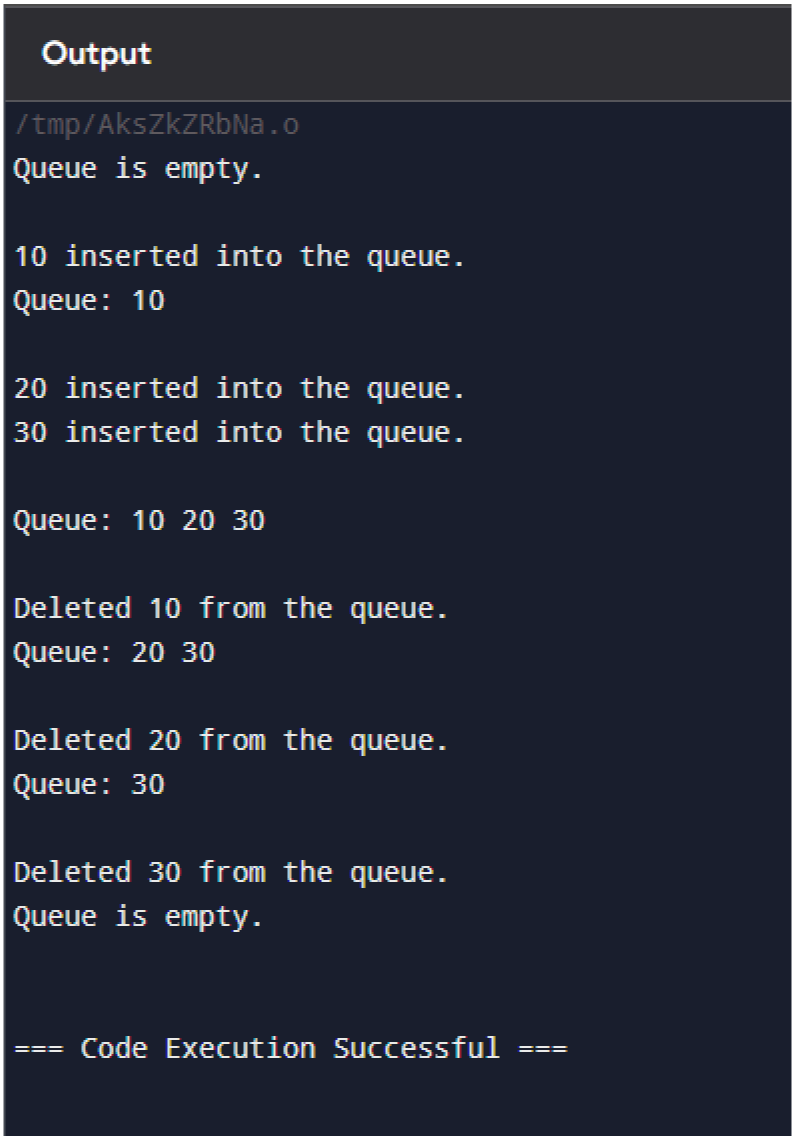


```cpp
    }


    void dequeue() {
        if (front == nullptr) {
            cout << "Queue is empty. Nothing to delete.\n";
            return;
        }

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


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

        cout << "Deleted " << temp->data << " from the queue.\n";
        delete temp;
    }


    void display() {
        if (front == nullptr) {
            cout << "Queue is empty.\n";
            return;
        }

        Node* temp = front;
        cout << "Queue: ";
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << "\n";

```



```cpp
    }
};

int main() {
    Queue q;

    q.display();
    cout<<endl;

    q.enqueue(10);
    q.display();
    cout<<endl;

    q.enqueue(20);
    q.enqueue(30);
    cout<<endl;

    q.display();
    cout<<endl;

    q.dequeue();
    q.display();
    cout<<endl;

    q.dequeue();
    q.display();
    cout<<endl;

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

    return 0;
}
```

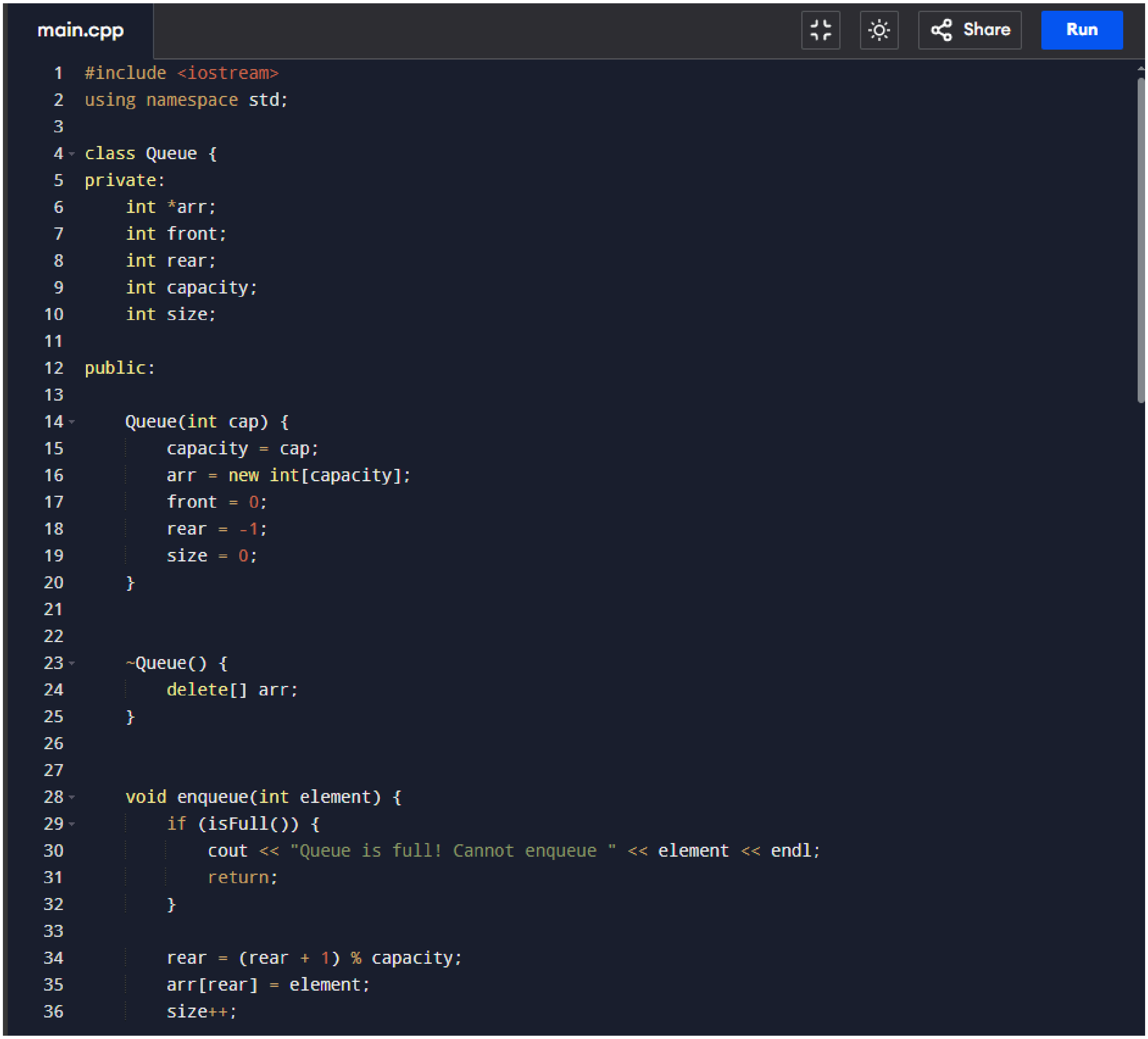
OUTPUT



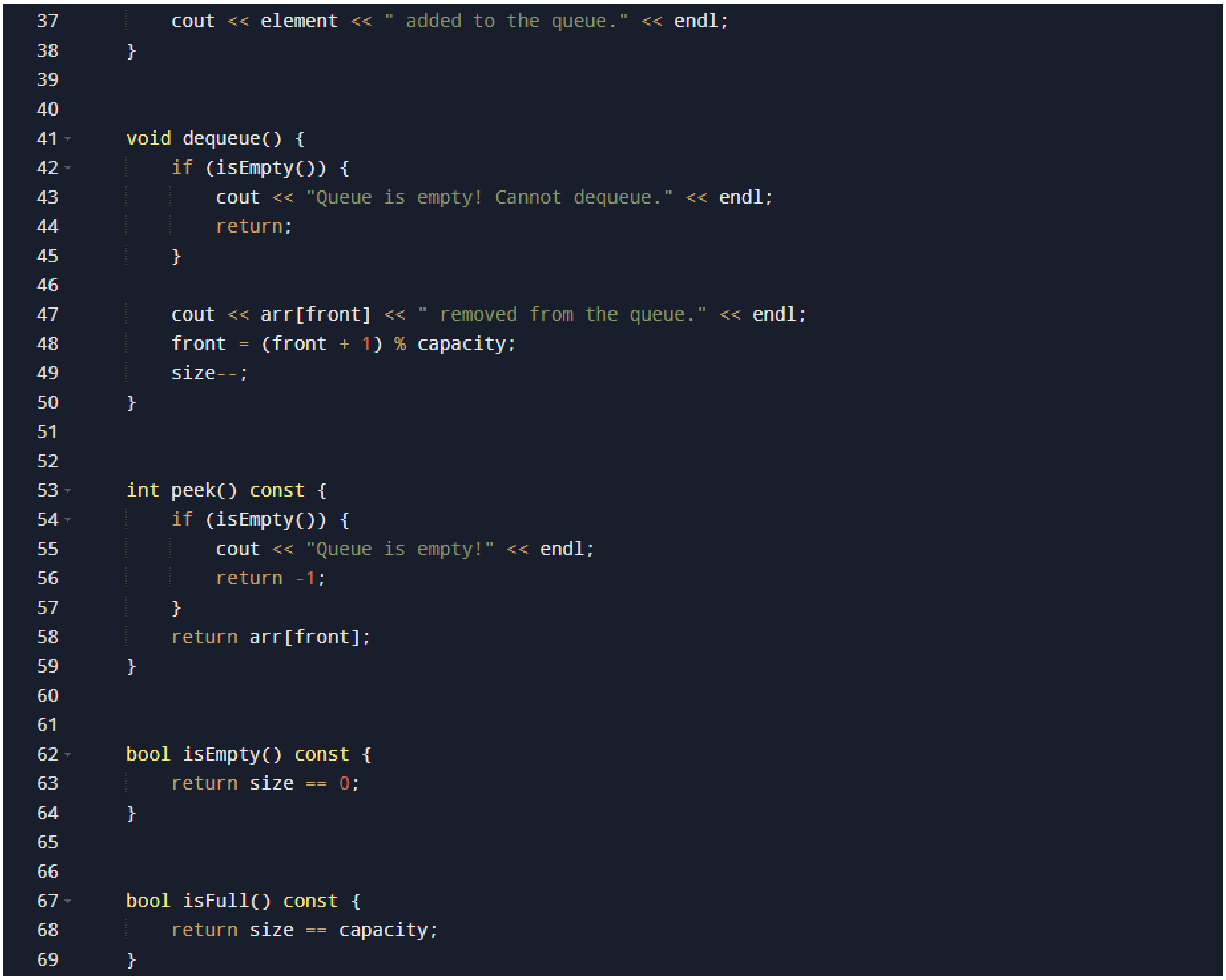
```
/tmp/AksZkZRbNa.o
Queue is empty.

10 inserted into the queue.
Queue: 10

20 inserted into the queue.
30 inserted into the queue.

Queue: 10 20 30

Deleted 10 from the queue.
Queue: 20 30

Deleted 20 from the queue.
Queue: 30

Deleted 30 from the queue.
Queue is empty.


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

Table 5-2. Queues using Linked List Implementation

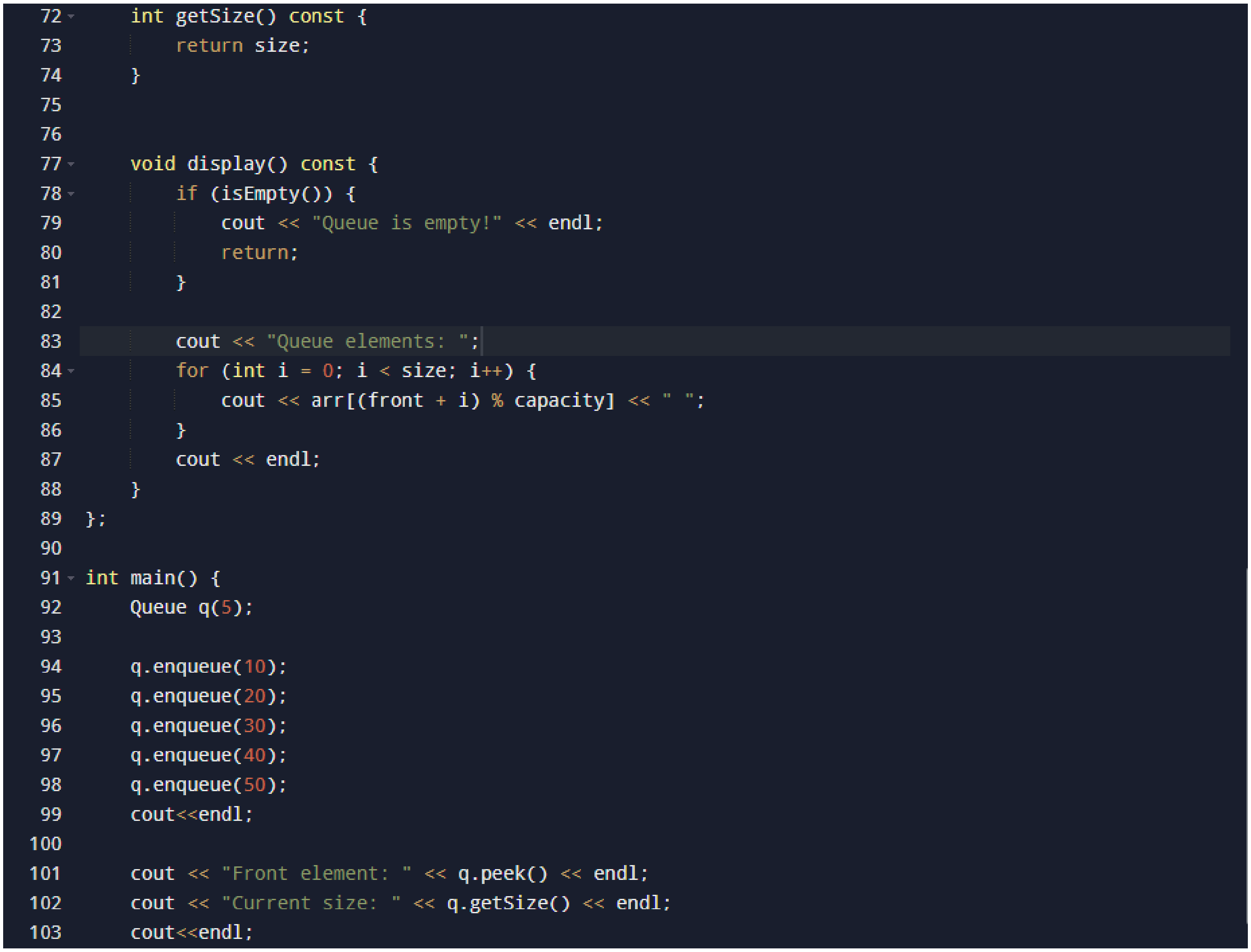
CODE



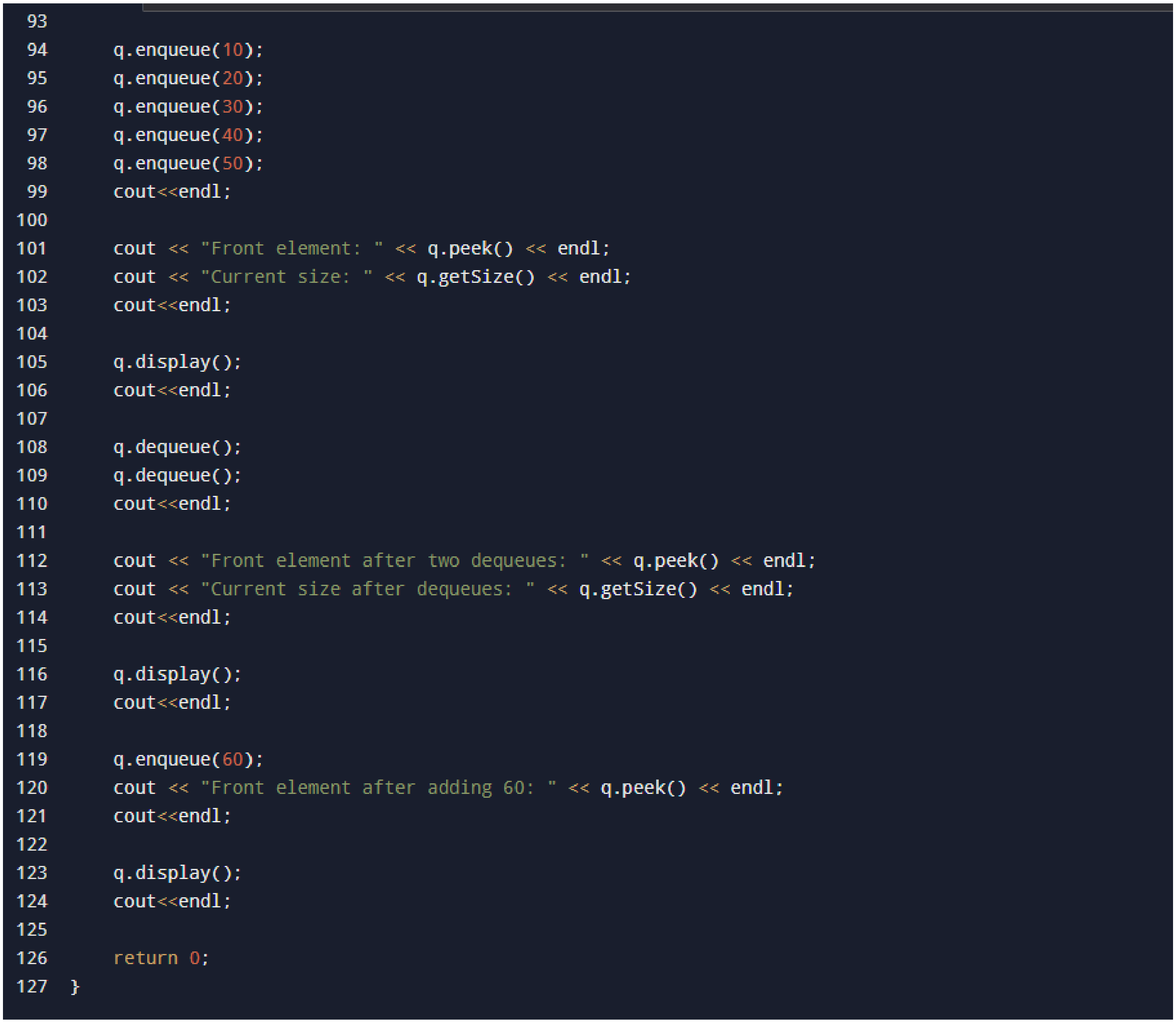
```cpp
#include <iostream>
using namespace std;

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

public:

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


    ~Queue() {
        delete[] arr;
    }


    void enqueue(int element) {
        if (isFull()) {
            cout << "Queue is full! Cannot enqueue " << element << endl;
            return;
        }

        rear = (rear + 1) % capacity;
        arr[rear] = element;
        size++;
```



```cpp
        cout << element << " added to the queue." << endl;
    }


    void dequeue() {
        if (isEmpty()) {
            cout << "Queue is empty! Cannot dequeue." << endl;
            return;
        }

        cout << arr[front] << " removed from the queue." << endl;
        front = (front + 1) % capacity;
        size--;
    }


    int peek() const {
        if (isEmpty()) {
            cout << "Queue is empty!" << endl;
            return -1;
        }
        return arr[front];
    }


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


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



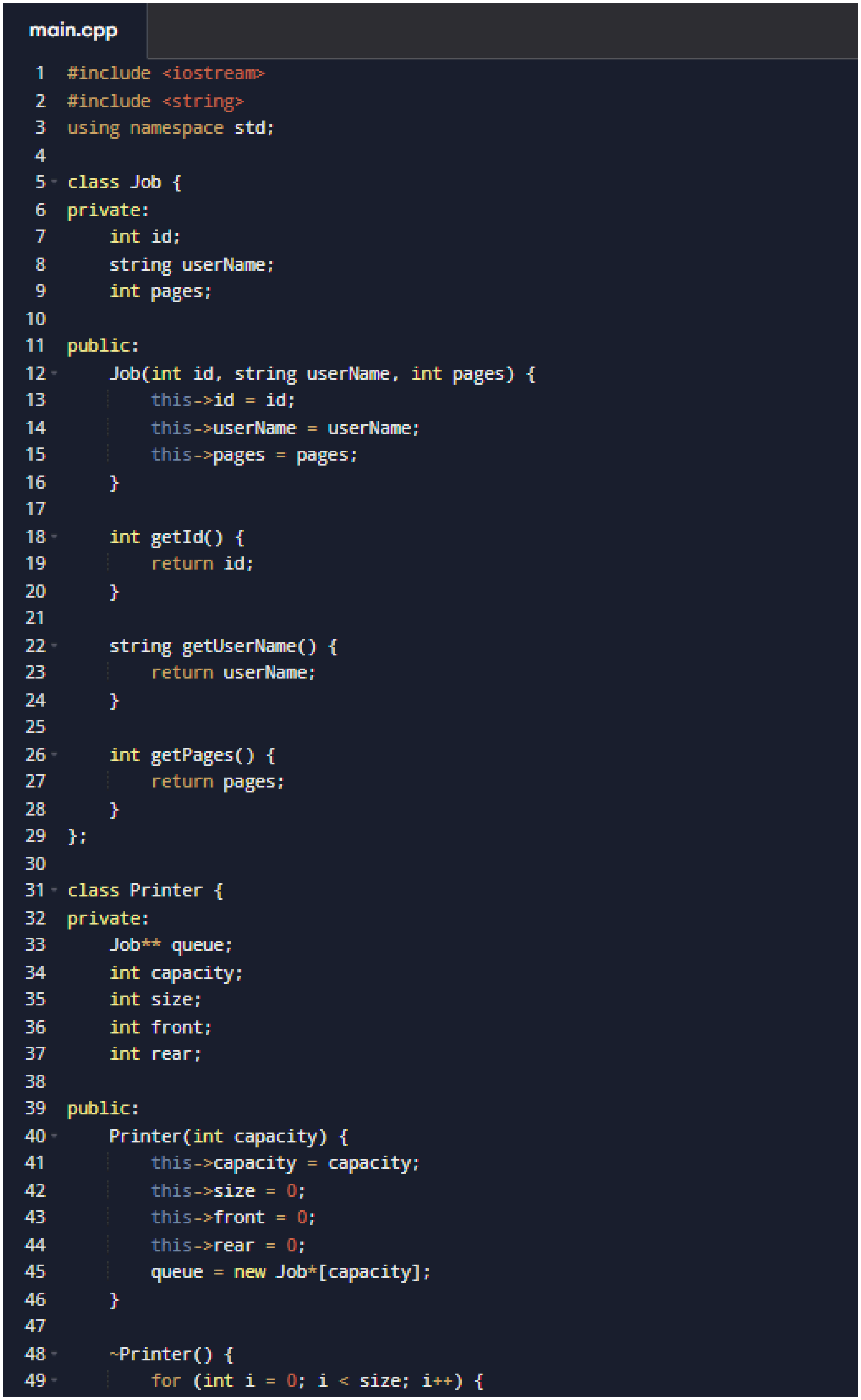
```cpp
    int getSize() const {
        return size;
    }


    void display() const {
        if (isEmpty()) {
            cout << "Queue is empty!" << endl;
            return;
        }

        cout << "Queue elements: ";
        for (int i = 0; i < size; i++) {
            cout << arr[(front + i) % capacity] << " ";
        }
        cout << endl;
    }
};

int main() {
    Queue q(5);

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.enqueue(40);
    q.enqueue(50);
    cout<<endl;

    cout << "Front element: " << q.peek() << endl;
    cout << "Current size: " << q.getSize() << endl;
    cout<<endl;
```



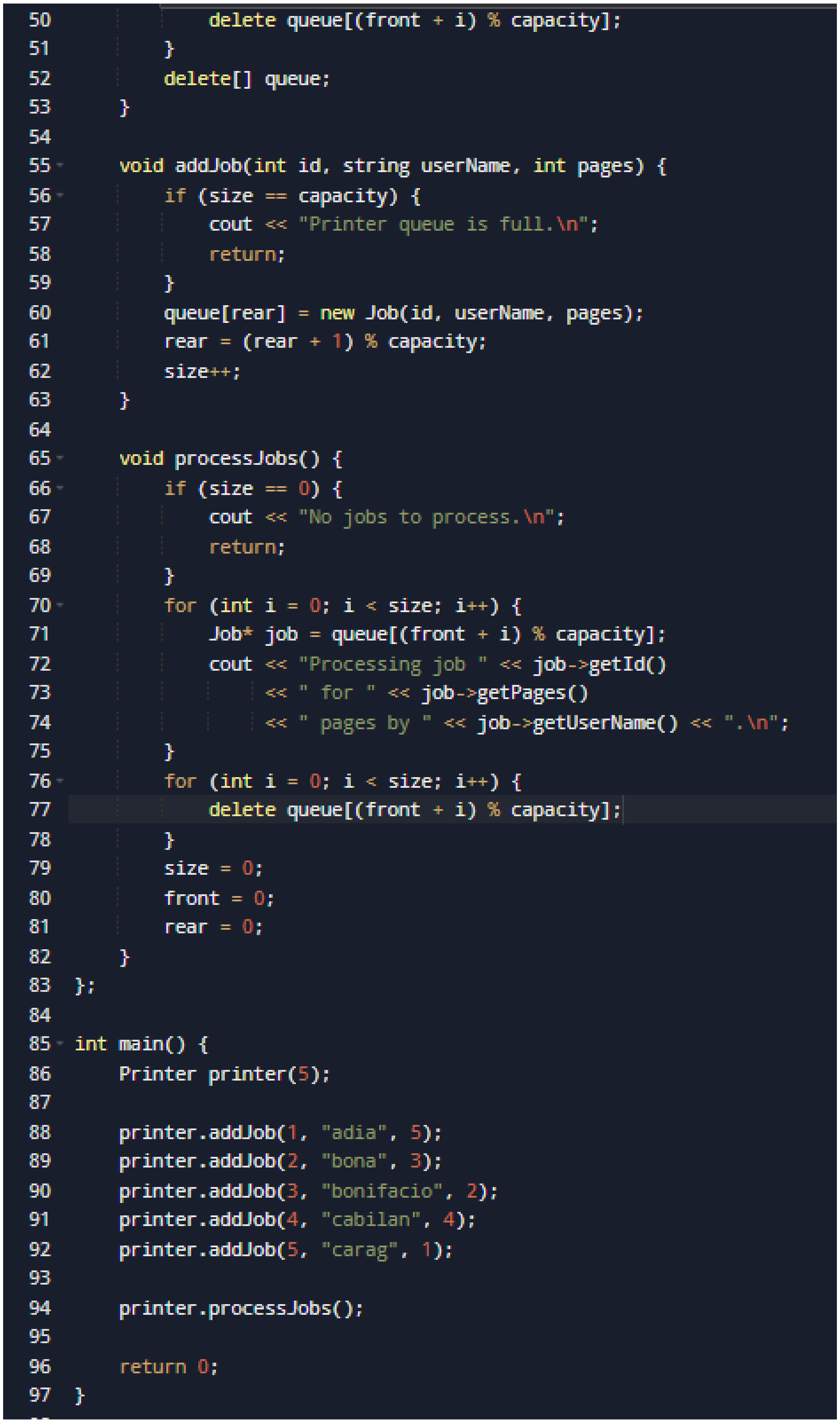
```cpp

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.enqueue(40);
    q.enqueue(50);
    cout<<endl;

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

    q.display();
    cout<<endl;

    q.dequeue();
    q.dequeue();
    cout<<endl;

    cout << "Front element after two dequeues: " << q.peek() << endl;
    cout << "Current size after dequeues: " << q.getSize() << endl;
    cout<<endl;

    q.display();
    cout<<endl;

    q.enqueue(60);
    cout << "Front element after adding 60: " << q.peek() << endl;
    cout<<endl;

    q.display();
    cout<<endl;

    return 0;
}
```

Table 5-3. Queues using Array Implementation

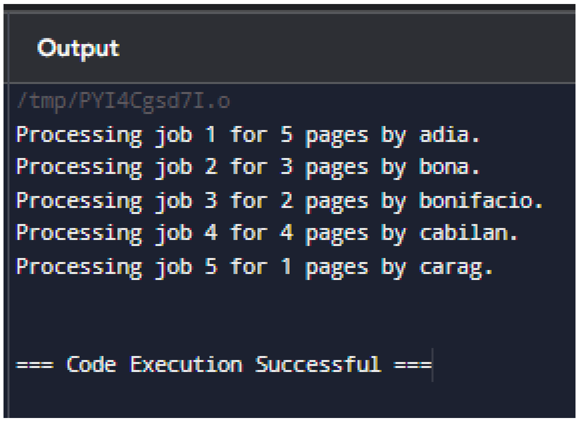
7. Supplementary Activity



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

class Job {
private:
    int id;
    string userName;
    int pages;

public:
    Job(int id, string userName, int pages) {
        this->id = id;
        this->userName = userName;
        this->pages = pages;
    }

    int getId() {
        return id;
    }

    string getUserName() {
        return userName;
    }

    int getPages() {
        return pages;
    }
};

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

public:
    Printer(int capacity) {
        this->capacity = capacity;
        this->size = 0;
        this->front = 0;
        this->rear = 0;
        queue = new Job*[capacity];
    }

    ~Printer() {
        for (int i = 0; i < size; i++) {
```



```cpp
            delete queue[(front + i) % capacity];
        }
        delete[] queue;
    }

    void addJob(int id, string userName, int pages) {
        if (size == capacity) {
            cout << "Printer queue is full.\n";
            return;
        }
        queue[rear] = new Job(id, userName, pages);
        rear = (rear + 1) % capacity;
        size++;
    }

    void processJobs() {
        if (size == 0) {
            cout << "No jobs to process.\n";
            return;
        }
        for (int i = 0; i < size; i++) {
            Job* job = queue[(front + i) % capacity];
            cout << "Processing job " << job->getId()
                 << " for " << job->getPages()
                 << " pages by " << job->getUserName() << ".\n";
        }
        for (int i = 0; i < size; i++) {
            delete queue[(front + i) % capacity];
        }
        size = 0;
        front = 0;
        rear = 0;
    }
};

int main() {
    Printer printer(5);

    printer.addJob(1, "adia", 5);
    printer.addJob(2, "bona", 3);
    printer.addJob(3, "bonifacio", 2);
    printer.addJob(4, "cabilan", 4);
    printer.addJob(5, "carag", 1);

    printer.processJobs();

    return 0;
}
```



```
/tmp/PYI4Cgsd7I.o
Processing job 1 for 5 pages by adia.
Processing job 2 for 3 pages by bona.
Processing job 3 for 2 pages by bonifacio.
Processing job 4 for 4 pages by cabilan.
Processing job 5 for 1 pages by carag.


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

Arrays in the printer queue provide fast access and better cache performance, making them efficient for a known maximum number of jobs. In contrast, linked lists offer dynamic sizing and easier insertion and deletion, making them flexible for varying job counts. The choice depends on whether job limits are fixed or fluctuating.

8. Conclusion

I gained an understanding of queues' basic functions as well as the variations between implementations such as arrays and linked lists from studying them. My comprehension of effectively managing data and memory was reinforced by the practical process. Examining other queue applications through additional exercises improved my understanding of their function in computer science, including task scheduling. All in all, I think I did a good job of understanding the main ideas, but I also know that in order to get better at solving problems, I need to learn more about sophisticated data structures.

9. Assessment Rubric