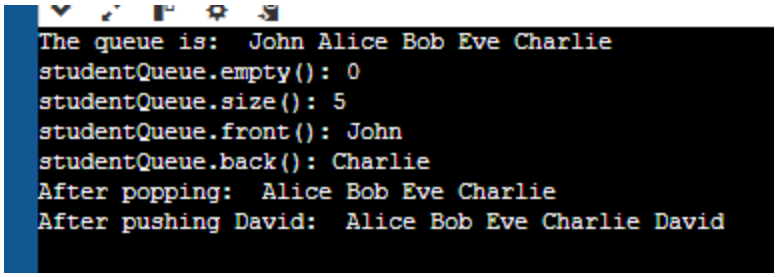
Activity No. 5	
Queues	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 07/10/2024**
Section: CPE21S4	**Date Submitted: 07/10/2024**
Name(s): TITONG, LEE IVAN B.	**Instructor: Ma'am Sayo**

6. Output

```
#include <iostream>
#include <queue>
#include <string>

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

int main() {
    std::string students[] = {"John", "Alice", "Bob", "Eve", "Charlie"};
    std::queue<std::string> studentQueue;

    for (int i = 0; i < 5; i++) {
        studentQueue.push(students[i]);
    }

    std::cout << "The queue is: ";
    display(studentQueue);

    std::cout << "studentQueue.empty(): " << studentQueue.empty() << "\n";
    std::cout << "studentQueue.size(): " << studentQueue.size() << "\n";
    std::cout << "studentQueue.front(): " << studentQueue.front() << "\n";
    std::cout << "studentQueue.back(): " << studentQueue.back() << "\n";

    studentQueue.pop();
    std::cout << "After popping: ";
    display(studentQueue);

    studentQueue.push("David");
    std::cout << "After pushing David: ";
    display(studentQueue);

    return 0;
}
```

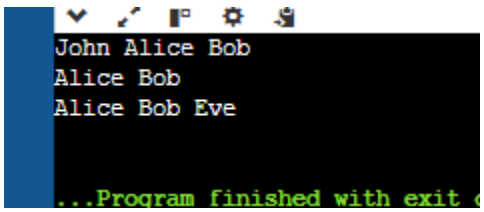


Operation	Output
Create queue	The queue is: John Alice Bob Eve Charlie
Check empty	studentQueue.empty(): 0
Get size	studentQueue.size(): 5
Get front	studentQueue.front(): John
Get back	studentQueue.back(): Charlie
Pop	After popping: Alice Bob Eve Charlie
Push	After pushing David: Alice Bob Eve Charlie David

Table 5-1. Queues using C++ STL

```
#include <iostream>

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

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

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

    void enqueue(std::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 dequeue() {
        if (front == nullptr) {
            std::cout << "Queue is empty\n";
            return;
        }

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

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

        delete temp;
    }

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

int main() {
    Queue q;

    q.enqueue("John");
    q.enqueue("Alice");
    q.enqueue("Bob");
    q.display();

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

    q.enqueue("Eve");
    q.display();

    return 0;
}
```



Operation	Output
Enqueue John, Alice, Bob	John Alice Bob
Dequeue	Alice Bob
Enqueue Eve	Alice Bob Eve

Table 5-2. Queues using Linked List Implementation

```
#include <iostream>

class Queue {
private:
    std::string* q_array;
    int q_capacity;
    int q_size;
    int q_front;
    int q_back;

public:
    Queue(int capacity) : q_capacity(capacity), q_size(0), q_front(0), q_back(0) {
        q_array = new std::string[capacity];
    }

    ~Queue() {
        delete[] q_array;
    }

    void enqueue(std::string item) {
        if (q_size == q_capacity) {
            std::cout << "Queue is full\n";
            return;
        }

        q_array[q_back] = item;
        q_back = (q_back + 1) % q_capacity;
        q_size++;
    }

    void dequeue() {
        if (q_size == 0) {
            std::cout << "Queue is empty\n";
            return;
        }

        std::string item = q_array[q_front];
        q_front = (q_front + 1) % q_capacity;
        q_size--;
    }

        void display() {
```

```
        for (int i = 0; i < q_size; i++) {
            std::cout << q_array[(q_front + i) % q_capacity] << " ";
        }
        std::cout << "\n";
    }

    bool empty() {
        return q_size == 0;
    }

    int size() {
        return q_size;
    }

    std::string front() {
        return q_array[q_front];
    }

    std::string back() {
        return q_array[(q_back - 1 + q_capacity) % q_capacity];
    }
};

int main() {
    Queue q(5);

    q.enqueue("John");
    q.enqueue("Alice");
    q.enqueue("Bob");
    q.display();

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

    q.enqueue("Eve");
    q.display();

    std::cout << "q.empty(): " << q.empty() << "\n";
    std::cout << "q.size(): " << q.size() << "\n";
    std::cout << "q.front(): " << q.front() << "\n";
    std::cout << "q.back(): " << q.back() << "\n";

    return 0;
}
```

```
John Alice Bob
Alice Bob
Alice Bob Eve
q.empty(): 0
q.size(): 3
q.front(): Alice
q.back(): Eve
```

Operation	Output
Enqueue John, Alice, Bob	John Alice Bob
Dequeue	Alice Bob
Enqueue Eve	Alice Bob Eve
Check empty	q.empty(): 0
Get size	q.size(): 3
Get front	q.front(): Alice
Get back	q.back(): Eve

Table 5-3. Queues using Array Implementation

7. Supplementary Activity

```
Added job 3 to the queue.
Added job 4 to the queue.
Added job 5 to the queue.
Processing jobs...
Processing job 1 submitted by Alice with 2 pages.
Processing job 2 submitted by Bob with 3 pages.
Processing job 3 submitted by Charlie with 1 pages.
Processing job 4 submitted by David with 4 pages.
All jobs processed.
```

Analysis:

- The output shows that all five jobs were added to the printer queue successfully.
- The processJobs function then processed each job in the order they were added to the queue, which is the expected behavior of a first-come, first-served queue.
- The output also shows the details of each job, including the job ID, user name, and number of pages, which demonstrates that the Job class is working correctly.
- The final message "All jobs processed" indicates that the printer has finished processing all jobs in the queue.

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;
```

```
class Job {
private:
    int id;
    string userName;
    int numPages;

public:
    Job() : id(0), userName(""), numPages(0) {} // default constructor
    Job(int id, string userName, int numPages) {
        this->id = id;
        this->userName = userName;
        this->numPages = numPages;
    }

    int getId() {
        return id;
    }

    string getUserName() {
        return userName;
    }

    int getNumPages() {
        return numPages;
    }
};

class Printer {
private:
    vector<Job> queue; // use a vector instead of an array
    int front;
    int rear;
    int capacity;

public:
    Printer(int capacity) {
        this->capacity = capacity;
        front = 0;
        rear = 0;
    }

    void addJob(Job job) {
        if (queue.size() == capacity) {
            cout << "Printer queue is full. Cannot add job." << endl;
            return;
        }
        queue.push_back(job);
        rear = queue.size() - 1;
        cout << "Added job " << job.getId() << " to the queue." << endl;
    }
```

```
    void processJobs() {
        cout << "Processing jobs..." << endl;
        while (front != rear) {
            Job currentJob = queue[front];
            cout << "Processing job " << currentJob.getId() << " submitted by " << currentJob.getUserName() << " with " <<
currentJob.getNumPages() << " pages." << endl;
            front++;
        }
        cout << "All jobs processed." << endl;
    }
};

int main() {
    Printer printer(5); // create a printer with a capacity of 5 jobs

    // add jobs to the printer
    printer.addJob(Job(1, "Alice", 2));
    printer.addJob(Job(2, "Bob", 3));
    printer.addJob(Job(3, "Charlie", 1));
    printer.addJob(Job(4, "David", 4));
    printer.addJob(Job(5, "Eve", 2));

    // process all jobs
    printer.processJobs();

    return 0;
}
```

8. Conclusion

In this activity, we learned about the Queue Abstract Data Type (ADT) and its implementation in C++. We created a queue using the C++ Standard Template Library (STL) and developed our own implementation using arrays and linked lists. We also simulated a shared printer scenario using queues. The procedure was well-structured and easy to follow, and the supplementary activity was a good application of the queue ADT. Overall, I think I did well in this activity, but I need to improve my debugging skills to catch errors earlier.

9. Assessment Rubric