Activity No. <5>	
< QUEUES>	
Course Code: CPE010	Program: Computer Engineering
Course Title: Data Structures and Algorithms	Date Performed: 9/11/25
Section: CPE21S4	Date Submitted: 9/11/25
Name(s): Alcantara, Jason P.	Instructor: Engr. Jimlord Quejado

6. Output:

Discussion:

Part: A

Syntax:

```cpp
#include <iostream>
#define SIZE 100

using namespace std;

class Queue {
private:
    int items[SIZE];
    int front, rear;

public:
    Queue() {
        front = -1;
        rear = -1;
    }

    // Enqueue operation
    void enqueue(int value) {
        if (rear == SIZE - 1) {
            cout << "The Queue is Overflow" << endl;
            return;
        }
        if (front == -1) front = 0;
        rear++;
        items[rear] = value;
        cout << "Enqueued: " << value << endl;
    }

    // Dequeue operation
    void dequeue() {
        if (front == -1 || front > rear) {
            cout << "The Queue is Underflow" << endl;
            return;
        }
        cout << "Dequeued: " << items[front] << endl;
        front++;
    }

    // Display the queue
    void display() {
        if (front == -1 || front > rear) {
            cout << "Queue is empty." << endl;
            return;
        }
        cout << "Queue elements: ";
        for (int i = front; i <= rear; i++) {
            cout << items[i] << " ";
        }
        cout << endl;
    }
};
```

```cpp
};

int main() {
    Queue q;

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.display();

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

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

    return 0;
}
```

Output:

```
C:\Users\Joshua\Documents\Discussion_A.exe
Enqueued: 10
Enqueued: 20
Enqueued: 30
Queue elements: 10 20 30
Dequeued: 10
Queue elements: 20 30
Dequeued: 20
Dequeued: 30
The Queue is Underflow

--------------------------------
Process exited after 0.09686 seconds with return value 0
Press any key to continue . . .
```

Part B:

Main cpp:

Discussion_B.cpp | Queue.h

```cpp
#include <iostream>
#include <string>
#include "Queue.h"


int main() {
    Queue<std::string> CPE2154;

    CPE2154.enqueue("Spiderman");
    CPE2154.enqueue("Batman");
    CPE2154.enqueue("Hulk");
    CPE2154.enqueue("Thanos");
    CPE2154.enqueue("Naruto");
    CPE2154.enqueue("Hikaru");

    CPE2154.getRear();
    CPE2154.getFront();

    CPE2154.dequeue();
    CPE2154.getFront();

    CPE2154.display();

    return 0;
}

```

Output:

```
C:\Users\Joshua\Documents\Discussion_B.exe
An empty queue has been created.
Enqueued: Spiderman
Enqueued: Batman
Enqueued: Hulk
Enqueued: Thanos
Enqueued: Naruto
Enqueued: Hikaru
Current Rear: Hikaru
Current Front: Spiderman
Current Front: Batman
Queue contents: Batman Hulk Thanos Naruto Hikaru

--------------------------------
Process exited after 0.09919 seconds with return value 0
Press any key to continue . . .
```

Header File:

Discussion_B.cpp | Queue.h

```cpp
#ifndef QUEUE_H
#define QUEUE_H

#include <iostream>

template <typename T>
class Queue {
private:
    struct Node {
        T data;
        Node* next;
    };

    Node* front;
    Node* rear;

public:
    Queue() : front(nullptr), rear(nullptr) {
        std::cout << "An empty queue has been created.\n";
    }

    // Check if queue is empty
    bool isEmpty() const {
        return front == nullptr;
    }

    // Enqueue
    void enqueue(const T& item) {
        Node* newNode = new Node{ item, nullptr };

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

        std::cout << "Enqueued: " << item << std::endl;
    }

    // Dequeue
    void dequeue() {
        if (isEmpty()) {
            std::cout << "The queue is empty.\n";
            return;
        }

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

        delete temp;

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

Discussion_B.cpp | Queue.h

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

        delete temp;

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

    // Get front
    void getFront() const {
        if (isEmpty()) {
            std::cout << "The queue is empty.\n";
            return;
        }

        std::cout << "Current Front: " << front->data << std::endl;
    }

    // Get rear
    void getRear() const {
        if (isEmpty()) {
            std::cout << "The queue is empty.\n";
            return;
        }

        std::cout << "Current Rear: " << rear->data << std::endl;
    }

    // Display
    void display() const {
        if (isEmpty()) {
            std::cout << "The queue is empty.\n";
            return;
        }

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

    // Destructor
    ~Queue() {
        while (!isEmpty())
            dequeue();
    }
};

#endif
```

Part C:

Main cpp:

Discussion_C.cpp | QUEUE_H.h

```cpp
#include <iostream>
#include "QUEUE_H.h"

int main() {
    Queue<int> x; // CreateQueue()

    x.enqueue(5);
    x.enqueue(3);
    x.enqueue(2);

    x.dequeue(); // Removes 5

    x.enqueue(7);

    // Dequeue the variable a
    if (!x.isEmpty()) {
        int a = x.getFrontValue();
        x.dequeue();
        std::cout << "Dequeued a: " << a << std::endl;
    }

    // Dequeue the variable b
    if (!x.isEmpty()) {
        int b = x.getFrontValue();
        x.dequeue();
        std::cout << "Dequeued b: " << b << std::endl;
    }

    x.display();

    return 0;
}

```

Header File:

Discussion_C.cpp | QUEUE_H.h

```cpp
#ifndef QUEUE_H
#define QUEUE_H

#include <iostream>
#include <stdexcept>

template <typename T>
class Queue {
private:
    struct Node {
        T data;
        Node* next;
    };

    Node* front;
    Node* rear;

public:
    // Constructor
    Queue() : front(nullptr), rear(nullptr) {
        std::cout << "An empty queue has been created.\n";
    }

    // Check if empty
    bool isEmpty() const {
        return front == nullptr;
    }

    // Enqueue
    void enqueue(const T& item) {
        Node* newNode = new Node{ item, nullptr };
        if (isEmpty()) {
            front = rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
        std::cout << "Enqueued: " << item << std::endl;
    }

    // Dequeue
    void dequeue() {
        if (isEmpty()) {
            std::cout << "Queue is empty.\n";
            return;
        }
        Node* temp = front;
        front = front->next;
        delete temp;

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

    // Get front value (without removing)
    T getFrontValue() const {
        if (isEmpty()) {
            throw std::runtime_error("Queue is empty.");
        }
        return front->data;
    }

    // Display queue
    void display() const {
        if (isEmpty()) {
            std::cout << "[Queue is empty]" << std::endl;
            return;
        }

        Node* temp = front;
        std::cout << "Queue: ";
        while (temp != nullptr) {
            std::cout << temp->data << " ";
            temp = temp->next;
        }
        std::cout << "(Front → Back)" << std::endl;
    }

    // Destructor
    ~Queue() {
        while (!isEmpty()) {
            dequeue();
        }
    }
```

```cpp
    // Destructor
    ~Queue() {
        while (!isEmpty()) {
            dequeue();
        }
    }
};

#endif // QUEUE_H

```

Output:

```
C:\Users\Joshua\Documents\Discussion_C.exe
An empty queue has been created.
Enqueued: 5
Enqueued: 3
Enqueued: 2
Enqueued: 7
Dequeued a: 3
Dequeued b: 2
Queue: 7 (Front ? Back)

--------------------------------
Process exited after 0.08925 seconds with return value 0
Press any key to continue . . .
```

Procedure A:

Discussion_C.cpp | QUEUE_H.h | [*] Procedure_A.cpp

```cpp
#include <iostream>
#include <queue>

int main() {
    std::queue<int> x;
    int a, b;

    x.push(5);
    x.push(3);
    x.push(2);
    x.pop();

    x.push(7);

    if (!x.empty()) {
        a = x.front();
        x.pop();
        std::cout << "Dequeued a = " << a << std::endl;
    }

    if (!x.empty()) {
        b = x.front();
        x.pop();
        std::cout << "Dequeued b = " << b << std::endl;
    }

    // Display remaining queue
    std::cout << "Final queue: ";
    std::queue<int> temp = x;
    while (!temp.empty()) {
        std::cout << temp.front() << " ";
        temp.pop();
    }
    std::cout << "(Front → Back)" << std::endl;

    return 0;
}
```

Output:

```
C:\Users\Joshua\Documents\Procedure_A.exe
Dequeued a = 3
Dequeued b = 2
Final queue: 7 (Front ? Back)

--------------------------------
Process exited after 0.1039 seconds with return value 0
Press any key to continue . . .
```

ILO B:

5.1

Syntax:

STL.cpp

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

int main() {
    std::queue<std::string> q;
    std::string students[] = {"Spderman", "Batman", "Hulk", "Superman"};

    // Insert students
    for (auto& name : students) {
        std::cout << "Enqueue: " << name << std::endl;
        q.push(name);
    }

    // Display
    std::cout << "Queue after enqueues: ";
    std::queue<std::string> temp = q;
    while (!temp.empty()) {
        std::cout << temp.front() << " ";
        temp.pop();
    }
    std::cout << "\n";

    // Dequeue the students one by one
    while (!q.empty()) {
        std::cout << "Dequeue: " << q.front() << std::endl;
        q.pop();
    }

    return 0;
}
```

Output:

```
C:\Users\Joshua\Documents\STL.exe
Enqueue: Spderman
Enqueue: Batman
Enqueue: Hulk
Enqueue: Superman
Queue after enqueues: Spderman Batman Hulk Superman
Dequeue: Spderman
Dequeue: Batman
Dequeue: Hulk
Dequeue: Superman

--------------------------------
Process exited after 0.08972 seconds with return value 0
Press any key to continue . . .
```

5.2

Syntax:

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

class CircularQueue {
private:
    int* q_array;
    int capacity;
    int front;
    int back;
    int size;

public:
    CircularQueue(int cap) {
        capacity = cap;
        q_array = new int[capacity];
        front = 0;
        back = capacity - 1;
        size = 0;
    }

    ~CircularQueue() {
        delete[] q_array;
    }

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

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

    void enqueue(int value) {
        if (isFull()) {
            cout << "Queue is full! " << value << endl;
            return;
        }
        back = (back + 1) % capacity;
        q_array[back] = value;
        size++;
        cout << "Enqueued: " << value << endl;
    }

    void dequeue() {
        if (isEmpty()) {
            cout << "Queue is empty! " << endl;
            return;
        }
        cout << "Dequeued: " << q_array[front] << endl;
        front = (front + 1) % capacity;
        size--;
    }

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

int main() {
    CircularQueue q(5);

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.enqueue(40);
    q.display();

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

    q.enqueue(50);
    q.enqueue(60);
    q.enqueue(70);
    q.display();

    return 0;
}
```

Output:

```
C:\Users\Joshua\Documents\ILO_5.2.exe
Enqueued: 10
Enqueued: 20
Enqueued: 30
Enqueued: 40
Queue contents: 10 20 30 40
Dequeued: 10
Dequeued: 20
Queue contents: 30 40
Enqueued: 50
Enqueued: 60
Enqueued: 70
Queue contents: 30 40 50 60 70

--------------------------------
Process exited after 0.1101 seconds with return value 0
Press any key to continue . . .
```

5.3

Syntax:

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

class SimpleQueue {
private:
    int* arr;
    int capacity;
    int front;
    int back;
    int size;

public:
    SimpleQueue(int cap) {
        capacity = cap;
        arr = new int[capacity];
        front = 0;
        back = -1;
        size = 0;
    }

    ~SimpleQueue() {
        delete[] arr;
    }

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

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

    void enqueue(int value) {
        if (isFull()) {
            cout << "Queue is full\n";
            return;
        }
        back = (back + 1) % capacity;
        arr[back] = value;
        size++;
        cout << "Enqueued: " << value << endl;
    }

    void dequeue() {
        if (isEmpty()) {
            cout << "Queue is empty\n";
            return;
        }
        cout << "Dequeued: " << arr[front] << endl;
        front = (front + 1) % capacity;
        size--;
    }

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

int main() {
    SimpleQueue q(5);

    q.enqueue(5);
    q.enqueue(10);
    q.enqueue(15);
    q.display();

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

    q.enqueue(20);
    q.display();

    return 0;
}
```

Output:

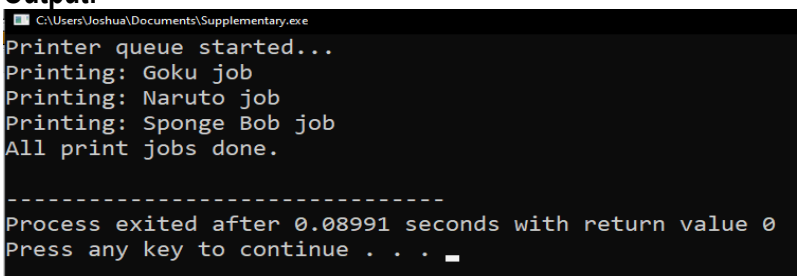
```
C:\Users\Joshua\Documents\ILO_5.3.exe
Enqueued: 5
Enqueued: 10
Enqueued: 15
Queue: 5 10 15
Dequeued: 5
Queue: 10 15
Enqueued: 20
Queue: 10 15 20

--------------------------------
Process exited after 0.209 seconds with return value 0
Press any key to continue . . .
```

7. Supplementary Activity:

Syntax:

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

int main() {
    std::queue<std::string> printerQueue;

    // Adding print jobs
    printerQueue.push("Goku job");
    printerQueue.push("Naruto job");
    printerQueue.push("Sponge Bob job");

    std::cout << "Printer queue started...\n";

    // Process the queue
    while (!printerQueue.empty()) {
        std::cout << "Printing: " << printerQueue.front() << std::endl;
        printerQueue.pop();  // Remove the job after printing
    }

    std::cout << "All print jobs done.\n";

    return 0;
}
```

Output:



8. Conclusion:

In HOA activity I learned how to make a queue by using an array in C++. The queue followed the "FIFO" rule. The rule says that elements are added at the back and taken out at the front. I also learned important things in practice like enqueue, dequeue and is the queue empty or full. This exercise helped me a lot in understanding how queues work and how to take care of data efficiently using array.

9. Assessment Rubric