Activity No. 5	
QUEUES	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 10/07/24**
Section: BSCPE21S1	**Date Submitted:10/07/24**
Name: Kurt Gabriel Anduque	**Instructor: Mrs. Maria Rizzete Sayu**
6. Output	

Source Code:

```
// Online C++ compiler to run C++ program online
#include <iostream>
#include <queue>
#include <string>
using namespace std;

void display (queue <string> names){
  while(!names.empty()){
    cout << names.front()<<endl;
    names.pop();
  };
};

int main() {

    queue <string> names;

    string student[5] = {"kurt","Dale","JP","Khent","Hendricks"};

    for (int counter = 0 ; counter< size(student); counter++){
      cout << "inserting: " <<  student[counter]  << " to queue.."<<endl;
      string name = student[counter];
      names.push(name);
    };
    cout << endl;
    cout << "The name under queue is"
    display(names);

    return 0;
}
```

Output:

```
/tmp/0IZ3R1e8Dh.o
inserting: kurt to queue..
inserting: Dale to queue..
inserting: JP to queue..
inserting: Khent to queue..
inserting: Hendricks to queue..
kurt
Dale
JP
Khent
Hendricks

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

Table 5.1 Queues using C++ STL

Source code:

```cpp
// Online C++ compiler to run C++ program online
#include <iostream>
#include <string>
using namespace std;

class Node{
  public:
      Node* next;
      string datas;
};

void display(Node* head){
    Node* current = head;
  while(current != nullptr){
      cout << current -> datas <<endl;
      current = current -> next;
  };
};




void push(Node*& head,string data){
    Node* last = new Node();
    last -> datas = data;
    last -> next = NULL;

    Node* current = head;
    while(current->next != nullptr){
      current = current -> next;
    };

    current -> next = last;
};

void pop(Node*& head){
    Node* first = head;
    head = head -> next;
    delete first;

};
```

```
46  int main() {
47      Node* head = new Node();
48
49      string data;
50      string insert;
51      int name_counter = 1;
52      int choice;
53
54      cout << "Enter the first name to insert in the queue: ";
55      getline(cin,data);
56      head -> datas = data;
57      head -> next = NULL;
58
59      cout << endl;
60      cout << endl;
61
62
63
64      while(true){
65          cout << "==================== QUEUE NAMES ====================="<<endl;
66          cout <<"[1] = insert element indside the queue"<<endl;
67          cout <<"[2] = delete element inside the queue"<<endl;
68          cout <<"[3] = Check all the datas inside the queue"<<endl;
69          cout<< "choice: ";
70          cin>>choice;
71          cout << "======================================================"<<endl;
72
73          cout <<endl;
74          cout <<endl;
75
```

```
    while(true){
        cout << "=================== QUEUE NAMES ==================="<<endl;
        cout <<"[1] = insert element indside the queue"<<endl;
        cout <<"[2] = delete element inside the queue"<<endl;
        cout <<"[3] = Check all the datas inside the queue"<<endl;
        cout<< "choice: ";
        cin>>choice;
        cout << "====================================================="<<endl;

        cout <<endl;
        cout <<endl;



        if(choice == 1){
            cin.ignore();
            cout << "Enter the name to be inserted inside the queue: ";
            getline(cin,insert);
            push(head,insert);
            cout << "Updated elements inside the queue"<<endl;
            cout <<endl;
            cout <<endl;
            display(head);
        }else if (choice == 2){
            pop(head);
            cout << "Updated elements inside the queue"<<endl;
            cout <<endl;
            cout <<endl;
            display(head);

        }else if(choice == 3){
            cout <<endl;
            cout <<endl;
             display(head);
        }else{
            cout << "You enter a wrong value!!!";

        };



    };



    return 0;
}
```

Output:

```
Enter the first name to insert in the queue: kurt


================== QUEUE NAMES ==================
[1] = insert element indside the queue
[2] = delete element inside the queue
[3] = Check all the datas inside the queue
choice: 1
================================================


Enter the name to be inserted inside the queue: dale
Updated elements inside the queue


kurt
dale
================== QUEUE NAMES ==================
[1] = insert element indside the queue
[2] = delete element inside the queue
[3] = Check all the datas inside the queue
choice: 1
================================================


Enter the name to be inserted inside the queue: hendricks
Updated elements inside the queue


kurt
dale
hendricks
================== QUEUE NAMES ==================
[1] = insert element indside the queue
[2] = delete element inside the queue
[3] = Check all the datas inside the queue
choice: 2
================================================


Updated elements inside the queue


dale
hendricks
================== QUEUE NAMES ==================
[1] = insert element indside the queue
[2] = delete element inside the queue
[3] = Check all the datas inside the queue
choice: 2
```

```
Updated elements inside the queue


dale
hendricks
==================== QUEUE NAMES ====================
[1] = insert element indside the queue
[2] = delete element inside the queue
[3] = Check all the datas inside the queue
choice: 2
=====================================================


Updated elements inside the queue


hendricks
==================== QUEUE NAMES ====================
[1] = insert element indside the queue
[2] = delete element inside the queue
[3] = Check all the datas inside the queue
choice:
```

Table 5.2 Queues using Linked List Implementation

Source Code:

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

class Queue {
private:
    int* arr;
    int capacity;
    int q_front;
    int q_back;
    int count;
    int counter;

public:

    Queue(int size = 100) {
        arr = new int[size];
        capacity = size;
        q_front = 0;
        q_back = -1;
        count = 0;
    }


    ~Queue() {
        delete[] arr;
    }


    void Enqueue(int item) {
        if (count == capacity) {
            cout << "Queue overflow\n";
            return;
        }
        q_back = (q_back + 1) % capacity;
        arr[q_back] = item;

        count++;
    }


    void Dequeue() {
        if (Empty()) {
            cout << "Queue underflow\n";
            return;
        }
        q_front = (q_front + 1) % capacity;

        count--;
    }


```

```
    int Front() {
        if (Empty()) {
            cout << "Queue is empty\n";
            return -1;
        }
        return arr[q_front];
    }


    int Back() {
        if (Empty()) {
            cout << "Queue is empty\n";
            return -1;
        }
        return arr[q_back];
    }


    bool Empty() {
        return count == 0;
    }


    int Size() {
        return count;
    }


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

    void Display() {
    if (Empty()) {
        cout << "Queue is empty\n";
        return;
    }

    int i = q_front;
    for (int counter1 = 0; counter1 < count; counter1++) {
        cout << arr[i] << " ";
        i = (i + 1) % capacity;  // Move to the next index, with wrap-arou
        }
    cout << endl;
    }

};
```

```
int main() {
    Queue q(5);

    q.Enqueue(10);
    q.Enqueue(20);
    q.Enqueue(30);
    q.Enqueue(40);

    cout << "Front element is: " << q.Front() << endl;
    cout << "Back element is: " << q.Back() << endl;

    q.Dequeue();
    cout << "After dequeue, front element is: " << q.Front() << endl;

    cout << "Queue size is: " << q.Size() << endl;

    q.Clear();
    cout << "Queue cleared. Is it empty? " << (q.Empty() ? "Yes" : "No") << endl;

    cout << endl;
    cout <<endl;
    cout << " INSERTING THE QUEUE AGAIN"<<endl;
    q.Enqueue(10);
    q.Enqueue(20);
    q.Enqueue(30);
    q.Enqueue(40);

    cout << "Display queue: ";
    q.Display();
    return 0;
}
```

Output:

```
Front element is: 10
Back element is: 40
After dequeue, front element is: 20
Queue size is: 3
Queue cleared. Is it empty? Yes


 INSERTING THE QUEUE AGAIN
Display queue: 10 20 30 40


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

Table 5.3 Queues using Array Implementation

7. Supplementary Activity

Source Code:

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

using namespace std;


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

    Job(int id, string user, int pages) {
        jobID = id;
        userName = user;
        numPages = pages;
        next = nullptr;
    }
};


class Printer {
private:
    Job* front;
    Job* rear;

public:

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

    void addJob(int id, string user, int pages) {
        Job* newJob = new Job(id, user, pages);


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

        cout << "Added Job ID: " << id << ", User: " << user << ", Pages: " << pages << endl;
    }


```

```
    void displayQueue() {
        if (front == nullptr) {
            cout << "No jobs to display." << endl;
            return;
        }

        Job* current = front;
        cout << "Currently in queue:\n";
        while (current != nullptr) {
            cout << current->userName << endl;
            current = current->next;
        }
    }


    void processJob() {

        if (front == nullptr) {
            cout << "No jobs to process." << endl;
            return;
        }


        cout << "Processing...\n";
        displayQueue();


        Job* jobToProcess = front;
        cout << "Processing Job ID: " << jobToProcess->jobID << ", User: " << jobToProcess->userName << ", Pages: " <<
            jobToProcess->numPages << endl;

        front = front->next;

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

        delete jobToProcess;


        cout << "Job done!\n\n";
    }


    bool hasJobs() {
        return front != nullptr;
    }
};
```

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


    printer.addJob(1, "Kurt", 5);
    printer.addJob(2, "Dale", 10);
    printer.addJob(3, "Hendricks", 2);


    while (printer.hasJobs()) {
        printer.processJob();
    }

    return 0;
}
```

OUTPUT:

```
Added Job ID: 1, User: Kurt, Pages: 5
Added Job ID: 2, User: Dale, Pages: 10
Added Job ID: 3, User: Hendricks, Pages: 2
Processing...
Currently in queue:
Kurt
Dale
Hendricks
Processing Job ID: 1, User: Kurt, Pages: 5
Job done!

Processing...
Currently in queue:
Dale
Hendricks
Processing Job ID: 2, User: Dale, Pages: 10
Job done!

Processing...
Currently in queue:
Hendricks
Processing Job ID: 3, User: Hendricks, Pages: 2
Job done!
```

Why did I used linked list ?

- In my code I implement the linked list Where the printer's job queue behaves as a first-in, first-out also known as FIFO.with the help of linked list it allows for both adding to the rear and removing from the front in constant time (O(1)), whereas an array requires shifting elements when jobs are processed, making it less efficient.

8. Conclusion
This activity taught me the fundamentals of queue implementation in a C++ program. I was able to understand the three different ways to implement a queue: using STL C++, linked lists, and arrays. I carefully followed the explanation of the various methods, such as dequeue and enqueue, and always remember that a queue is "First In, First Out". Then I was able to apply what I had learned to a simple program that simulates a printer. I conclude that a queue is an important data structure designed for specific scenarios in which the FIFO principle is required to make work efficient.
9. Assessment Rubric