Submitted By: Alisangco, Arnold Nathan M.

Course & Section: BSIT-WMA (AW-12)

CCS0015 – Queue Activity

main.cpp

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

using namespace std;

// Node Class
class Node {
public:
  string data;
  Node *next;

  Node(string value) {
    data = value;
    next = nullptr;
  }
};

// HospitalQueue Class
class HospitalQueue {
private:
  Node *front;
  Node *back;

public:
  HospitalQueue() {
    front = nullptr;
    back = nullptr;
  }

  // Boolean function to check if the queue is empty
```

main.cpp

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

// Function to add a patient to the queue
void enqueue(string value) {
  Node *AddedNode = new Node(value);
  if (isEmpty()) {
    front = back = AddedNode;
  } else {
    back->next = AddedNode;
    back = AddedNode;
  }
  cout << "Next In Line: Added " << value << endl;
}

// Function to remove a patient from the back of the queue
void dequeue() {
  if (isEmpty()) {
    cout << "No Patients In Line..." << endl;
  } else {
    Node *temp = front;
    front = front->next;
    cout << "Removed Patient: " << temp->data << endl;
    delete temp;
    if (front == nullptr) {
      back = nullptr;
    }
  }
}

// Function to display the patient at the front of the queue
```

```
61    string peek() {
62      if (!isEmpty()) {
63        return front->data;
64      } else {
65        cout << "No Patients In Line..." << endl;
66        return "";
67      }
68    }
69
70    // Function to display all patients in the queue
71    void display() {
72      if (isEmpty()) {
73        cout << "No Patients In Line..." << endl;
74        return;
75      }
76      cout << "Current Patients:" << endl;
77      Node *temp = front;
78      while (temp != nullptr) {
79        cout << temp->data << endl;
80        temp = temp->next;
81      }
82      cout << endl;
83    }
84  };
85
86  int main() {
87    HospitalQueue patientQueue;
88    char choice;
89    char actionChoice;
90    string patientName;
```

```
91    string queueActions[] = {"Add New Patient", "Check Status", "Display Queue",
92                             "Remove Latest Patient",
93                             "Exit"}; // Array of queue actions
94
95    cout << "\n*-=-*-=-*-=-*-=-*-=-*-=-*" << endl;
96    cout << "|    HOSPITAL QUEUE!    |" << endl;
97    cout << "*-=-*-=-*-=-*-=-*-=-*-=-*" << endl;
98
99    cout << endl;
100   patientQueue.enqueue("Walter White");
101   patientQueue.enqueue("Jesse Pinkman");
102   patientQueue.enqueue("Mike Ehrmantraut");
103
104   cout << endl;
105   patientQueue.display();
106
107   // Main loop for demonstrating the example queue of the program
108   for (int i = 0; i < 3; i++) {
109     cout << "Currently Serving: " << patientQueue.peek() << endl;
110     cout << endl;
111     patientQueue.dequeue();
112     cout << endl;
113     patientQueue.display();
114   }
115
116   cout << "\n*=-*-=-*-=-*-=-*-=-*-=-*" << endl;
117   cout << "|    MODIFY QUEUE?    |" << endl;
118   cout << "*=-*-=-*-=-*-=-*-=-*-=-*" << endl;
119
120   // Loop to allow the user to modify the queue
121   do {
```

```
122     cout << "\n'Y' to modify queue, 'N' to exit: ";
123     cin >> choice;
124     choice = toupper(choice);
125     if (choice != 'Y' && choice != 'N') {
126       cout << "\nError! Input not recognized. Try again." << endl;
127     }
128   } while (choice != 'Y' && choice != 'N');
129
130   if (choice == 'N') {
131     cout << "\nThank you for using the program!" << endl;
132   } else {
133     cout << endl;
134     int actionCounter = 1;
135     for (const string &action : queueActions) {
136       cout << actionCounter << ". " << action << endl;
137       actionCounter++;
138     }
139     do {
140       do {
141         cout << "\nChoice: ";
142         cin >> actionChoice;
143         if (actionChoice < '1' || actionChoice > '5') {
144           cout << "\nError! Action not recognized. Try again." << endl;
145         }
146       } while (actionChoice < '1' || actionChoice > '5');
147
148       // Switch statement to handle the user's choice
149       switch (actionChoice) {
150       case '1':
151         cout << "\nEnter Patient Name: ";
152         getline(cin >> ws, patientName);
```

```
153             cout << endl;
154             patientQueue.enqueue(patientName);
155             break;
156         case '2':
157             cout << endl;
158             cout << "Currently Serving: " << patientQueue.peek() << endl;
159             break;
160         case '3':
161             cout << endl;
162             patientQueue.display();
163             break;
164         case '4':
165             cout << endl;
166             patientQueue.dequeue();
167             break;
168         case '5':
169             cout << "\nThank you for using the program!" << endl;
170             break;
171         default:
172             cout << "\nThank you for using the program!" << endl;
173             break;
174         }
175       } while (actionChoice != '5');
176     }
177
178     return 0;
179   }
180
181   /*
182
```

```
    } while (actionChoice != '5');
  }

  return 0;
}

/*

Program Description:
- Overall, automated systems heavily utilize queues in the real world to keep
track of various things, especially customers in business-related matters.
- This program emulates a very basic interpretation of a hospital queue used for
keeping patients in line.
- There are 5 main functions included in the class "HospitalQueue," which
represents the inner workings of how the queue operates via the C++ stack. These
functions are: enqueue, dequeue, peek, display, and isEmpty.
- Enqueue adds a new patient to the front of the stack list.
- Dequeue removes the patient at the back of the list (because they're the first
to go in order once they've been checked out).
- Peek checks the current status of the queue and the current patient.
- Display displays the current patients in the queue.
- isEmpty checks if the queue is empty, which determines if the program will
take action or not.
- Only names (as strings) are available in the current program when it comes to
inputting values.
- Finally, the do-whiles and for loops automate certain processes that involve
functions or act as contingencies for any user misinput.

*/

```

```
*-=-*-=-*-=-*-=-*-=-*-=-*
|    HOSPITAL QUEUE!    |
*-=-*-=-*-=-*-=-*-=-*-=-*

Next In Line: Added Walter White
Next In Line: Added Jesse Pinkman
Next In Line: Added Mike Ehrmantraut

Current Patients:
Walter White
Jesse Pinkman
Mike Ehrmantraut

Currently Serving: Walter White

Removed Patient: Walter White

Current Patients:
Jesse Pinkman
Mike Ehrmantraut

Currently Serving: Jesse Pinkman

Removed Patient: Jesse Pinkman

Current Patients:
Mike Ehrmantraut

Currently Serving: Mike Ehrmantraut

Removed Patient: Mike Ehrmantraut

No Patients In Line...

*=-*-=-*-=-*-=-*-=-*-=-*
|    MODIFY QUEUE?    |
*=-*-=-*-=-*-=-*-=-*-=-*
```

```
'Y' to modify queue, 'N' to exit: a

Error! Input not recognized. Try again.

'Y' to modify queue, 'N' to exit: y

1. Add New Patient
2. Check Status
3. Display Queue
4. Remove Latest Patient
5. Exit

Choice: 1

Enter Patient Name: Nathan Alisangco

Next In Line: Added Nathan Alisangco

Choice: 2

Currently Serving: Nathan Alisangco

Choice: 3

Current Patients:
Nathan Alisangco


Choice: 4

Removed Patient: Nathan Alisangco

Choice: 2

Currently Serving: No Patients In Line...
```

```
Choice: 3

No Patients In Line...

Choice: 5

Thank you for using the program!
```