LAPORAN PRAKTIKUM

PERTEMUAN 3

08_QUEUE

Nama :

Ilham Lii Assidaq (2311104068)

Dosen :

Wahyu Andi Saputra S.Pd., M.Eng.

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

1. Mahasiswa mampu menjelaskan definisi dan konsep dari queue
2. Mahasiswa mampu menerapkan operasi tambah, menghapus pada queue
3. Mahasiswa mampu menerapkan operasi tampil data pada queue

II. TOOL

Dev C++

III. DASAR TEORI

Queue adalah struktur data yang menggunakan prinsip FIFO, atau FirstIn First-Out, di mana elemen pertama yang dimasukkan akan dikeluarkan. Sisi depan menunjuk elemen pertama dan sisi belakang menunjuk elemen terakhir.Enqueue (menambahkan elemen di belakang), dequeue (menghapus elemen didepan), peek (mengambil elemen depan tanpa menghapusnya), isEmpty(mengecek apakah queue kosong), isFull (mengecek apakah queue penuh), dan size adalah operasi dasar yang dilakukan pada queue. Queue memungkinkan penambahan dan penghapusan elemen di ujung yang berbeda dari stack yang menggunakan metode LIFO (Last-In First-Out). Ini membuatnya ideal untuk aplikasi seperti sistem penjadwalan atau antrian pelanggan.

IV. GUIDED

1. GUIDED 0031

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

struct Node {
    string name;
    string nim;
    Node* next;
};

struct Queue {
    Node* front;
    Node* rear;
    int count;
};

Queue* createQueue() {
    Queue* queue = new Queue();
    queue->front = NULL;
    queue->rear = NULL;
    queue->count = 0;
    return queue;
}

bool isEmpty(Queue* queue) {
    return (queue->front == NULL);
}

void enqueue(Queue* queue, string name, string nim) {
    Node* newNode = new Node();
    newNode->name = name;
    newNode->nim = nim;
    newNode->next = NULL;

    if (isEmpty(queue)) {
        queue->front = newNode;
        queue->rear = newNode;
    } else {
        if (nim < queue->front->nim) {
            newNode->next = queue->front;
            queue->front = newNode;
        } else {
            Node* current = queue->front;
            while (current->next != NULL && current->next->nim < nim) {
                current = current->next;
            }
            newNode->next = current->next;
            current->next = newNode;

```

```
            if (newNode->next == NULL) {
                queue->rear = newNode;
            }
        }
    }
    queue->count++;
    cout << "Mahasiswa " << name << " (NIM: " << nim << ") ditambahkan ke antrian." << endl;
}

void dequeue(Queue* queue) {
    if (isEmpty(queue)) {
        cout << "Antrian kosong." << endl;
        return;
    }
    Node* temp = queue->front;
    cout << "Mahasiswa " << temp->name << " (NIM: " << temp->nim << ") dikeluarkan dari antrian." << endl;
    queue->front = queue->front->next;
    delete temp;
    queue->count--;

    if (queue->front == NULL) {
        queue->rear = NULL;
        cout << "Antrian telah dikosongkan." << endl;
    }
}

void viewQueue(Queue* queue) {
    Node* current = queue->front;
    cout << "Data antrian mahasiswa:" << endl;
    int index = 1;
    while (current != NULL) {
        cout << index++ << ". Nama: " << current->name << ", NIM: " << current->nim << endl;
        current = current->next;
    }
    cout << "Jumlah antrian = " << queue->count << endl;
}

int main() {
    Queue* queue = createQueue();
    int n;

    cout << "Masukkan jumlah mahasiswa yang ingin ditambahkan: ";
    cin >> n;
    cin.ignore();

    for (int i = 0; i < n; i++) {
        string name, nim;
        cout << "Masukkan nama mahasiswa: ";
        getline(cin, name);
```

```
            cout << "Masukkan NIM mahasiswa: ";
            getline(cin, nim);
            enqueue(queue, name, nim);
        }

        viewQueue(queue);

        while (!isEmpty(queue)) {
            dequeue(queue);
            viewQueue(queue);
        }

        return 0;
    }

```

2. GUIDED2

```cpp
#include <iostream>

using namespace std;

class Node {
public:
    int data;
    Node* next;

    Node(int value) {
        data = value;
        next = NULL;
    }
};

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

public:
    Queue() {
        front = rear = NULL;
    }

    bool isEmpty() {
        return front == NULL;
    }

    void enqueue(int x) {
        Node* newNode = new Node(x);
        if (isEmpty()) {
            front = rear = newNode;
            return;
        }
        rear->next = newNode;
        rear = newNode;
    }

    void dequeue() {
        if (isEmpty()) {
           cout << "Queue Underflow\n";
            return;
        }
        Node* temp = front;
        front = front->next;
        delete temp;
        if (front == NULL)
            rear = NULL;
    }

    int peek() {
        if (!isEmpty()) {
            return front->data;
        }
       cout << "Queue is empty\n";
        return -1;
    }

    void display() {
        if (isEmpty()) {
           cout << "Queue is empty\n";
            return;
        }
        Node* current = front;
        while (current) {
           cout << current->data << " ";
            current = current->next;
        }
       cout << "\n";
    }
};

int main() {
    Queue q;

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

   cout << "Queue elements: ";
    q.display();

   cout << "Front element: " << q.peek() << "\n";

    q.dequeue();
   cout << "After dequeuing, queue elements: ";
    q.display();

    return 0;
}
```

3.GUIDED3

```
#include<iostream>

using namespace std;

const int maksimalQueue = 5;
int front = 0;
int back = 0;
string queueTeller[5];

bool isFull() {
   if (back == maksimalQueue) {
      return true;
   } else {
      return false;
   }
}

bool isEmpty() {
   if (back == 0) {
      return true;
   } else {
      return false;
   }
}

void enqueueAntrian(string data) {
   if (isFull()) {
      cout << "Antrian penuh" << endl;
   } else {
      if (isEmpty()) {
         queueTeller[0] = data;
         front++;
         back++;
      } else {
         queueTeller[back] = data;
         back++;
      }
   }
}

void dequeueAntrian() {
   if (isEmpty()) {
      cout << "Antrian kosong" << endl;
   } else {
      for (int i = 0; i < back; i++) {
         queueTeller[i] = queueTeller[i + 1];
      }
      back--;
   }
}
```

```
int countQueue() {
   return back;
}

void clearQueue() {
   if (isEmpty()) {
      cout << "Antrian kosong" << endl;
   } else {
      for (int i = 0; i < back; i++) {
         queueTeller[i] = "";
      }
      back = 0;
      front = 0;
   }
}

void viewQueue() {
   cout << "Data antrian teller:" << endl;
   for (int i = 0; i < maksimalQueue; i++) {
      if (queueTeller[i] != "") {
         cout << i + 1 << ". " << queueTeller[i] << endl;
      } else {
         cout << i + 1 << ". (kosong)" << endl;
      }
   }
}

int main() {
   enqueueAntrian("Rizki Ridho");
   enqueueAntrian("Marceng");

   viewQueue();
   cout << "Jumlah antrian = " << countQueue() << endl;

   dequeueAntrian();
   viewQueue();
   cout << "Jumlah antrian = " << countQueue() << endl;

   clearQueue();
   viewQueue();
   cout << "Jumlah antrian = " << countQueue() << endl;

   return 0;
}
```

V. UNGUIDED

1. UNGUIDED1

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

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

struct Queue {
    Node* front;
    Node* rear;
    int count;
};

Queue* createQueue() {
    Queue* queue = new Queue();
    queue->front = nullptr;
    queue->rear = nullptr;
    queue->count = 0;
    return queue;
}

bool isEmpty(Queue* queue) {
    return (queue->front == nullptr);
}

void enqueue(Queue* queue, string data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = nullptr;

    if (isEmpty(queue)) {
        queue->front = newNode;
        queue->rear = newNode;
    } else {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
    queue->count++;
    cout << data << " ditambahkan ke antrian." << endl;
}

void dequeue(Queue* queue) {
    if (isEmpty(queue)) {
        cout << "Antrian kosong." << endl;
        return;
    }
    Node* temp = queue->front;
    cout << temp->data << " dikeluarkan dari antrian." << endl;
    queue->front = queue->front->next;
```

```
    delete temp;
    queue->count--;

    if (queue->front == nullptr) {
        queue->rear = nullptr;
        cout << "Antrian telah dikosongkan." << endl;
    }
}

void viewQueue(Queue* queue) {
    Node* current = queue->front;
    cout << "Data antrian :" << endl;
    int index = 1;
    while (current != nullptr) {
        cout << index++ << ". " << current->data << endl;
        current = current->next;
    }
    cout << "Jumlah antrian = " << queue->count << endl;
}

int main() {
    Queue* queue = createQueue();
    enqueue(queue, "sintaeyong");
    enqueue(queue, "Jeje");
    enqueue(queue, "Rodri");
    viewQueue(queue);
    dequeue(queue);
    viewQueue(queue);
    dequeue(queue);
    dequeue(queue);
    return 0;
}
```

2. UNGUIDED2

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

struct Node {
    string name;
    string nim;
    Node* next;
};

struct Queue {
    Node* front;
    Node* rear;
    int count;
};

Queue* createQueue() {
    Queue* queue = new Queue();
    queue->front = nullptr;
    queue->rear = nullptr;
    queue->count = 0;
    return queue;
}

bool isEmpty(Queue* queue) {
    return (queue->front == nullptr);
}

void enqueue(Queue* queue, string name, string nim) {
    Node* newNode = new Node();
    newNode->name = name;
    newNode->nim = nim;
    newNode->next = nullptr;

    if (isEmpty(queue)) {
        queue->front = newNode;
        queue->rear = newNode;
    } else {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
    queue->count++;
    cout << "Mahasiswa " << name << " [NIM: " << nim << "] ditambahkan ke antrian." << endl;
}

void dequeue(Queue* queue) {
    if (isEmpty(queue)) {
        cout << "Antrian kosong." << endl;
        return;
    }
```

```
52.     Node* temp = queue->front;
53.     cout << "Mahasiswa " << temp->name << " [NIM: " << temp->nim << "]
    dikeluarkan dari antrian." << endl;
54.     queue->front = queue->front->next;
55.     delete temp;
56.     queue->count--;
57. 
58.     if (queue->front == nullptr) {
59.         queue->rear = nullptr;
60.         cout << "Antrian telah dikosongkan." << endl;
61.     }
62. }
63. 
64. void viewQueue(Queue* queue) {
65.     Node* current = queue->front;
66.     cout << "Data antrian mahasiswa:" << endl;
67.     int index = 1;
68.     while (current != nullptr) {
69.         cout << index++ << ". Nama: " << current->name << ", NIM: " <<
    current->nim << endl;
70.         current = current->next;
71.     }
72.     cout << "Jumlah antrian = " << queue->count << endl;
73. }
74. 
75. int main() {
76.     Queue* queue = createQueue();
77.     enqueue(queue, "Ilham Lii Assidaq", "231110468");
78.     enqueue(queue, "Shin Tae Yong", "2311104099");
79.     viewQueue(queue);
80.     dequeue(queue);
81.     viewQueue(queue);
82.     dequeue(queue);
83.     dequeue(queue);
84.     return 0;
```

85. UNGUIDED 3

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

struct Node {
   string name;
   string nim;
   Node* next;
};

struct Queue {
   Node* front;
   Node* rear;
   int count;
};

Queue* createQueue() {
   Queue* queue = new Queue();
   queue->front = nullptr;
   queue->rear = nullptr;
   queue->count = 0;
   return queue;
}

bool isEmpty(Queue* queue) {
   return (queue->front == nullptr);
}

void enqueue(Queue* queue, string name, string nim) {
   Node* newNode = new Node();
   newNode->name = name;
   newNode->nim = nim;
   newNode->next = nullptr;

   if (isEmpty(queue)) {
      queue->front = newNode;
      queue->rear = newNode;
   } else {
      if (nim < queue->front->nim) {
         newNode->next = queue->front;
         queue->front = newNode;
      } else {
         Node* current = queue->front;
         while (current->next != nullptr && current->next->nim < nim) {
            current = current->next;
         }
         newNode->next = current->next;
         current->next = newNode;

         if (newNode->next == nullptr) {
            queue->rear = newNode;
```

```
            }
        }
    }
    queue->count++;
    cout << "Mahasiswa " << name << " (NIM: " << nim << ") ditambahkan ke antrian."
<< endl;
}

void dequeue(Queue* queue) {
    if (isEmpty(queue)) {
        cout << "Antrian kosong." << endl;
        return;
    }
    Node* temp = queue->front;
    cout << "Mahasiswa " << temp->name << " (NIM: " << temp->nim << ")
dikeluarkan dari antrian." << endl;
    queue->front = queue->front->next;
    delete temp;
    queue->count--;

    if (queue->front == nullptr) {
        queue->rear = nullptr;
        cout << "Antrian telah dikosongkan." << endl;
    }
}

void viewQueue(Queue* queue) {
    Node* current = queue->front;
    cout << "Data antrian mahasiswa:" << endl;
    int index = 1;
    while (current != nullptr) {
        cout << index++ << ". Nama: " << current->name << ", NIM: " << current->nim
<< endl;
        current = current->next;
    }
    cout << "Jumlah antrian = " << queue->count << endl;
}

int main() {
    Queue* queue = createQueue();
    int n;

    cout << "Masukkan jumlah mahasiswa yang ingin ditambahkan: ";
    cin >> n;
    cin.ignore();

    for (int i = 0; i < n; i++) {
        string name, nim;
        cout << "Masukkan nama mahasiswa: ";
        getline(cin, name);
        cout << "Masukkan NIM mahasiswa: ";
        getline(cin, nim);
        enqueue(queue, name, nim);
```

```
    }

    viewQueue(queue);

    while (!isEmpty(queue)) {
        dequeue(queue);
        viewQueue(queue);
    }

    return 0;
}
```

KESIMPULAN

Melalui praktik ini, saya telah memahami konsep dasar struktur data queue dan cara kerjanya. Struktur ini beroperasi berdasarkan prinsip FIFO (First-In, First-Out) serta melibatkan operasi utama seperti enqueue dan dequeue. Selain mengasah logika pemrograman melalui tugas modifikasi program untuk mengelola prioritas tertentu, praktik ini juga memperkenalkan penerapan queue menggunakan array dan linked list. Setelah menyelesaikan praktik ini, saya diharapkan mampu menerapkan konsep queue untuk menyelesaikan masalah yang membutuhkan pengelolaan data secara terorganisir dan efisien.