Telkom University Purwokerto

Aturan Praktikum Struktur Data

1. **Akun GitHub:** Setiap praktikan wajib memiliki akun GitHubyang aktif dan digunakan selama praktikum berlangsung.
2. **Invite Collaborator:** Setiap praktikan diwajibkan untuk menambahkan collaborator di setiap repository
    a. Asisten Praktikum: AndiniNH
    b. AsistenPraktikum:4ldiputra
3. **Repository Praktikum:** Setiap praktikan diwajibkan untuk membuat satu repository di GitHub yang akan digunakan untuk seluruh tugas dan laporan praktikum. Repository ini harus diatur dengan rapi dan sesuai dengan instruksi yang akan diberikan di lampiran.
4. **Penamaan Folder:** Penamaan folderdalam repositoryakan dibahas secara rinci di lampiran. Praktikan wajib mengikuti aturan penamaan yang telah ditentukan.

Nomor	Pertemuan	Penamaan
1	Penganalan Bahasa C++ Bagian Pertama	01_Pengenalan_CPP_Bagian_1
2	Pengenalan Bahasa C++ Bagian Kedua	02_Pengenalan_CPP_Bagian_2
3	Abstract Data Type	03_Abstract_Data_Type
4	Single Linked List Bagian Pertama	04_Single_Linked_List_Bagian_1
5	Single Linked List Bagian Kedua	05_Single_Linked_List_Bagian_2
6	Double Linked List Bagian Pertama	06_Double_Linked_List_Bagian_1
7	Stack	07_Stack
8	Queue	08_Queue
9	Assessment Bagian Pertama	09_Assessment_Bagian_1
10	Tree Bagian Pertama	10_Tree_Bagian_1
11	Tree Bagian Kedua	11_Tree_Bagian_2
12	Asistensi Tugas Besar	12_Asistensi_Tugas_Besar
13	Multi Linked List	13_Multi_Linked_List
14	Graph	14_Graph
15	Assessment Bagian Kedua	15_Assessment_Bagian_2
16	Tugas Besar	16_Tugas_Besar

5. **Jam Praktikum:**
    - Jam masuk praktikum adalah **1 jam lebih lambat** dari jadwal yang tercantum. Sebagai contoh, jika jadwalpraktikumadalah pukul 06.30 - 09.30, maka aturan praktikum akan diatur sebagai berikut:
        - **06.30 - 07.30**: Waktuini digunakan untuk **Tugas Praktikum dan Laporan Praktikum** yang dilakukan di luar laboratorium.
        - **07.30 - 0G.30**: Sesi ini mencakup **tutorial, diskusi, dan kasus problem- solving**. Kegiatan ini berlangsung di dalam laboratorium dengan alokasi waktu sebagai berikut:
            - **60 menit pertama**: Tugas terbimbing.
            - **60 menit kedua**: Tugas mandiri.
6. **Pengumpulan Tugasn Pendahuluan:** Tugas Pendahuluan (TP) wajib dikumpulkan melalui GitHub sesuai dengan format berikut:

    nama_repo/nama_pertemuan/TP_Pertemuan_Ke.md

    Sebagai contoh:

    STD_Yudha_Islalmi_Sulistya_XXXXXXXX/01_Running_Modul/TP_01.md

7. **Pengecekan Tugas Pendahuluan:** Pengumpulan laporan praktikum akan diperiksa **1 hari sebelum praktikum selanjutnya** dimulai. Pastikan tugas telah diunggah tepat waktu untuk menghindari sanksi.

LAPORAN PRAKTIKUM
MODUL 8
QUEUE

Disusun Oleh :

Izzaty Zahara Br Barus – 2311104052

Kelas :

SE-07-02

Dosen :

Wahyu Andi Saputra, S.pd,M.Eng

PROGRAM STUDI SOFTWARE ENGINEERING
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. LANDASAN TEORI

Queue adalah struktur data yang menggunakan prinsip FIFO (First In, First Out), yaitu elemen yang pertama masuk akan menjadi elemen pertama yang keluar, seperti antrian pada kehidupan sehari-hari. Queue terdiri dari dua pointer utama: front yang menunjuk elemen pertama dan rear yang menunjuk elemen terakhir. Berbeda dengan stack yang menggunakan prinsip LIFO (Last In, First Out), pada queue, penambahan elemen dilakukan di ujung belakang (enqueue), sedangkan penghapusan dilakukan di ujung depan (dequeue). Beberapa operasi pada queue meliputi enqueue untuk menambah data, dequeue untuk menghapus data, peek untuk melihat data tanpa menghapus, isEmpty untuk mengecek apakah queue kosong, isFull untuk memeriksa apakah penuh, dan size untuk menghitung jumlah elemen. Queue dapat diimplementasikan menggunakan array atau linked list dan sering digunakan dalam simulasi antrean, seperti di kasir atau layanan pelanggan.

III. GUIDE

1. Guide1

   a. Syntax

```
#include <iostream>
#define MAX 100

using namespace std;

class Queue {
```

```
private:
    int front, rear;
    int arr[MAX];
public:

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



    bool isFull() {
        return rear == MAX - 1;
    }



    bool isEmpty() {
        return front == -1 || front > rear;
    }



    void enqueue(int x) {
        if (isFull()) {
            cout << "Queue Overflow\n";
            return;
        }
        if (front == -1) front = 0;
        arr[++rear] = x;
    }



    void dequeue() {
        if (isEmpty()) {
```

```
            cout << "Queue Underflow\n";
            return;
        }
        front++;
    }

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


    void display() {
        if (isEmpty()) {
            cout << "Queue is empty\n";
            return;
        }
        for (int i = front; i <= rear; i++) {
            cout << arr[i] << " ";
        }
        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";

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

    return 0;
}
```

b. Penjelasan syntax guide1 :

Program di atas mendefinisikan kelas Queue untuk mengimplementasikan struktur data queue menggunakan array dengan kapasitas maksimal 100 elemen. Queue memiliki dua pointer: front untuk elemen pertama dan rear untuk elemen terakhir. Fungsi-fungsi penting yang ada meliputi enqueue() untuk menambahkan elemen ke belakang queue, dequeue() untuk menghapus elemen dari depan queue, peek() untuk melihat elemen di depan tanpa menghapusnya, isFull() untuk mengecek apakah queue penuh, isEmpty() untuk mengecek apakah queue kosong, dan display() untuk mencetak semua elemen queue. Di fungsi utama, queue diuji dengan menambahkan tiga elemen (10, 20, 30), mencetak elemen di depan, dan menampilkan isi queue sebelum serta sesudah proses penghapusan (dequeue).

c. Output

```
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 10 20 30
```

2. Guide2

a. Syntax :

```
#include <iostream>

using namespace std;

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

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

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

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

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

    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 == nullptr)
        rear = nullptr;
}

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;
}
```

b. Penjelasan Syntax guide2:

Program di atas adalah implementasi Queue menggunakan linked list. Kelas Node digunakan untuk merepresentasikan elemen queue, dengan atribut data untuk menyimpan nilai dan next untuk menunjuk elemen berikutnya. Kelas Queue memiliki pointer front untuk menunjuk elemen pertama dan rear untuk elemen terakhir. Fungsi enqueue() menambahkan elemen baru di akhir queue, dequeue() menghapus elemen dari depan queue,

peek() mengambil nilai elemen pertama tanpa menghapusnya, dan display() mencetak semua elemen queue. Dalam fungsi main(), queue diuji dengan menambahkan elemen 10, 20, dan 30, mencetak isi queue, menampilkan elemen pertama, kemudian menghapus elemen pertama dan mencetak isi queue yang diperbarui. Struktur ini cocok untuk data dinamis karena ukurannya dapat berkembang sesuai kebutuhan.

c. Output :

```
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 20 30
```

3. Guide3

a. Syntax

```
#include<iostream>

using namespace std;

const int maksimalQueue = 5; // Maksimal antrian
int front = 0; // Penanda antrian
int back = 0; // Penanda
string queueTeller[5]; // Fungsi pengecekan

bool isFull() { // Pengecekan antrian penuh atau tidak
if (back == maksimalQueue) { return true; // =1
} else {
return false;
}
}

bool isEmpty() { // Antriannya kosong atau tidak
if (back == 0) { return true;
} else {
return false;
```

```
}
}

void enqueueAntrian(string data) { // Fungsi menambahkan antrian
if (isFull()) {
cout << "Antrian penuh" << endl;
} else {
if (isEmpty()) { // Kondisi ketika queue kosong
queueTeller[0] = data; front++;
back++;
} else { // Antrianya ada isi queueTeller[back] = data; back++;
}
}
}

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

int countQueue() { // Fungsi menghitung banyak antrian
return back;
}

void clearQueue() { // Fungsi menghapus semua antrian
if (isEmpty()) {
cout << "Antrian kosong" << endl;
```

```
} else {
for (int i = 0; i < back; i++) { queueTeller[i] = "";
}
back = 0;
front = 0;
}
}

void viewQueue() { // Fungsi melihat antrian
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("Andi");

enqueueAntrian("Maya");

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

dequeueAntrian();
```

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

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

    return 0;
    }
```

b. Penjelasan Syntax guide3 :

Program di atas adalah implementasi queue sederhana menggunakan array dengan kapasitas maksimum lima elemen. Fungsi enqueueAntrian() digunakan untuk menambahkan data ke akhir antrian, sementara dequeueAntrian() menghapus data dari awal antrian. Fungsi isFull() mengecek apakah antrian penuh, dan isEmpty() mengecek apakah antrian kosong. countQueue() menghitung jumlah elemen dalam antrian, dan clearQueue() menghapus semua elemen dari antrian. Fungsi viewQueue() menampilkan isi antrian. Dalam fungsi main(), program menambahkan data ("Andi" dan "Maya"), menampilkan antrian, menghapus elemen pertama, dan membersihkan seluruh antrian, dengan hasil setiap langkah ditampilkan ke layar.

c. Output :

```
Data antrian teller:
1. Gonzales
2. (kosong)
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 1
Data antrian teller:
1. (kosong)
2. (kosong)
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 0
Antrian kosong
Data antrian teller:
1. (kosong)
2. (kosong)
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 0
```

IV. UNGUIDED

1. Unguided1

   a. Syntax :

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

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

class Queue {
private:
```

```
    Node* front;
    Node* rear;

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

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

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

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

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

```
        Node* temp = front;
        cout << front->data << " telah dikeluarkan dari antrian." << endl;
        front = front->next;
        delete temp;

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

    void viewQueue() {
        if (isEmpty()) {
            cout << "Antrian kosong." << endl;
            return;
        }

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

int main() {
    Queue queue;
    queue.enqueue("Izzaty");
    queue.enqueue("Zahara");
    queue.viewQueue();
    queue.dequeue();
    queue.viewQueue();
```

```
    return 0;
}
```

b. Penjelasan Syntax :

Program di atas adalah implementasi queue menggunakan linked list. Kelas Queue memiliki dua pointer: front menunjuk elemen pertama, dan rear menunjuk elemen terakhir. Fungsi enqueue() menambahkan data ke akhir antrian, sedangkan dequeue() menghapus data dari awal antrian. Fungsi isEmpty() mengecek apakah antrian kosong, dan viewQueue() menampilkan semua elemen dalam antrian. Dalam fungsi main(), program menambahkan dua nama ("Izzaty" dan "Zahara") ke antrian, menampilkan isi antrian, menghapus elemen pertama ("Izzaty"), dan menampilkan antrian yang diperbarui. Program ini cocok untuk mengelola data yang terus berubah tanpa batasan ukuran tetap.

c. Output

```
Izzaty telah ditambahkan ke antrian.
Zahara telah ditambahkan ke antrian.
Data antrian:
Izzaty
Zahara
Izzaty telah dikeluarkan dari antrian.
Data antrian:
Zahara
```

2. Unguided2

a. Syntax

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

struct Node {
    string nama;
```

```
    string nim;
    Node* next;
};

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

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

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

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

        if (isEmpty()) {
            front = newNode;
            rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
        cout << nama << " dengan NIM " << nim << " telah ditambahkan ke
```

```
antrian." << endl;
    }

    void dequeue() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
            return;
        }

        Node* temp = front;
        cout << front->nama << " dengan NIM " << front->nim << " telah dikeluarkan dari antrian." << endl;
        front = front->next;
        delete temp;

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

    void viewQueue() {
        if (isEmpty()) {
            cout << "Antrian kosong." << endl;
            return;
        }

        Node* temp = front;
        cout << "Data antrian:" << endl;
        while (temp != nullptr) {
            cout << "Nama: " << temp->nama << ", NIM: " << temp->nim << endl;
            temp = temp->next;
        }
```

```
    }
};

int main() {
    Queue queue;
    queue.enqueue("izzaty", "2311104052");
    queue.enqueue("zahara", "2311104049");
    queue.viewQueue();
    queue.dequeue();
    queue.viewQueue();
    return 0;
}
```

b. Penjelasan Syntax

Program di atas mengimplementasikan queue menggunakan linked list, di mana setiap node menyimpan data berupa nama dan NIM. Fungsi enqueue() digunakan untuk menambahkan data baru ke akhir antrian, sedangkan dequeue() menghapus data dari awal antrian. Fungsi isEmpty() memeriksa apakah antrian kosong, dan viewQueue() menampilkan semua data dalam antrian. Pada fungsi main(), program menambahkan dua data ("izzaty" dengan NIM "2311104052" dan "zahara" dengan NIM "2311104049"), menampilkan isi antrian, menghapus elemen pertama, lalu menampilkan antrian yang tersisa. Program ini cocok untuk mengelola data dengan format kompleks secara dinamis.

c. Output

```
izzaty dengan NIM 2311104052 telah ditambahkan ke antrian.
zahara dengan NIM 2311104049 telah ditambahkan ke antrian.
Data antrian:
Nama: izzaty, NIM: 2311104052
Nama: zahara, NIM: 2311104049
izzaty dengan NIM 2311104052 telah dikeluarkan dari antrian.
Data antrian:
Nama: zahara, NIM: 2311104049
```

3. Unguided3

a. Syntax

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

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

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

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

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

    void enqueue(string nama, string nim) {
        Node* newNode = new Node();
        newNode->nama = nama;
        newNode->nim = nim;
        newNode->next = nullptr;
```

```
        if (isEmpty() || front->nim > nim) {
            newNode->next = front;
            front = newNode;
            if (rear == nullptr) {
                rear = newNode;
            }
        } else {
            Node* current = front;
            while (current->next != nullptr && current->next->nim < nim) {
                current = current->next;
            }
            newNode->next = current->next;
            current->next = newNode;
            if (newNode->next == nullptr) {
                rear = newNode;
            }
        }
          cout << nama << " dengan NIM " << nim << " telah ditambahkan ke antrian." << endl;
    }

    void dequeue() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
            return;
        }

        Node* temp = front;
        cout << front->nama << " dengan NIM " << front->nim << " telah dikeluarkan dari antrian." << endl;
        front = front->next;
```

```
        delete temp;

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

    void viewQueue() {
        if (isEmpty()) {
            cout << "Antrian kosong." << endl;
            return;
        }

        Node* temp = front;
        cout << "Data antrian:" << endl;
        while (temp != nullptr) {
            cout << "Nama: " << temp->nama << ", NIM: " << temp->nim << endl;
            temp = temp->next;
        }
    }
};

int main() {
    Queue queue;
    queue.enqueue("izzaty", "2311104052");
    queue.enqueue("zahara", "2311104049");
    queue.enqueue("br", "2311104048");
    queue.enqueue("barus", "2311104047");
    queue.viewQueue();
    queue.dequeue();
    queue.viewQueue();
    return 0;
```

```
}
```

b. Penjelasan Syntax

Program ini mengimplementasikan queue menggunakan linked list dengan antrian yang terurut berdasarkan NIM secara ascending (kecil ke besar). Fungsi enqueue() menambahkan data ke antrian dengan memastikan elemen baru disisipkan di posisi yang tepat berdasarkan urutan NIM. Jika antrian kosong atau NIM baru lebih kecil dari elemen pertama, elemen baru menjadi front. Fungsi dequeue() menghapus elemen pertama dari antrian, sementara viewQueue() menampilkan semua data dalam antrian. Pada fungsi main(), program menambahkan beberapa data ke antrian ("izzaty", "zahara", "br", dan "barus"), lalu menampilkan isi antrian sebelum dan setelah elemen pertama dihapus. Program ini berguna untuk pengelolaan data yang memerlukan pengurutan otomatis.

c. Output

```
izzaty dengan NIM 2311104052 telah ditambahkan ke antrian.
zahara dengan NIM 2311104049 telah ditambahkan ke antrian.
br dengan NIM 2311104048 telah ditambahkan ke antrian.
barus dengan NIM 2311104047 telah ditambahkan ke antrian.
Data antrian:
Nama: barus, NIM: 2311104047
Nama: br, NIM: 2311104048
Nama: zahara, NIM: 2311104049
Nama: izzaty, NIM: 2311104052
barus dengan NIM 2311104047 telah dikeluarkan dari antrian.
Data antrian:
Nama: br, NIM: 2311104048
Nama: zahara, NIM: 2311104049
Nama: izzaty, NIM: 2311104052
```

V. KESIMPULAN

Bahwa queue sebagai salah satu struktur data memiliki prinsip kerja *First In, First Out* (FIFO), di mana elemen pertama yang masuk akan menjadi elemen pertama yang keluar. Queue dapat diimplementasikan dalam berbagai cara, seperti

menggunakan array atau *linked list*. Setiap implementasi memiliki kelebihan, seperti kemudahan dalam pengelolaan memori pada *linked list* dan kecepatan akses langsung pada array. Operasi dasar pada queue meliputi enqueue (menambahkan elemen), dequeue (menghapus elemen), dan peek (melihat elemen di depan tanpa menghapusnya). Implementasi queue dapat disesuaikan dengan kebutuhan, seperti queue dengan data sederhana, queue dengan elemen kompleks seperti nama dan NIM, maupun queue terurut berdasarkan kriteria tertentu. Struktur data ini sangat berguna untuk berbagai kasus nyata, seperti pengelolaan antrian pelanggan, simulasi proses antrean, dan pengaturan proses dalam sistem komputer. Dengan memahami teori dan penerapannya, mahasiswa diharapkan mampu mengaplikasikan konsep queue dalam menyelesaikan masalah yang relevan di dunia nyata.