LAPORAN PRAKTIKUM
Modul 6
QUEUE

Disusun Oleh:
Berlian Seva Astryana - 2311104067
Kelas:
SE-07-02

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

1. **Tujuan**
   1.1 Mahasiswa mampu menjelaskan definisi dan konsep dari queue
   1.2 Mahasiswa mampu menerapkan operasi tambah, menghapus pada queue
   1.3 Mahasiswa mampu menerapkan operasi tampil data pada queue
2. **Landasan Teori**

Queue adalah struktur data linear yang menggunakan prinsip First In, First Out (FIFO), di mana elemen pertama yang dimasukkan akan menjadi elemen pertama yang dikeluarkan. Konsep ini mirip dengan antrian dalam kehidupan sehari-hari, seperti antrean pelanggan di kasir, di mana orang yang datang lebih dulu akan dilayani lebih dahulu. Queue memiliki dua operasi utama, yaitu enqueue untuk menambahkan elemen di bagian belakang (rear/tail) dan dequeue untuk menghapus elemen dari bagian depan (front/head). Selain itu, terdapat operasi tambahan seperti peek untuk melihat elemen di bagian depan tanpa menghapusnya, isEmpty untuk memeriksa apakah queue kosong, isFull untuk mengecek apakah queue penuh, dan size untuk menghitung jumlah elemen di dalam queue.

Perbedaan utama antara queue dan stack terletak pada aturan pengelolaan elemen. Queue menggunakan prinsip FIFO, sedangkan stack menggunakan prinsip Last In, First Out (LIFO), di mana elemen terakhir yang dimasukkan akan menjadi elemen pertama yang dihapus. Dalam implementasinya, queue dapat dibangun menggunakan array atau linked list. Array lebih sederhana tetapi memiliki keterbatasan ukuran tetap, sedangkan linked list lebih fleksibel karena ukurannya bergantung pada memori yang tersedia.

Queue memiliki banyak aplikasi dalam kehidupan nyata, seperti pengelolaan antrean pelanggan, penjadwalan proses pada CPU, dan buffering data dalam sistem jaringan. Dengan memahami prinsip dasar FIFO dan operasinya, queue menjadi alat yang sangat berguna dalam menyelesaikan berbagai masalah yang memerlukan pengolahan data secara terorganisasi dan efisien.

3. Guided

```
#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;
}
```

Kode tersebut merupakan implementasi Queue menggunakan array di dalam bahasa C++. Queue ini mengikuti prinsip First In, First Out (FIFO), di mana elemen pertama yang masuk akan menjadi elemen pertama yang keluar. Kode diawali dengan pendefinisian kelas Queue yang memiliki atribut front dan rear untuk menunjuk elemen pertama dan terakhir dalam queue, serta array arr[MAX] dengan ukuran maksimum yang telah ditentukan (MAX = 100).

Konstruktor Queue menginisialisasi nilai front dan rear dengan -1, menandakan queue kosong. Fungsi isFull() digunakan untuk memeriksa apakah queue penuh dengan membandingkan rear dengan indeks maksimum array. Sebaliknya, fungsi isEmpty() memeriksa apakah queue kosong dengan memeriksa apakah front bernilai -1 atau jika posisi front lebih besar dari rear.

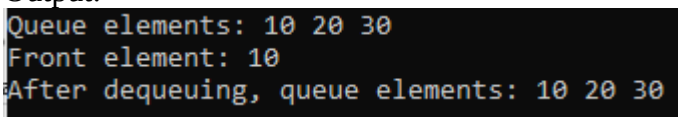
Operasi enqueue memungkinkan penambahan elemen ke bagian belakang queue. Jika queue penuh, akan ditampilkan pesan "Queue Overflow". Jika queue kosong, front

diatur menjadi 0, lalu elemen baru dimasukkan ke dalam array pada posisi rear + 1. Operasi dequeue digunakan untuk menghapus elemen dari bagian depan queue. Jika queue kosong, ditampilkan pesan "Queue Underflow". Untuk menghapus elemen, cukup dengan menggeser pointer front ke indeks berikutnya.

Fungsi peek mengembalikan elemen paling depan dari queue tanpa menghapusnya. Jika queue kosong, fungsi akan menampilkan pesan "Queue is empty". Fungsi display menampilkan seluruh elemen queue dari posisi front hingga rear, atau pesan "Queue is empty" jika tidak ada elemen.

Pada fungsi main, objek Queue dibuat, dan elemen 10, 20, dan 30 ditambahkan menggunakan enqueue. Queue kemudian ditampilkan menggunakan fungsi display, menunjukkan elemen yang telah dimasukkan. Elemen paling depan ditampilkan menggunakan fungsi peek, lalu queue ditampilkan kembali setelah operasi dequeue, menunjukkan perubahan posisi pointer front. Implementasi ini menggambarkan operasi dasar queue dengan array secara sederhana dan efisien.

Output:




```
#include <iostream>

using namespace std;

// Node untuk setiap elemen Queue
class Node {
public:
    int data;      // Data elemen
    Node* next;    // Pointer ke node berikutnya

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

// Kelas Queue menggunakan linked list
class Queue {
private:
    Node* front; // Pointer ke elemen depan Queue
    Node* rear;  // Pointer ke elemen belakang Queue

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

    // Mengecek apakah Queue kosong
    bool isEmpty() {
        return front == nullptr;
```

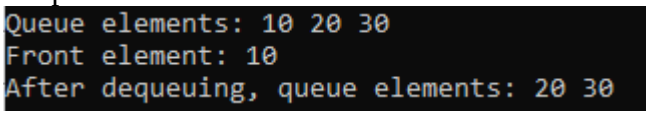
```
    }

    // Menambahkan elemen ke Queue
    void enqueue(int x) {
        Node* newNode = new Node(x);
        if (isEmpty()) {
            front = rear = newNode; // Jika Queue kosong
            return;
        }
        rear->next = newNode; // Tambahkan node baru ke belakang
        rear = newNode;       // Perbarui rear
    }

    // Menghapus elemen dari depan Queue
    void dequeue() {
        if (isEmpty()) {
            cout << "Queue Underflow\n";
            return;
        }
        Node* temp = front;      // Simpan node depan untuk dihapus
        front = front->next;     // Pindahkan front ke node berikutnya
        delete temp;             // Hapus node lama
        if (front == nullptr)    // Jika Queue kosong, rear juga harus null
            rear = nullptr;
    }

    // Mengembalikan elemen depan Queue tanpa menghapusnya
    int peek() {
        if (!isEmpty()) {
            return front->data;
        }
        cout << "Queue is empty\n";
        return -1; // Nilai sentinel
    }

    // Menampilkan semua elemen di Queue
    void display() {
        if (isEmpty()) {
            cout << "Queue is empty\n";
            return;
        }
        Node* current = front; // Mulai dari depan
        while (current) {      // Iterasi sampai akhir
            cout << current->data << " ";
            current = current->next;
        }
        cout << "\n";
    }
};

// Fungsi utama untuk menguji Queue
int main() {
```

```
    Queue q;

    // Menambahkan elemen ke Queue
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);

    // Menampilkan elemen di Queue
   cout << "Queue elements: ";
    q.display();

    // Menampilkan elemen depan
   cout << "Front element: " << q.peek() << "\n";

    // Menghapus elemen dari depan Queue
    q.dequeue();
   cout << "After dequeuing, queue elements: ";
    q.display();

    return 0;
}
```

Kode tersebut adalah implementasi Queue menggunakan linked list dalam bahasa C++. Queue ini mengikuti prinsip First In, First Out (FIFO), di mana elemen pertama yang masuk akan menjadi elemen pertama yang keluar. Setiap elemen queue diwakili oleh objek dari kelas Node, yang memiliki atribut data untuk menyimpan nilai elemen dan next sebagai pointer ke node berikutnya. Kelas Queue memiliki dua pointer utama, yaitu front untuk menunjuk elemen paling depan dan rear untuk menunjuk elemen paling belakang. Operasi utama dalam kelas ini mencakup enqueue, untuk menambahkan elemen di belakang queue; dequeue, untuk menghapus elemen dari depan queue; peek, untuk mengakses elemen depan tanpa menghapusnya; serta display, untuk menampilkan seluruh elemen queue dari depan hingga belakang. Fungsi tambahan isEmpty digunakan untuk memeriksa apakah queue kosong.

Pada fungsi utama (main()), objek queue dibuat, lalu beberapa elemen (10, 20, dan 30) dimasukkan menggunakan fungsi enqueue. Elemen-elemen ini kemudian ditampilkan menggunakan fungsi display, dan elemen paling depan diakses menggunakan fungsi peek. Setelah itu, elemen di depan queue dihapus dengan fungsi dequeue, dan queue ditampilkan kembali untuk memperlihatkan perubahan. Implementasi queue menggunakan linked list ini lebih fleksibel dibandingkan array, karena tidak dibatasi oleh ukuran tetap dan elemen dapat ditambahkan atau dihapus secara dinamis selama memori masih tersedia. Kode ini merepresentasikan pengelolaan queue dengan cara yang efisien dan dinamis.

Output:




```
#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;
}
```

Kode di atas adalah implementasi queue statis menggunakan array untuk mengelola antrian pada teller dengan kapasitas maksimum lima elemen. Queue ini menggunakan prinsip First In, First Out (FIFO), di mana elemen pertama yang masuk akan menjadi elemen pertama yang keluar. Variabel global front dan back digunakan untuk menandai posisi awal dan akhir elemen dalam antrian. Array queueTeller digunakan untuk menyimpan data antrian.
Fungsi utama yang ada dalam kode ini adalah:

a. isFull(): Memeriksa apakah antrian penuh dengan membandingkan nilai back dengan kapasitas maksimum (maksimalQueue).
b. isEmpty(): Memeriksa apakah antrian kosong dengan mengecek apakah back bernilai nol.
c. enqueueAntrian(string data): Menambahkan elemen baru ke antrian. Jika antrian kosong, elemen dimasukkan pada posisi awal. Jika tidak, elemen dimasukkan

pada posisi berikutnya. Jika penuh, ditampilkan pesan "Antrian penuh".

d. dequeueAntrian(): Menghapus elemen dari antrian dengan cara menggeser elemen-elemen setelahnya ke posisi sebelumnya, dan mengurangi nilai back. Jika kosong, ditampilkan pesan "Antrian kosong".
e. countQueue(): Menghitung jumlah elemen dalam antrian dengan mengembalikan nilai back.
f. clearQueue(): Menghapus semua elemen dalam antrian dengan mengosongkan array dan mereset nilai front dan back ke nol.
g. viewQueue(): Menampilkan isi antrian, menunjukkan elemen pada posisi tertentu atau menandainya sebagai "(kosong)" jika tidak ada elemen.

Pada fungsi main(), beberapa operasi dilakukan untuk menguji fungsi-fungsi di atas. Elemen "Andi" dan "Maya" ditambahkan ke antrian menggunakan enqueueAntrian. Isi antrian ditampilkan menggunakan viewQueue, dan jumlah elemen dihitung menggunakan countQueue. Setelah itu, elemen pertama dihapus dengan dequeueAntrian, lalu antrian ditampilkan kembali. Terakhir, semua elemen dihapus menggunakan clearQueue, dan hasilnya diperiksa. Implementasi ini sederhana dan efisien untuk jumlah elemen yang kecil tetapi tidak fleksibel karena ukuran antrian ditentukan secara tetap (statis).

Output:

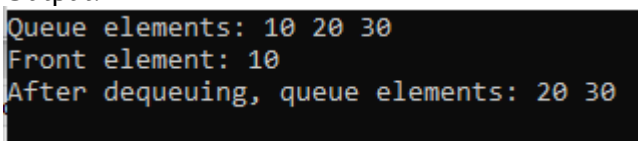
```
Data antrian teller:
1. Andi
2. (kosong)
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 1
Data antrian teller:
1. (kosong)
2. (kosong)
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 0
Antrian kosong
Data antrian teller:
1. (kosong)
2. (kosong)
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah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

4. **Unguided**

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

// Node untuk menyimpan data queue
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;
        } else {
            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;
}
```

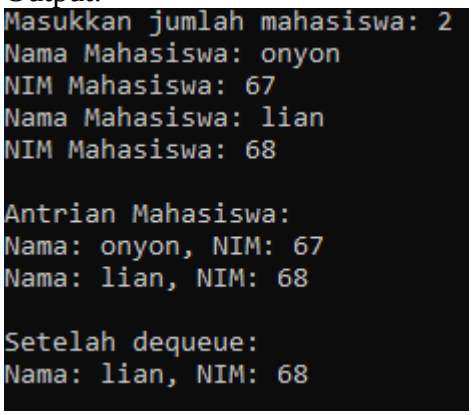
Output:




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

// Node untuk menyimpan data queue
class Node {
public:
    string nama;
    string nim;
    Node* next;

    Node(string nama, string nim) {
        this->nama = nama;
        this->nim = nim;
        this->next = nullptr;
    }
};

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

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

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

    void enqueue(string nama, string nim) {
        Node* newNode = new Node(nama, nim);
        if (isEmpty()) {
            front = rear = newNode;
        } else {
            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;
        }
    }

    void display() {
        if (isEmpty()) {
            cout << "Queue is empty\n";
            return;
        }
        Node* current = front;
        while (current) {
            cout << "Nama: " << current->nama << ", NIM: " << current->nim << "\n";
            current = current->next;
        }
    }
};

int main() {
    Queue q;
    int n;

    cout << "Masukkan jumlah mahasiswa: ";
    cin >> n;

    for (int i = 0; i < n; i++) {
        string nama, nim;
        cout << "Nama Mahasiswa: ";
        cin >> nama;
        cout << "NIM Mahasiswa: ";
        cin >> nim;
        q.enqueue(nama, nim);
    }

    cout << "\nAntrian Mahasiswa:\n";
    q.display();

    q.dequeue();
    cout << "\nSetelah dequeue:\n";
    q.display();

    return 0;
}
```

Output:




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

// Node untuk menyimpan data queue
class Node {
public:
    string nama;
    string nim;
    Node* next;

    Node(string nama, string nim) {
        this->nama = nama;
        this->nim = nim;
        this->next = nullptr;
    }
};

class PriorityQueue {
private:
    Node* front;

public:
    PriorityQueue() {
        front = nullptr;
    }

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

    void enqueue(string nama, string nim) {
        Node* newNode = new Node(nama, nim);
        if (isEmpty() || nim < front->nim) {
            newNode->next = front;
            front = newNode;
```

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

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

    void display() {
        if (isEmpty()) {
            cout << "Queue is empty\n";
            return;
        }
        Node* current = front;
        cout << "Daftar Mahasiswa Prioritas:\n";
        while (current) {
            cout << "Nama: " << current->nama << ", NIM: " << current->nim << "\n";
            current = current->next;
        }
    }
};

int main() {
    PriorityQueue pq;
    int n;

    cout << "Masukkan jumlah mahasiswa: ";
    cin >> n;

    for (int i = 0; i < n; i++) {
        string nama, nim;
        cout << "Nama Mahasiswa: ";
        cin >> nama;
        cout << "NIM Mahasiswa: ";
        cin >> nim;
        pq.enqueue(nama, nim);
    }

    cout << "\nAntrian berdasarkan prioritas NIM:\n";
    pq.display();
```

```
    return 0;
}
```

Output:

```
Masukkan jumlah mahasiswa: 2
Nama Mahasiswa: onyon
NIM Mahasiswa: 89
Nama Mahasiswa: lian
NIM Mahasiswa: 86

Antrian berdasarkan prioritas NIM:
Daftar Mahasiswa Prioritas:
Nama: lian, NIM: 86
Nama: onyon, NIM: 89
```

5. Kesimpulan

Queue adalah struktur data yang dapat diimplementasikan menggunakan array atau linked list, dengan keunggulan linked list dalam fleksibilitas ukuran dan efisiensi penanganan data dinamis. Pada implementasi lebih kompleks, queue dapat digunakan untuk menyimpan data seperti nama dan NIM mahasiswa, serta dimodifikasi menjadi priority queue agar elemen dengan NIM lebih kecil diprioritaskan. Fungsi dasar seperti enqueue, dequeue, peek, dan display memastikan operasi queue berjalan sesuai konsep FIFO. Dengan penerapan ini, queue menjadi solusi yang efektif untuk berbagai kebutuhan, seperti pengelolaan antrian layanan berbasis prioritas.