LAPORAN PRAKTIKUM
Modul 8
“Queue”

Disusun Oleh:
Dimastian Aji Wibowo (2311104058)
SE-07-02

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

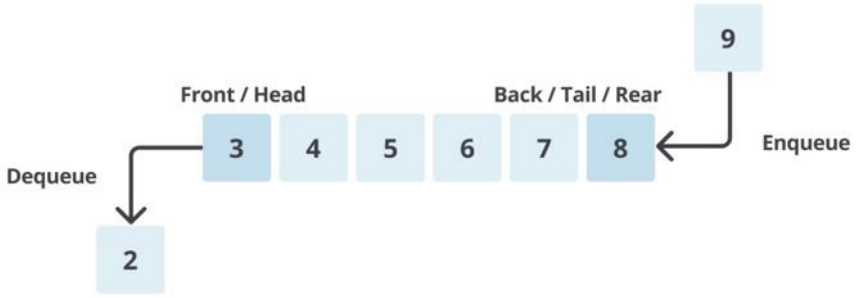
2024

1. **Tujuan**
    - Mampu menjelaskan definisi dan konsep dari Queue.
    - Mampu menerapkan operasi tambah dan hapus pada Queue.
    - Mampu menerapkan operasi menampilkan data pada Queue.

2. **Landasan Teori**

Queue adalah struktur data yang digunakan untuk menyimpan data dengan metode FIFO (First-In First-Out). Data yang pertama dimasukkan ke dalam queue akan menjadi data yang pertama pula untuk dikeluarkan dari queue. Queue mirip dengan konsep antrian pada kehidupan sehari-hari, dimana konsumen yang datang lebih dulu akan dilayani terlebih dahulu.

Implementasi queue dapat dilakukan dengan menggunakan array atau linked list. Struktur data queue terdiri dari dua pointer yaitu front dan rear. Front/head adalah pointer ke elemen pertama dalam queue dan rear/tail/back adalah pointer ke elemen terakhir dalam queue.



Perbedaan antara tumpukan dan antrean memperkenalkan beberapa aturan untuk menambah dan menghapus elemen. Dalam sebuah tumpukan, operasi penambahan dan penghapusan dilakukan pada satu ujung yang disebut puncak. Elemen terakhir yang dimasukkan ke dalam tumpukan akan berada di bagian atas dan ini akan menjadi elemen pertama yang dihapus. Sifat ini dikenal sebagai LIFO (Last In, First Out). Analogi sederhana untuk tumpukan adalah tumpukan piring, di mana piring terakhir yang ditambahkan berada di bagian atas dan akan dihapus terlebih dahulu.

Sebaliknya, antrean sangat berbeda: ada dua ujung di mana operasi penambahan dan penghapusan terjadi. Elemen-elemen baru didorong masuk

dari ujung belakang (ujung ekor), dan ditarik keluar dari ujung depan (ujung kepala). Antrian didasarkan pada prinsip FIFO (First In First Out), yang berarti elemen pertama yang masuk ke dalam antrean adalah yang pertama keluar. Menambahkan elemen ke dalam antrean disebut Enqueue, sementara menghapusnya disebut Dequeue. Selama Enqueue, elemen yang ditambahkan akan masuk ke ujung belakang yang baru setelah yang sebelumnya; tetapi untuk Dequeue, ujung depan akan dibersihkan dari elemen yang pertama kali masuk, dan posisi depan sekarang akan menunjuk ke elemen berikutnya yang tersedia. Aplikasi khas dari konsep antrian dalam kehidupan nyata adalah antrian yang terbentuk di kasir, di mana pelanggan pertama adalah yang pertama dilayani.

Berikut merupakakn operasi – operasi pada queue:

1. enqueue(): menambahkan data ke dalam queue.
2. dequeue(): mengeluarkan data dari queue.
3. peek() : mengambil data dari queue tanpa menghapusnya.
4. isEmpty(): memeriksa apakah queue kosong.
5. isFull() : memeriksa apakah queue penuh.
6. size() : menghitung jumlah elemen dalam queue.

3. Guided

A. Guided 1

1. Definisikan MAX dengan nilai 100 untuk menentukan ukuran maksimum queue, mendefinisikan class Stack dengan atribut int front dan rear untuk menyimpan indeks elemen depan dan belakang pada stack dan atribut arr[MAX] yaitu ukuran dari array untuk menyimpan elemen – elemen queue.
2. Membuat konstruktor kelas Queue dengan mengisikan nilai front dan rear menjadi -1 yang menunjukan queue kosong.
3. Membuat method isFull() dengan menggunakan boolean untuk memeriksa apakah queue sudah penuh dengan cara mengembalikan nilai true jika rear sudah mencapai MAX-1 maka queue penuh.
4. Membuat method isEmpty() dengan menggunakan boolean untuk memeriksa apakah queue kosong dengan cara mengembalikan true jika front bernilai -1 dan front lebih besar dari rear.
5. Fungsi enqueue() menambahkan elemen kedalam queue dengan memeriksa apakah queue penuh, jika penuh maka tampilkan pesan queue penuh dan jika queue kosong maka memperbarui nilai front menjadi 0

untuk menunjukan elemen pertama telah dimasukan dan memperbarui rear menjadi rear+1.

6. Fungsi dequeue() untuk mengeluarkan elemen dari queue dengan memeriksa apakah queue kosong, jika kosong maka tampilkan pesan queue kosong dan jika queue tidak kosong maka naikkan front untuk menunjukkan bahwa elemen pertama telah dikeluarkan.

```
#define MAX 100
#include <iostream>
using namespace std;

class Queue {
private:
    int front, rear;
    int arr[MAX];

public:
    Queue() {
        front = -1;
        rear = -1;
    }
    bool isEmpty() {
        return front == -1 || front > rear;
    }
    bool isFull() {
        return rear == MAX - 1;
    }
    void enqueue(int x) {
        if (isFull()) {
            cout << "Queue Overflow\n";
            return;
        }
        if (isEmpty()) {
            front = 0;
        }
        arr[++rear] = x;
    }
    void dequeue() {
        if (isEmpty()) {
            cout << "Queue Underflow\n";
            return;
        }
        front++;
    }
```

7. Fungsi peek() mengembalikan elemen di depan queue (arr[front]) tanpa menghapusnya dan jika queue kosong maka menampilkan pesan dan return -1.
8. Fungsi display() menampilkan elemen pada queue, jika queue kosong maka tampilkan pesan queue kosong dan jika tidak maka menggunakan perulangan untuk menampilkan elemen dengan front lebih kecil sama dengan dari rear lalu bertambah.
9. Membuat main() dengan menginisialisasi class Queue menjadi q, menambahkan elemen kedalam stack dengan enqueue(), menghapus elemen dari stack menggunakan dequeue(), dan menampilkan isi stack menggunakan display().

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

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

    return 0;
}
```

B. Guided 2

1. Mendefinisikan struktur node dengan atribut int data untuk menyimpan nilai integer untuk setiap node, serta Node* next yang merupakan pointer yang menunjuk ke node berikutnya.
2. Membuat konstruktor untuk menginisialisasi data dengan nilai value dan mengatur next ke nullptr.
3. Mendefinisikan class Queue dengan atribut Node* front sebagai pointer ke elemen pertama dalam antrean (front) dan Node* rear sebagai pointer ke elemen terakhir dalam antrean (rear). Membuat konstruktor class Queue untuk menginisialisasi antrean kosong dengan front dan rear bernilai nullptr.
4. Method isEmpty() menggunakan boolean untuk memeriksa apakah antrean kosong dengan mengembalikan true jika front adalah nullptr.
5. Method enqueue(int x) untuk menambahkan elemen x ke dalam antrean dengan membuat node baru newNode dengan nilai x, jika antrean kosong maka front dan rear diatur ke newNode dan jika tidak kosong maka newNode ditambahkan ke akhir antrean dengan cara mengatur rear->next ke newNode dan memperbarui rear ke newNode.

```cpp
#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 = nullptr;
        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;
        }
    }
```

6. Method dequeue() untuk menghapus elemen pada front dengan memeriksa apakah antrean kosong, jika kosong maka menampilkan pesan queue kosong dan jika tidak kosong maka node front disimpan ke temp, front diperbarui menjadi front->next, dan temp dihapus. Jika setelah penghapusan front menjadi nullptr maka rear juga diatur ke nullptr.
7. Method peek() untuk mengembalikan nilai elemen pada front tanpa menghapusnya dengan memeriksa apakah antrean kosong jika kosong maka menampilkan pesan bahwa queue kosong dan mengembalikan -1 dan jika tidak maka mengembalikan front->data.
8. Method display() untuk menampilkan semua elemen antrean dari front ke rear dengan memeriksa apakah antrean kosong, jika kosong maka menampilkan pesan bahwa queue kosong dan jika tidak maka menggunakan pointer current yang dimulai dari front dan melakukan perulangan untuk menampilkan data hingga mencapai nullptr.

```
    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";
    }
};
```

9. Membuat fungsi main() dengan menginisialisasi class Queue menjadi q, menambahkan elemen ke dalam antrean dengan enqueue(), menghapus elemen dari antrean menggunakan dequeue(), dan menampilkan isi antrean menggunakan display().

```
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 dequeue, queue elements: ";
    q.display();

    return 0;
}
```

C. Guided 3

1. Mendefinisikan array queueTeller dengan tipe data string untuk menyimpan data antrean sebanyak maksimal 5 elemen serta variabel front untuk menunjuk elemen pertama dan back untuk menunjuk elemen terakhir.
2. Method isFull() menggunakan boolean untuk memeriksa apakah antrean penuh dengan mengembalikan true jika back sama dengan kapasitas maksimum.
3. Method isEmpty() menggunakan boolean untuk memeriksa apakah antrean kosong dengan mengembalikan true jika back bernilai 0.
4. Method enqueueAntrian(string data) untuk menambahkan elemen data ke dalam antrean dengan memeriksa apakah antrean penuh menggunakan isFull(), jika penuh maka menampilkan pesan "Antrian penuh", jika tidak penuh maka elemen ditambahkan pada indeks back

dan back dinaikkan 1, dan jika antrean sebelumnya kosong, front juga dinaikkan ke 1.

```cpp
#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++;
        }
    }
}
```

5. Method dequeueAntrian() untuk menghapus elemen dari antrean dengan memeriksa apakah antrean kosong menggunakan isEmpty(), jika kosong maka menampilkan pesan "Antrian kosong", jika tidak kosong maka elemen pada indeks pertama dihapus dengan cara menggeser semua elemen ke depan, dan nilai back dikurangi 1.
6. Method countQueue() untuk menghitung jumlah elemen dalam antrean dengan mengembalikan nilai back.
7. Method clearQueue() untuk menghapus semua elemen dalam antrean dengan memeriksa apakah antrean kosong menggunakan isEmpty(), jika kosong maka menampilkan pesan "Antrian kosong", dan jika tidak maka setiap elemen dihapus dengan mengganti nilainya menjadi string kosong, lalu front dan back diatur kembali ke 0.
8. Method viewQueue() untuk menampilkan semua elemen dalam antrean dengan memeriksa apakah antrean kosong menggunakan isEmpty(), jika kosong maka menampilkan pesan "Queue kosong" dan jika tidak maka elemen – elemen dalam antrean ditampilkan dari indeks pertama hingga indeks terakhir (maksimalQueue), dengan elemen kosong ditampilkan sebagai "(kosong)".

```
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;
        }
    }
}
```

9. Membuat main() dengan memanggil fungsi – fungsi enqueueAntrian() untuk memasukkan nama seperti "Andi" dan "Maya", menampilkan elemen dalam antrean menggunakan viewQueue(), menghapus elemen antrean dengan dequeueAntrian(), menampilkan elemen dalam antrean setelah penghapusan elemen menggunakan viewQueue(), dan membersihkan antrean dengan clearQueue(), menampilkan kondisi antrean yang kosong, dan menghitung jumlah elemen.

```
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;
}
```

4. Unguided

A. Unguided 1

1. Mendefinisikan struktur Node dengan atribut string data untuk menyimpan data elemen antrian, Node* next berupa pointer yang menunjuk ke node berikutnya, dan membuat konstruktor Node dengan parameter string value untuk menginisialisasi data dengan nilai value dan mengatur next ke nullptr.
2. Definisikan class Queue dengan atribut Node* front yang menunjuk ke elemen pertama (head) dalam antrian, Node* back yang menunjuk elemen terakhir (tail) dalam antrian, int size untuk menyimpan jumlah

elemen dalam antrian, dan const int maksimalQueue untuk menentukan kapasitas maksimum antrian.

3. Membuat konstruktor class Queue untuk menginisialisasi antrian kosong dengan front dan back bernilai nullptr dan size diatur ke 0.
4. Method isFull() untuk memeriksa apakah antrian penuh dengan membandingkan size dengan maksimalQueue dan mengembalikan true jika antrian penuh dan sebaliknya.
5. Method isEmpty() untuk memeriksa apakah antrian kosong dengan memeriksa apakah size bernilai 0, lalu mengembalikan true jika antrian kosong dan false jika tidak.
6. Method enqueueAntrian(string data) untuk menambahkan elemen baru ke antrian dengan memeriksa apakah antrian penuh atau tidak, jika penuh maka menampilkan pesan bahwa antrian penuh dan jika kosong maka elemen baru ditambahkan sebagai elemen pertama, jika tidak kosong maka elemen baru ditambahkan di akhir antrian dengan mengatur back->next ke node baru dan back diperbarui, dan size bertambah satu.
7. Method dequeueAntrian() untuk menghapus elemen dari antrian dengan memeriksa apakah antrian kosong dan jika kosong maka menampilkan pesan bahwa antrian kosong, jika tidak kosong maka elemen pertama (front) dihapus dan front diperbarui ke front->next, jika antrian menjadi kosong setelah penghapusan maka back juga diatur ke nullptr, lalu size berkurang satu.
8. Method countQueue() mengembalikan jumlah elemen dalam antrian dengan mengembalikan nilai size.
9. Method clearQueue() digunakan untuk menghapus semua elemen dalam antrian dengan menggunakan perulangan lalu memanggil method dequeueAntrian() hingga antrian kosong, lalu menampilkan pesan bahwa antrian telah kosong.
10. Method viewQueue() untuk menampilkan semua elemen dalam antrian, jika antrian kosong maka menampilkan pesan antrian kosong, jika antrian tidak kosong maka elemen ditampilkan satu per satu mulai dari front hingga back menggunakan pointer current.
11. Membuat main() dengan menginisialisasi objek Queue menjadi q, menambahkan elemen ke antrian menggunakan enqueueAntrian(), menampilkan isi antrian dengan viewQueue(), menghapus elemen dalam antrian menggunakan dequeueAntrian(), membersihkan semua

elemen dalam anntrian menggunakan clearQueue(), dan menampilkan jumlah elemen dalam antrian dengan countQueue().

12. Berikut merupakan output dari program tersebut.

B. Unguided 2

1. Mendefinisikan struktur Node dengan atribut string nama untuk menyimpan nama mahasiswa, string nim untuk menyimpan NIM, Node* next yang merupakan pointer yang menunjuk ke node berikutnya dalam antrian.
2. Membuat konstruktor class Node dengan parameter string mahasiswa dan string nimMahasiswa unruk menginisialisasi nama dan NIM mahasiswa serta mengatur next ke nullptr.
3. Mendefinisikan class Queue dengan atribut Node* front untuk menunjuk ke elemen pertama dalam antrian, Node* back untuk menunjuk ke elemen terakhir dalam antrian, int size untuk menyimpan jumlah elemen dalam antrian, dan const int maksimalQueue untuk menentukan kapasitas maksimum antrian.
4. Membuat konstruktor class Queue untuk menginisialisasi antrian kosong dengan front dan back bernilai nullptr dan size diatur ke 0.
5. Method isFull() untuk memeriksa apakah antrian penuh dengan membandingkan size dengan maksimalQueue dengan mengembalikan true jika antrian penuh dan false jika tidak.
6. Method isEmpty() untuk memeriksa apakah antrian kosong dengan memeriksa apakah size bernilai 0 dengan mengembalikan true jika antrian kosong dan false jika tidak.
7. Method enqueueAntrian(string namaMahasiswa, string nimMahasiswa) untuk menambahkan mahasiswa baru ke antrian, jika antrian penuh maka menampilkan pesan bahwa antrian penuh, jika antrian kosong maka elemen baru ditambahkan sebagai elemen pertama dengan front dan back menunjuk ke node baru, jika tidak kosong maka elemen baru ditambahkan di akhir antrian dengan mengatur back->next ke node baru dan back diperbarui, lalu ukuran size bertambah satu.

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

class Node{
public:
    string nama;
    string nim;
    Node* next;
    Node(string namaMahasiswa, string nimMahasiswa){
        nama = namaMahasiswa;
        nim = nimMahasiswa;
        next = nullptr;
    }
};
class Queue{
private:
    Node* front;
    Node* back;
    int size;
    const int maksimalQueue = 5;

public:
    Queue(){
        front = nullptr;
        back = nullptr;
        size = 0;
    }
    bool isFull(){
        return size == maksimalQueue;
    }
    bool isEmpty(){
        return size == 0;
    }
    void enqueueAntrian(string namaMahasiswa, string nimMahasiswa) {
        if(isFull()){
            cout << "Antrian penuh" << endl;
        }else{
            Node* newNode = new Node(namaMahasiswa, nimMahasiswa);
            if (isEmpty()){
                front = back = newNode;
            }else{
                back->next = newNode;
                back = newNode;
            }
            size++;
        }
    }
```

8. Method dequeueAntrian() untuk menghapus mahasiswa dari antrian, jika antrian kosong maka menampilkan pesan antrian kosong, jika tidak kosong maka elemen pertama dihapus dan front diperbarui ke front->next, jika antrian menjadi kosong setelah pnghapusan dan back juga diatur ke nullptr, lalu size berkurang satu.
9. Method countQueue() untuk mengembalikan jumlah mahasiswa dalam antrian dengan mengembalikan nilai dari size.
10. Method clearQueue() untuk menghapus semua mahasiswa dalam antrian dengan memanggol dequeueAntrian lalu menggunakan perulangan hingga antrian kosong dan menampilkan pesan antrian telah dikosongkan setelah semua data dihapus.
11. Method viewQueue() untuk menampilkan semua mahasiswa dalam antrian, jika antrian kosong maka menampilkan pesan bahwa antrian kosong, dan jika tidak kosong maka setiap elemen ditampilkan satu per satu mulai dari front hingga back menggunakan pointer current.

```
    void dequeueAntrian(){
        if(isEmpty()){
            cout << "Antrian kosong" << endl;
        }else{
            Node* temp = front;
            front = front->next;
            delete temp;
            size--;
            if(front == nullptr){
                back = nullptr;
            }
        }
    }
    int countQueue(){
        return size;
    }
    void clearQueue(){
        while(!isEmpty()){
            dequeueAntrian();
        }
        cout<<"Antrian telah dikosongkan"<<endl;
    }
    void viewQueue(){
        cout<<"Data antrian mahasiswa:"<<endl;
        if(isEmpty()){
            cout<<"(Kosong)"<<endl;
        }else{
            Node* current = front;
            int i = 1;
            while(current != nullptr){
                cout<<i++<<". Nama: "<<current->nama
                    <<", NIM: "<<current->nim<<endl;
                current = current->next;
            }
        }
    }
};
```

12. Membuat main() lalu menginisialisasi Queue menjadi q, dengan menampilkan pilihan dengan menggunakan switch dan setiap case akan memanggil method – method sesuai dengan tampilan pilihan.
13. Untuk pilihan satu yaitu enqueueAntrian() meminta user untuk menginput nama mahasiswa dan NIM mahasiswa dengan menggunakan perulangan jika sudah penuh maha tidak bisa menginputkan data lagi.

```cpp
int main(){
    Queue q;
    int pilihan;

    do{
        cout<<"\nMenu Antrian Mahasiswa:" <<endl;
        cout<<"1. Tambah mahasiswa ke antrian"<<endl;
        cout<<"2. Hapus mahasiswa dari antrian"<<endl;
        cout<<"3. Lihat semua antrian"<<endl;
        cout<<"4. Bersihkan antrian"<<endl;
        cout<<"5. Keluar"<<endl;
        cout<<"Pilihan: ";
        cin>>pilihan;

        switch(pilihan){
        case 1:{
            if(q.isFull()){
                cout<<"Antrian penuh, tidak dapat menambahkan data baru."<<endl;
            }else{
                string nama, nim;
                cout<<"Masukkan nama mahasiswa: ";
                cin.ignore();
                getline(cin, nama);
                cout<<"Masukkan NIM mahasiswa: ";
                cin>>nim;
                q.enqueueAntrian(nama, nim);
            }
            break;
        }
        case 2:
            q.dequeueAntrian();
            break;
        case 3:
            q.viewQueue();
            break;
        case 4:
            q.clearQueue();
            break;
        case 5:
            cout<<"Keluar dari program."<<endl;
            break;
        default:
            cout<<"Pilihan tidak valid."<<endl;
        }
    }while(pilihan != 5);

    return 0;
}
```

14. Berikut merupakan output dari program tersebut.

```
"D:\College\Semester 3\Prakt
Pilihan: 1
Antrian penuh, tidak dapat menambahkan data baru.

Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 3
Data antrian mahasiswa:
1. Nama: Dimastian, NIM: 1
2. Nama: Aji, NIM: 2
3. Nama: Wibowo, NIM: 3
4. Nama: Alucard, NIM: 4
5. Nama: Zilong, NIM: 5

Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 2

Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 3
Data antrian mahasiswa:
1. Nama: Aji, NIM: 2
2. Nama: Wibowo, NIM: 3
3. Nama: Alucard, NIM: 4
4. Nama: Zilong, NIM: 5

Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
```

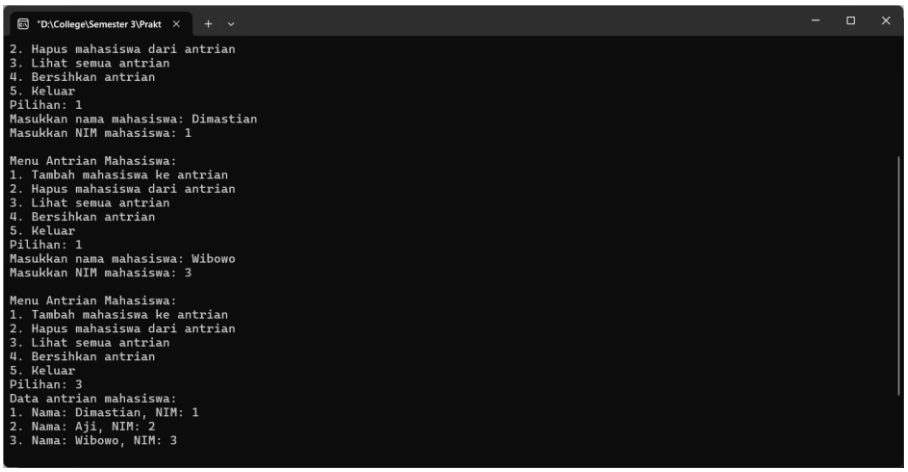
```
"D:\College\Semester 3\Prakt

Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 4
Antrian telah dikosongkan

Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 3
Data antrian mahasiswa:
(Kosong)

Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 5
Keluar dari program.

Process returned 0 (0x0)   execution time : 122.254 s
Press any key to continue.
```

C. Unguided 3

1. Perbedaan terdapat pada class Queue dengan tidak adanya atribut Node* back.
2. Pada method enqueueAntrian() elemen baru dimasukkan pada posisi tertentu berdasarkan nilai nim sehingga antrian tetap terurut, dengan cara jika antrian kosong atau elemen baru memiliki NIM lebih kecil dari elemen di front, elemen baru ditambahkan di depan atau front dan jika tidak maka elemen baru dimasukkan pada posisi yang tepat dalam antrian berdasarkan NIM.

```
33  void enqueueAntrian(string namaMahasiswa, string nimMahasiswa){
34      if(isFull()){
35          cout<<"Antrian penuh"<<endl;
36          return;
37      }
38
39      Node* newNode = new Node(namaMahasiswa, nimMahasiswa);
40
41      if(isEmpty() || nimMahasiswa < front->nim){
42          newNode->next = front;
43          front = newNode;
44      }else{
45          Node* current = front;
46          while(current->next != nullptr && current->next->nim < nimMahasiswa){
47              current = current->next;
48          }
49          newNode->next = current->next;
50          current->next = newNode;
51      }
52      size++;
53  }
```

3. Berikut merupakan output dari program tersebut,



5. Kesimpulan

Queue merupakan struktur data linear yang bekerja dengan prinsip **FIFO (First In, First Out)**, di mana elemen yang pertama masuk akan menjadi elemen pertama yang keluar. Implementasi *queue* dapat dilakukan menggunakan array atau *linked list*, di mana masing-masing memiliki kelebihan dan kekurangan. Praktikum ini juga memperkenalkan operasi dasar seperti **enqueue** (menambahkan elemen ke antrian), **dequeue** (menghapus elemen dari antrian), **isEmpty** (memeriksa apakah antrian kosong), dan **isFull** (memeriksa apakah antrian penuh). Dengan memahami dan mengimplementasikan operasi ini, mahasiswa dapat mengelola data dalam antrian dengan efisien, termasuk menambahkan fitur seperti prioritas elemen berdasarkan kriteria tertentu. Praktikum ini memperkuat pemahaman konsep antrian dan relevansinya dalam berbagai aplikasi dunia nyata, seperti manajemen antrean pelanggan atau penjadwalan proses dalam sistem operasi.