

LAPORAN PRAKTIKUM

MODUL 7

QUEUE



Disusun oleh:

Nabila Shasya Sabrina

NIM: 2311102039

Dosen Pengampu:

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

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

PURWOKERTO

2024

BAB I

TUJUAN PRAKTIKUM

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

BAB II

DASAR TEORI

a. Pengertian Queue

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.



FIRST IN FIRST OUT (FIFO)

Perbedaan antara stack dan queue terdapat pada aturan penambahan dan penghapusan elemen. Pada stack, operasi penambahan dan penghapusan elemen dilakukan di satu ujung. Elemen yang terakhir diinputkan akan berada paling dengan dengan ujung atau dianggap paling atas sehingga pada operasi penghapusan, elemen teratas tersebut akan dihapus paling awal, sifat demikian dikenal dengan LIFO.

Pada Queue, operasi tersebut dilakukan ditempat berbeda (melalui salah satu ujung) karena perubahan data selalu mengacu pada Head, maka hanya ada 1 jenis insert maupun delete. Prosedur ini sering disebut **Enqueue** dan **Dequeue** pada kasus Queue. Untuk Enqueue, cukup tambahkan elemen setelah elemen terakhir Queue, dan untuk Dequeue, cukup "geser"kan Head menjadi elemen selanjutnya.

Operasi pada Queue

- `enqueue()` : menambahkan data ke dalam queue.
- `dequeue()` : mengeluarkan data dari queue.
- `peek()` : mengambil data dari queue tanpa menghapusnya.
- `isEmpty()` : mengecek apakah queue kosong atau tidak.
- `isFull()` : mengecek apakah queue penuh atau tidak.
- `size()` : menghitung jumlah elemen dalam queue.

BAB III

GUIDED

1. Guided 1

Source code:

```
#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;
    }
}

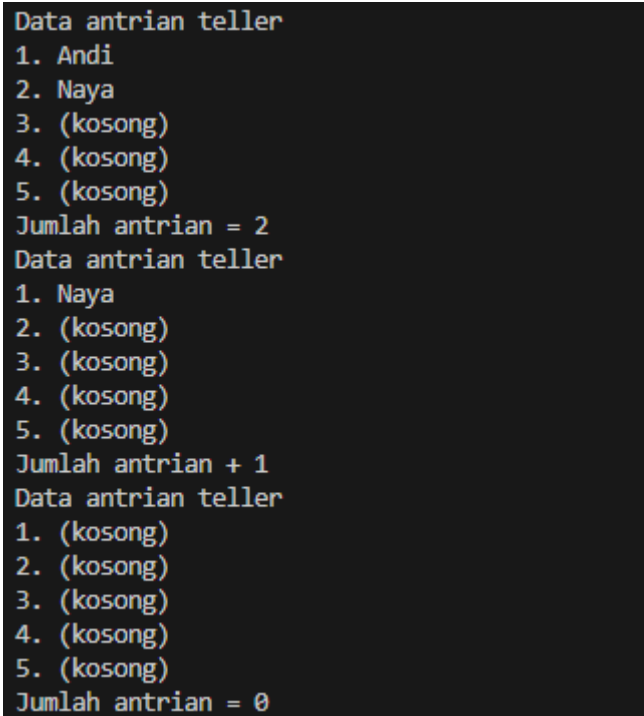
void viewQueue()
{
    cout << "Data antrian teller" << endl;
    for (int i = 0; i < maksimalQueue; i++)
    {
        if (queueTeller[i] != "")
        {
            cout << i + 1 << ". " << queueTeller[i] << endl;
        }
        else
        {
            cout << i + 1 << ". (kosong)" << endl;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

int main()
{
    enqueueAntrian("Andi");
    enqueueAntrian("Naya");
    viewQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;
    dequeueAntrian();
    viewQueue();
    cout << "Jumlah antrian + " << countQueue() << endl;
}

```

```
clearQueue();  
viewQueue();  
cout << "Jumlah antrian = " << countQueue() << endl;  
return 0;  
}
```

Screenshoot Program:



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

Deskripsi program:

Program berikut merupakan implementasi sederhana dari antrian (queue) menggunakan array. Antrian memiliki kapasitas maksimal lima elemen, yang disimpan dalam array **queueTeller**. Program ini menggunakan dua variabel, **front** dan **back**, untuk melacak posisi depan dan belakang antrian.

Fungsi **isFull()** digunakan untuk memeriksa apakah antrian penuh, sementara fungsi **isEmpty()** memeriksa apakah antrian kosong. Fungsi **enqueueAntrian()** menambahkan elemen baru ke dalam antrian jika antrian tidak penuh, dan menampilkan pesan "Antrian Penuh" jika sebaliknya. Fungsi **dequeueAntrian()** mengeluarkan elemen dari antrian jika antrian tidak kosong, dan menampilkan pesan "Antrian Kosong" jika sebaliknya.

Fungsi **countQueue()** mengembalikan jumlah elemen dalam antrian, dan fungsi **clearQueue()** mengosongkan antrian dengan mengatur semua elemen array menjadi string kosong dan mengatur ulang nilai **front** dan **back** ke 0. Fungsi **viewQueue()** menampilkan semua elemen dalam antrian beserta posisi masing-masing, menampilkan "(kosong)" untuk posisi yang tidak diisi.

Pada fungsi **main()**, beberapa operasi antrian dilakukan: menambahkan elemen "Andi" dan "Naya" ke dalam antrian, menampilkan antrian, menampilkan jumlah elemen dalam antrian, mengeluarkan elemen terdepan, menampilkan antrian lagi, mengosongkan antrian, dan terakhir menampilkan antrian yang sudah kosong. Output dari program ini menunjukkan perubahan kondisi antrian setelah setiap operasi.

BAB IV

UNGUIDED

1. Ubahlah penerapan konsep queue pada bagian guided dari array menjadi linked list!

Source code:

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

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

class Queue {
private:
    Node* front;
    Node* back;
public:
    Queue() {
        front = nullptr;
        back = nullptr;
    }

    void enqueueAntrian(string data) {
        Node* newNode = new Node();
        newNode->data = data;
        newNode->next = nullptr;
        if (isEmpty()) {
            front = newNode;
            back = newNode;
        } else {
            back->next = newNode;
            back = newNode;
        }
        cout << "Antrian ditambahkan: " << data << endl;
    }

    void dequeueAntrian() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
            return;
        }
        Node* temp = front;
        cout << "Antrian dihapus: " << front->data << endl;
        front = front->next;
        delete temp;
    }

    int countQueue() {
        int count = 0;
        Node* current = front;
        while (current != nullptr) {
            count++;
            current = current->next;
        }
        return count;
    }

    void clearQueue() {
```

```

        while (!isEmpty()) {
            dequeueAntrian();
        }
        cout << "Antrian dikosongkan" << endl;
    }

    void viewQueue() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
            return;
        }
        cout << "Data antrian teller:" << endl;
        Node* current = front;
        int position = 1;
        while (current != nullptr) {
            cout << position << ". " << current->data << endl;
            current = current->next;
            position++;
        }
    }

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

int main() {
    Queue antrian;
    antrian.enqueueAntrian("Andi");
    antrian.enqueueAntrian("Maya");
    antrian.viewQueue();
    cout << "Jumlah antrian = " << antrian.countQueue() << endl;
    antrian.dequeueAntrian();
    antrian.viewQueue();
    cout << "Jumlah antrian = " << antrian.countQueue() << endl;
    antrian.clearQueue();
    antrian.viewQueue();
    cout << "Jumlah antrian = " << antrian.countQueue() << endl;
    return 0;
}

```

Screenshot Output:

```

Antrian ditambahkan: Andi
Antrian ditambahkan: Maya
Data antrian teller:
1. Andi
2. Maya
Jumlah antrian = 2
Antrian dihapus: Andi
Data antrian teller:
1. Maya
Jumlah antrian = 1
Antrian dihapus: Maya
Antrian dikosongkan
Antrian kosong
Jumlah antrian = 0

```

Deskripsi Program:

Program ini merupakan implementasi antrian (queue) menggunakan struktur data linked list. Antrian diimplementasikan dalam kelas **Queue** yang memiliki dua pointer, **front** dan **back**, untuk melacak elemen pertama dan terakhir dalam antrian. Setiap elemen dalam antrian diwakili oleh node yang terdiri dari data string dan pointer ke node berikutnya.

Konstruktor **Queue** menginisialisasi antrian dengan **front** dan **back** bernilai **nullptr**, yang menandakan bahwa antrian kosong. Fungsi **enqueueAntrian()** menambahkan elemen baru ke antrian. Jika antrian kosong, elemen baru menjadi elemen pertama dan terakhir. Jika tidak, elemen baru ditambahkan ke belakang antrian dan pointer **back** diperbarui.

Fungsi **dequeueAntrian()** menghapus elemen dari depan antrian. Jika antrian kosong, pesan "Antrian kosong" ditampilkan. Jika tidak, elemen depan dihapus dan pointer **front** diperbarui ke elemen berikutnya.

Fungsi **countQueue()** menghitung jumlah elemen dalam antrian dengan melakukan iterasi dari depan ke belakang. Fungsi **clearQueue()** menghapus semua elemen dalam antrian dengan menggunakan **dequeueAntrian()** berulang kali sampai antrian kosong.

Fungsi **viewQueue()** menampilkan semua elemen dalam antrian beserta posisinya. Jika antrian kosong, pesan "Antrian kosong" ditampilkan. Fungsi **isEmpty()** mengembalikan nilai true jika antrian kosong, dan false jika tidak.

Pada fungsi **main()**, beberapa operasi antrian dilakukan: menambahkan elemen "Andi" dan "Maya" ke antrian, menampilkan antrian, menampilkan jumlah elemen dalam antrian, menghapus elemen terdepan, menampilkan antrian lagi, mengosongkan antrian, dan terakhir menampilkan antrian yang sudah kosong. Output dari program ini menunjukkan kondisi antrian setelah setiap operasi dilakukan.

2. Dari nomor 1 buatlah konsep antri dengan atribut Nama mahasiswa dan NIM Mahasiswa

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

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

class Queue {
private:
    Node* front;
    Node* back;
public:
    Queue() {
        front = nullptr;
        back = nullptr;
    }
};
```

```

~Queue() {
    clearQueue();
}

void enqueueAntrian(string nama, string nim) {
    Node* newNode = new Node();
    newNode->nama = nama;
    newNode->nim = nim;
    newNode->next = nullptr;
    if (isEmpty()) {
        front = newNode;
        back = newNode;
    } else {
        back->next = newNode;
        back = newNode;
    }
    cout << "Antrian ditambahkan: Nama: " << nama << ",
NIM: " << nim << endl;
}

void dequeueAntrian() {
    if (isEmpty()) {
        cout << "Antrian kosong" << endl;
        return;
    }
    Node* temp = front;
    cout << "Antrian dihapus: Nama: " << front->nama << ",
NIM: " << front->nim << endl;
    front = front->next;
    delete temp;
    if (front == nullptr) {
        back = nullptr;
    }
}

int countQueue() const {
    int count = 0;
    Node* current = front;
    while (current != nullptr) {
        count++;
        current = current->next;
    }
    return count;
}

void clearQueue() {
    while (!isEmpty()) {
        dequeueAntrian();
    }
    cout << "Antrian dikosongkan" << endl;
}

void viewQueue() const {
    if (isEmpty()) {
        cout << "Antrian kosong" << endl;
        return;
    }
    cout << "Data antrian mahasiswa:" << endl;
    Node* current = front;
    int position = 1;

```

```

        while (current != nullptr) {
            cout << position << ". Nama: " << current->nama <<
            ", NIM: " << current->nim << endl;
            current = current->next;
            position++;
        }
    }

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

int main() {
    Queue antrian;
    antrian.enqueueAntrian("NABILA SHASYA SABRINA",
        "2311102039");
    antrian.viewQueue();
    cout << "Jumlah antrian = " << antrian.countQueue() << endl;
    antrian.dequeueAntrian();
    antrian.viewQueue();
    cout << "Jumlah antrian = " << antrian.countQueue() << endl;
    antrian.clearQueue();
    antrian.viewQueue();
    cout << "Jumlah antrian = " << antrian.countQueue() << endl;
    return 0;
}

```

Screenshot Output:

```

Antrian ditambahkan: Nama: NABILA SHASYA SABRINA, NIM: 2311102039
Data antrian mahasiswa:
1. Nama: NABILA SHASYA SABRINA, NIM: 2311102039
Jumlah antrian = 1
Antrian dihapus: Nama: NABILA SHASYA SABRINA, NIM: 2311102039
Antrian kosong
Jumlah antrian = 0
Antrian dikosongkan
Antrian kosong
Jumlah antrian = 0
Antrian dikosongkan

```

Deskripsi Program:

Program ini adalah implementasi antrian (queue) menggunakan struktur data linked list di bahasa C++. Antrian direpresentasikan dalam kelas **Queue** yang menggunakan node untuk menyimpan data. Setiap node terdiri dari dua elemen: nama mahasiswa dan NIM, serta pointer ke node berikutnya.

Kelas **Queue** memiliki dua pointer privat, **front** dan **back**, yang digunakan untuk melacak elemen pertama dan terakhir dalam antrian. Konstruktor **Queue** menginisialisasi antrian sebagai kosong, dengan **front** dan **back** diatur ke **nullptr**. Destruktor **Queue** memastikan semua elemen antrian dihapus dengan memanggil fungsi **clearQueue()**.

Fungsi **enqueueAntrian()** menambahkan elemen baru ke antrian. Jika antrian

kosong, elemen baru menjadi elemen pertama dan terakhir. Jika tidak, elemen baru ditambahkan di belakang antrian. Fungsi ini juga menampilkan pesan bahwa elemen baru telah ditambahkan.

Fungsi **dequeueAntrian()** menghapus elemen dari depan antrian. Jika antrian kosong, pesan "Antrian kosong" ditampilkan. Jika tidak, elemen depan dihapus dan pointer **front** diperbarui ke elemen berikutnya. Jika setelah penghapusan antrian menjadi kosong, pointer **back** diatur ke **nullptr**.

Fungsi **countQueue()** menghitung dan mengembalikan jumlah elemen dalam antrian dengan mengiterasi dari depan ke belakang. Fungsi **clearQueue()** menghapus semua elemen antrian dengan memanggil **dequeueAntrian()** berulang kali hingga antrian kosong.

Fungsi **viewQueue()** menampilkan semua elemen dalam antrian beserta posisinya. Jika antrian kosong, pesan "Antrian kosong" ditampilkan. Jika tidak, informasi setiap elemen ditampilkan dengan format "Nama" dan "NIM". Fungsi **isEmpty()** mengembalikan nilai **true** jika antrian kosong, dan **false** jika tidak.

Pada fungsi **main()**, beberapa operasi antrian dilakukan: menambahkan elemen "NABILA SHASYA SABRINA" dengan NIM "2311102039" ke antrian, menampilkan antrian, menampilkan jumlah elemen dalam antrian, menghapus elemen terdepan, menampilkan antrian lagi, mengosongkan antrian, dan terakhir menampilkan antrian yang sudah kosong. Output dari program ini menunjukkan kondisi antrian setelah setiap operasi dilakukan.

BAB V

KESIMPULAN

Antrian adalah struktur data linier yang mengikuti prinsip FIFO (First In, First Out), di mana elemen yang pertama kali dimasukkan akan menjadi elemen pertama yang dihapus.

Dalam praktikum ini, kita belajar bagaimana membuat kelas **Queue** yang memanfaatkan node untuk menyimpan data elemen antrian. Kelas ini memiliki metode untuk menambahkan elemen ke antrian (**enqueueAntrian**), menghapus elemen dari antrian (**dequeueAntrian**), menghitung jumlah elemen dalam antrian (**countQueue**), mengosongkan antrian (**clearQueue**), dan menampilkan semua elemen antrian (**viewQueue**). Selain itu, metode **isEmpty** digunakan untuk memeriksa apakah antrian kosong.

Implementasi menggunakan linked list memungkinkan antrian memiliki ukuran yang dinamis dan efisien dalam penggunaan memori. Praktikum ini juga menunjukkan pentingnya manajemen memori yang baik, seperti menghapus elemen yang tidak diperlukan untuk menghindari kebocoran memori.

Dengan memahami dan mengimplementasikan antrian, kita memperoleh keterampilan penting dalam pemrograman struktur data yang dapat diterapkan dalam berbagai aplikasi komputer, seperti pengelolaan antrean di dunia nyata, penjadwalan tugas, dan banyak lagi. Praktikum ini memberikan dasar yang kuat untuk mempelajari struktur data yang lebih kompleks dan algoritma terkait.

DAFTAR PUSTAKA

Karumanchi, N. (2016). *Data Structures and algorithms made easy: Concepts, problems, Interview Questions*. CareerMonk Publications.