

**LAPORAN PRAKTIKUM STRUKTUR DATA DAN
PEMROGRAGAMAN**

**MODUL VII
QUEUE**



Disusun oleh :
RAFAALDHINO
2311102023
IF-11-A

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

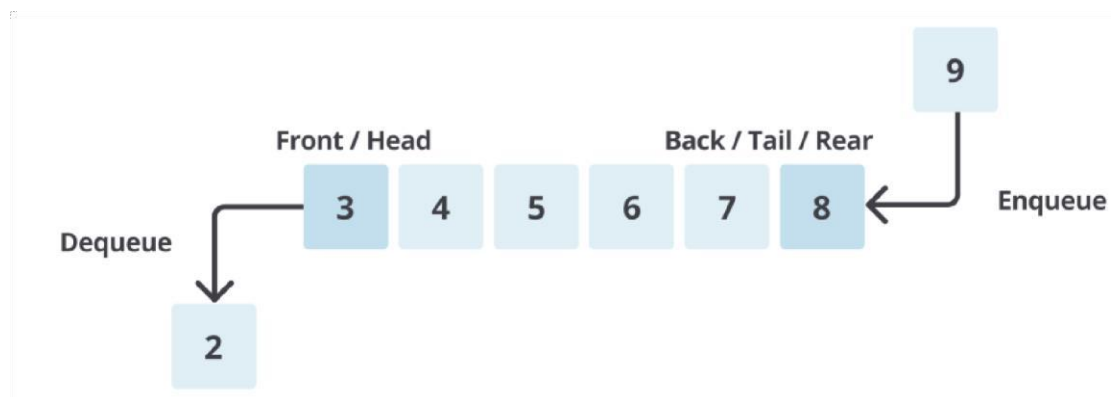
BAB II

DASAR TEORI

B. DASAR 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.



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

```

bool isEmpty()
{ // Antiannya 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;
}

```

OUTPUT

```
Jumlah antrian = 1
Data antrian teller:
1. (kosong)
2. (kosong)
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 0
PS D:\KULIAH\SEMESTER2\struktur data praktikum> |
```

DESKRIPSI PROGRAM

Program ini adalah implementasi sederhana dari struktur data antrian (queue) menggunakan array menggunakan C++. Program ini memiliki fungsi-fungsi dasar seperti enqueueAntrian untuk menambahkan elemen ke dalam antrian, dequeueAntrian untuk menghapus elemen dari antrian, isEmpty untuk memeriksa apakah antrian kosong, isFull untuk memeriksa apakah antrian penuh, countQueue untuk menghitung jumlah elemen dalam antrian, clearQueue untuk menghapus semua elemen dalam antrian, dan viewQueue untuk menampilkan elemen-elemen dalam antrian. Batas maksimum elemen dalam antrian ditentukan oleh konstanta maksimalQueue, dan posisi penandaan antrian ditandai oleh variabel front dan back.

BAB IV

Unguided

Unguided 1

```
#include <iostream>
using namespace std;
// Definisi struktur Node
struct Node
{
    string data;
    Node *next;
};
// Deklarasi variabel untuk front dan back
Node *front = nullptr;
Node *back = nullptr;
// Fungsi untuk mengecek apakah queue penuh
bool isFull()
{
    return false; // Linked list tidak terbatas ukuran
}
// Fungsi untuk mengecek apakah queue kosong
bool isEmpty()
{
    return front == nullptr;
}
// Fungsi untuk menambahkan elemen ke dalam queue
void enqueueAntrian(string data)
{
    Node *newNode = new Node();
    newNode->data = data;
    newNode->next = nullptr;

    if (isEmpty())
    {
        front = back = newNode;
    }
    else
    {
        back->next = newNode;
        back = newNode;
    }
}
```



```

// Fungsi untuk menghapus elemen dari queue
void dequeueAntrian()
{
    if (isEmpty())
    {
        cout << "Antrian kosong" << endl;
    }
    else
    {
        Node *temp = front;
        front = front->next;
        if (front == nullptr)
        {
            back = nullptr;
        }
        delete temp;
    }
}

// Fungsi untuk menghitung jumlah elemen dalam queue
int countQueue()
{
    int count = 0;
    Node *temp = front;
    while (temp != nullptr)
    {
        count++;
        temp = temp->next;
    }
    return count;
}

// Fungsi untuk menghapus semua elemen dalam queue
void clearQueue()
{
    while (!isEmpty())
    {
        dequeueAntrian();
    }
}

// Fungsi untuk menampilkan elemen dalam queue
void viewQueue()
{
    cout << "Data antrian teller:" << endl;
    Node *temp = front;
    int index = 1;
    while (temp != nullptr)
    {
        cout << index << ". " << temp->data << endl;
        temp = temp->next;
        index++;
    }
    if (index == 1)
    {
        cout << "Antrian kosong" << endl;
    }
}

```

```

int main()
{
    enqueueAntrian("budi");
    enqueueAntrian("ani");
    viewQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;
    dequeueAntrian();
    viewQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;
    clearQueue();
    viewQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;
    return 0;
}

```

OUTPUT

```

Data antrian teller:
1. budi
2. ani
Jumlah antrian = 2
Data antrian teller:
1. ani
Jumlah antrian = 1
Data antrian teller:
Antrian kosong
Jumlah antrian = 0

```

DESKRIPSI PROGRAM

Setiap elemen dalam antrian direpresentasikan oleh sebuah simpul (Node) yang memiliki dua bagian data: data yang merupakan informasi yang disimpan dalam antrian, dan pointer next yang menunjuk ke simpul berikutnya dalam antrian. Fungsi-fungsi seperti enqueueAntrian, dequeueAntrian, isEmpty, countQueue, dan clearQueue digunakan untuk menambah, menghapus, memeriksa kekosongan, menghitung jumlah, dan menghapus semua elemen dalam antrian secara berturut-turut. Program ini juga menyediakan fungsi viewQueue untuk menampilkan seluruh elemen yang ada dalam antrian. Dalam contoh ini, elemen-elemen "budi" dan "ani" ditambahkan ke dalam antrian, kemudian satu elemen dihapus dari antrian, dan akhirnya semua elemen dihapus dari antrian..

Unguided 2

```

#include <iostream>
using namespace std;

// Definisi struktur Node dengan atribut nama mahasiswa dan NIM mahasiswa
struct Node
{
    string nama;
    string nim;
    Node *next;
};

// Deklarasi variabel untuk front dan back
Node *front = nullptr;
Node *back = nullptr;

// Fungsi untuk mengecek apakah queue kosong
bool isEmpty()
{
    return front == nullptr;
}

// Fungsi untuk menambahkan elemen ke dalam queue
void enqueueAntrian(string nama, string nim)
{
    Node *newNode = new Node();
    newNode->nama = nama;
    newNode->nim = nim;
    newNode->next = nullptr;

    if (isEmpty())
    {
        front = back = newNode;
    }
    else
    {
        back->next = newNode;
        back = newNode;
    }
}

// Fungsi untuk menghapus elemen dari queue
void dequeueAntrian()
{
    if (isEmpty())
    {
        cout << "Antrian kosong" << endl;
    }
    else
    {
        Node *temp = front;
        front = front->next;
        delete temp;

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

```

```

// Fungsi untuk menghitung jumlah elemen dalam queue
int countQueue()
{
    int count = 0;
    Node *temp = front;
    while (temp != nullptr)
    {
        count++;
        temp = temp->next;
    }
    return count;
}

// Fungsi untuk menghapus semua elemen dalam queue
void clearQueue()
{
    while (!isEmpty())
    {
        dequeueAntrian();
    }
}

// Fungsi untuk menampilkan elemen dalam queue
void viewQueue()
{
    cout << "Data antrian mahasiswa:" << endl;
    Node *temp = front;
    int index = 1;
    while (temp != nullptr)
    {
        cout << index << ". Nama: " << temp->nama << ", NIM: " << temp->nim << endl;
        temp = temp->next;
        index++;
    }
    if (index == 1)
    {
        cout << "Antrian kosong" << endl;
    }
}

int main()
{
    enqueueAntrian("Rafa Aldhino", "2311102023");
    enqueueAntrian("Sigma", "2311102099");
    viewQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;
    dequeueAntrian();
    viewQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;
    clearQueue();
    viewQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;
    return 0;
}

```

OUTPUT

```
Data antrian mahasiswa:
1. Nama: Rafa Aldhino, NIM: 231110202
2. Nama: Sigma, NIM: 2311102099
Jumlah antrian = 2
Data antrian mahasiswa:
1. Nama: Sigma, NIM: 2311102099
Jumlah antrian = 1
Data antrian mahasiswa:
Antrian kosong
Jumlah antrian = 0
```

DESKRIPSI PROGRAM

Setiap elemen dalam antrian direpresentasikan oleh sebuah simpul (Node) yang memiliki dua atribut: nama dan NIM mahasiswa, serta pointer next yang menunjuk ke simpul berikutnya dalam antrian. Fungsi-fungsi seperti enqueueAntrian, dequeueAntrian, isEmpty, countQueue, dan clearQueue digunakan untuk menambahkan, menghapus, memeriksa kekosongan, menghitung jumlah, dan menghapus semua elemen dalam antrian secara berturut-turut. Program ini juga menyediakan fungsi viewQueue untuk menampilkan seluruh elemen yang ada dalam antrian berserta nama dan NIM mahasiswanya. Dalam contoh ini, dua mahasiswa dengan nama "Rafa Aldhino" dan "Sigma" ditambahkan ke dalam antrian, kemudian satu elemen dihapus dari antrian, dan akhirnya semua elemen dihapus dari antrian.

BAB V

KESIMPULAN

Queue atau antrian adalah struktur data yang mengikuti prinsip FIFO (First In First Out), di mana elemen yang ditambahkan pertama kali akan dihapus pertama kali. Dalam C++, queue diimplementasikan sebagai container adapter, yang menggunakan objek terkapsulasi dari kelas container spesifik (seperti deque atau list) sebagai wadah utamanya.

Ada juga kegunaan utamanya yaitu:

- Mensimulasikan antrian dunia nyata: Queue dapat digunakan untuk mensimulasikan antrian dunia nyata, seperti antrian di kasir, antrian di pompa bensin, atau antrian di bandara.
- Menyimpan data yang akan diproses nanti: Queue dapat digunakan untuk menyimpan data yang akan diproses nanti, seperti data input dari pengguna, data yang dihasilkan dari satu bagian program, atau data yang perlu diprioritaskan.
- Memimplementasikan algoritma: Queue dapat digunakan untuk mengimplementasikan berbagai algoritma, seperti algoritma pencarian breadth-first, algoritma sorting, dan algoritma scheduling.

E. DAFTAR PUSTAKA

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