LAPORAN PRAKTIKUM

MODUL 7
QUEUE

Disusun oleh:
Denny Budiansyach
NIM: 2311102022

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

BAB II
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 1

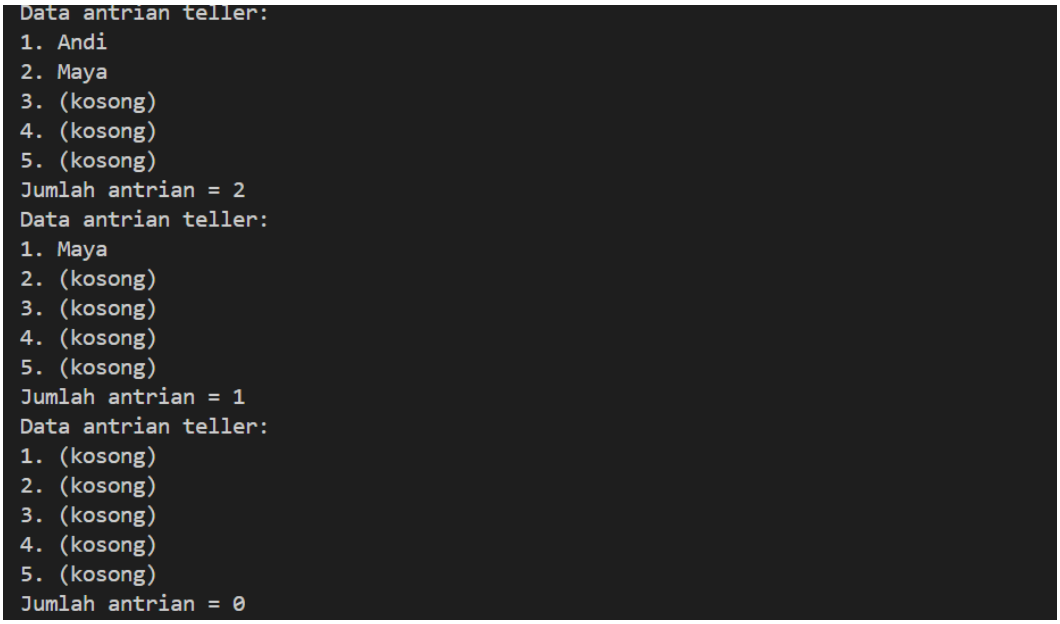
Guided 1 Source code

```
#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;
}
```

Screenshoot program



Deskripsi program

Program di atas adalah bentuk implementasi dari Queue yang menggunakan array untuk menyimpan dan mengelola data antrian. Dengan menerapkan prinsip queue maka setiap data akan disimpan pada program dengan metode FIFO (First In, First Out), di mana data pertama yang dimasukkan adalah data pertama yang dapat diambil. Program ini menggunakan beberapa fungsi untuk mengelola queue, seperti enqueueAntrian(string data) untuk menambahkan antrian, dequeueAntrian() untuk mengurangi antrian, clearQueue() untuk menghapus semua antrian, viewQueue() untuk menampilkan antrian, dan fungsi fungsi pemeriksaan status stack.

LATIHAN KELAS - UNGUIDED

1. **Unguided Source code**

```
#include <iostream>
using namespace std;
struct Node {
    string data;
    Node* next;
};

Node* front = nullptr;
Node* back = nullptr;

bool isEmpty() {
    return (front == nullptr);
}
void enqueue(string nama) {
    Node* newNode = new Node;
    newNode->data = nama;
    newNode->next = nullptr;
    if (isEmpty()) {
        front = back = newNode;
    } else {
        back->next = newNode;
        back = newNode;
    }
}
void dequeue() {
    if (isEmpty()) {
        cout << "Antrian kosong" << endl;
    } else {
        Node* temp = front;
        front = front->next;
        if (front == nullptr) {
            back = nullptr;
        }
        delete temp;
    }
}
int countQueue() {
    int count = 0;
```

```
    Node* current = front;
    while (current != nullptr) {
        count++;
        current = current->next;
    }
    return count;
}
void clearQueue() {
    while (!isEmpty()) {
        dequeue();
    }
}
void viewQueue() {
    if (!isEmpty()) {
        Node* current = front;
        cout << "Data antrian teller: \n";
        int count = 0;
        while (current != nullptr) {
            count++;
            cout << count <<". "<< current->data << "\n";
            current = current->next;
        }
        cout << endl;
    } else {
        cout << "(kosong)" << endl;
    }
}

int main()
{
    enqueue("Andi");
    enqueue("Maya");
    cout << "Jumlah antrian = " << countQueue() << endl;
    viewQueue();
    dequeue();
    cout << "Jumlah antrian = " << countQueue() << endl;
    viewQueue();
    clearQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;
    viewQueue();
    return 0;
}
```

Screenshoot program

Deskripsi program

Program di atas adalah bentuk implementasi dari Queue yang menggunakan Linked List untuk menyimpan dan mengelola data antrian. Dengan menerapkan prinsip FIFO (First In, First Out) maka node head atau front akan menjadi node pertama yang akan diakses. Program ini menggunakan beberapa fungsi untuk mengelola queue, seperti enqueue(string data) untuk menambahkan antrian, dequeue () untuk mengurangi antrian, countQueue() untuk menampilkan jumlah antrian, clearQueue() untuk menghapus semua antrian, viewQueue() untuk menampilkan antrian, dan fungsi fungsi pemeriksaan status stack. Berikut adalah penjelasan untuk fungsi diatas:

1. enqueue(string data)

   Fungsi enqueue berfungsi untuk menambahkan data baru ke dalam antrian. fungsi ini akan membuat node baru untuk data yang akan dimasukan kedalam antrian. Kemudian fungsi ini akan melakukan pengecekan pada pantrian, jika antrian kosong, node front dan back akan diinisialisasi ke node baru tersebut. jika tidak, node baru akan ditambahkan ke belakang antrian melalui pointer next pada node back, dan node back akan diperbarui untuk menunjuk ke node baru yang telah ditambahkan dalam antrian.
2. dequeue()

   Fungsi dequeue berfungsi untuk menghapus data dari depan antrian. Fungsi ini akan melakukan pengecekan pada antrian, jika antrian kosong maka fungsi akan mencetak string "Antrian kosong". Jika ternyata terdapat antrian, maka node di depan antrian atau node front akan dihapus dan node front akan

diperbarui untuk menunjuk ke node berikutnya. Jika setelah penghapusan antrian hingga value pada node front menjadi nullptr, maka value node back juga diset ke nullptr.

3. countQueue()

   Fungsi countQueue berfungsi untuk menghitung jumlah elemen dalam antrian dengan mengakses setiap pointer next pada node antrian dari node front hingga node yang pointer next nya berisi nullptr. Pada setiap iterasi tersebut variabel count akan diincrement, yang kemudian value dari variabel count akan di return sebagai jumlah elemen dalam antrian.

4. clearQueue()

   Fungsi clearQueue berfungsi untuk menghapus semua elemen dalam antrian dengan memanggil fungsi dequeue secara berulang hingga antrian kosong yang ditunjukan oleh fungsi isEmpty().

5. viewQueue()

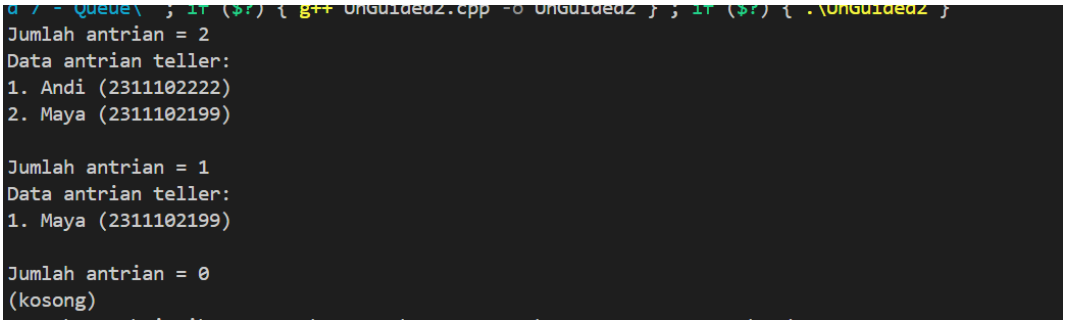
   Fungsi viewQueue berfungsi untuk menampilkan semua data dalam antrian dengan mengakses dan mencetak isi setiap node pada antrian dari node front hingga node terakhir. Jika antrian kosong, fungsi viewQueue akan mencetak string"(kosong)".

2. **Unguided Source code**

```
#include <iostream>
using namespace std;
struct Node {
    string nama;
    string nim;
    Node* next;
};

Node* front = nullptr;
Node* back = 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 = back = newNode;
    } else {
        back->next = newNode;
        back = newNode;
    }
}
void dequeue() {
    if (isEmpty()) {
        cout << "Antrian kosong" << endl;
    } else {
        Node* temp = front;
        front = front->next;
        if (front == nullptr) {
            back = nullptr;
        }
        delete temp;
    }
}
int countQueue() {
    int count = 0;
    Node* current = front;
```

```
    while (current != nullptr) {
        count++;
        current = current->next;
    }
    return count;
}
void clearQueue() {
    while (!isEmpty()) {
        dequeue();
    }
}
void viewQueue() {
    if (!isEmpty()) {
        Node* current = front;
        cout << "Data antrian teller: \n";
        int count = 0;
        while (current != nullptr) {
            count++;
            cout << count <<". "<< current->nama << " (" << current-
>nim << ")"<< "\n";
            current = current->next;
        }
        cout << endl;
    } else {
        cout << "(kosong)" << endl;
    }
}

int main() {
    enqueue("Andi", "2311102222");
    enqueue("Maya", "2311102199");
    cout << "Jumlah antrian = " << countQueue() << endl;
    viewQueue();
    dequeue();
    cout << "Jumlah antrian = " << countQueue() << endl;
    viewQueue();
    clearQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;
    viewQueue();
    return 0;
}
```

Screenshoot program



Deskripsi program

Pada dasarnya program UnGuided 2 ini merupakan program update dari UnGuided 1. Jika pada UnGuided 1 data yang terdapat pada setiap node hanya berisikan 1 data string saja, maka pada UnGuided 2 terdapat 2 data string pada setiap node nya. 2 data string pada setiap node ini akan digunakan untuk menyimpan nama dan nim mahasiswa.

BAB IV
KESIMPULAN

Queue adalah sebuah struktur data yang menerapkan konsep FIFO (First-In First-Out), di mana data yang pertama dimasukkan akan menjadi data pertama yang dapat diakses atau dikeluarkan. Implementasi dari queue dapat dilakukan menggunakan array atau linked list. Pada implementasi array, variabel front dan back digunakan untuk penanda elemen antrian pada array. Sedangkan pada implementasi Linked List menggunakan node head akan menunjuk pada elemen pertama dalam antrian dan node back akan menunjuk pada elemn terakhir yang ditambahkan dalam antrian.

DAFTAR PUSTAKA

GeeksforGeeks. (2024, Mei 11). Diambil kembali dari geeksforgeeks.com: https://www.geeksforgeeks.org/queue-data-structure/

Asisten Praktikum. (2024). *MODUL 7 - QUEUE*, Learning Managament System.