LAPORAN PRAKTIKUM

MODUL 7
QUEUE

Disusun oleh:
Rakha Yudhistira
NIM: 2311102010

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

Stack adalah struktur data sederhana yang digunakan untuk menyimpan data (mirip dengan Linked Lists). Dalam tumpukan, urutan kedatangan data penting. Sebuah tumpukan piring di kafetaria adalah contoh bagus dari tumpukan. Piring ditambahkan ke tumpukan saat mereka dibersihkan dan ditempatkan di bagian atas. Ketika sebuah piring dibutuhkan, diambil dari bagian atas tumpukan. Piring pertama yang ditempatkan di tumpukan adalah yang terakhir digunakan.

Definisi: Sebuah tumpukan adalah daftar terurut di mana penyisipan dan penghapusan dilakukan di satu ujung, disebut atas. Elemen terakhir yang dimasukkan adalah yang pertama dihapus. Oleh karena itu, disebut daftar Last in First out (LIFO).

Operasi pada stack melibatkan beberapa fungsi dasar yang dapat dilakukan pada struktur data ini. Berikut adalah beberapa operasi umum pada stack:

a. Push (Masukkan): Menambahkan elemen ke dalam tumpukan pada posisi paling atas atau ujung.
b. Pop (Keluarkan): Menghapus elemen dari posisi paling atas atau ujung tumpukan.
c. Top (Atas): Mendapatkan nilai atau melihat elemen teratas pada tumpukan tanpa menghapusnya.
d. IsEmpty (Kosong): Memeriksa apakah tumpukan kosong atau tidak.
e. IsFull (Penuh): Memeriksa apakah tumpukan penuh atau tidak (terutama pada implementasi tumpukan dengan kapasitas terbatas).
f. Size (Ukuran): Mengembalikan jumlah elemen yang ada dalam tumpukan.
g. Peek (Lihat): Melihat nilai atau elemen pada posisi tertentu dalam tumpukan tanpa menghapusnya.
h. Clear (Hapus Semua): Mengosongkan atau menghapus semua elemen dari tumpukan.
i. Search (Cari): Mencari keberadaan elemen tertentu dalam tumpukan.

BAB III

GUIDED

1. Guided 1

Source code

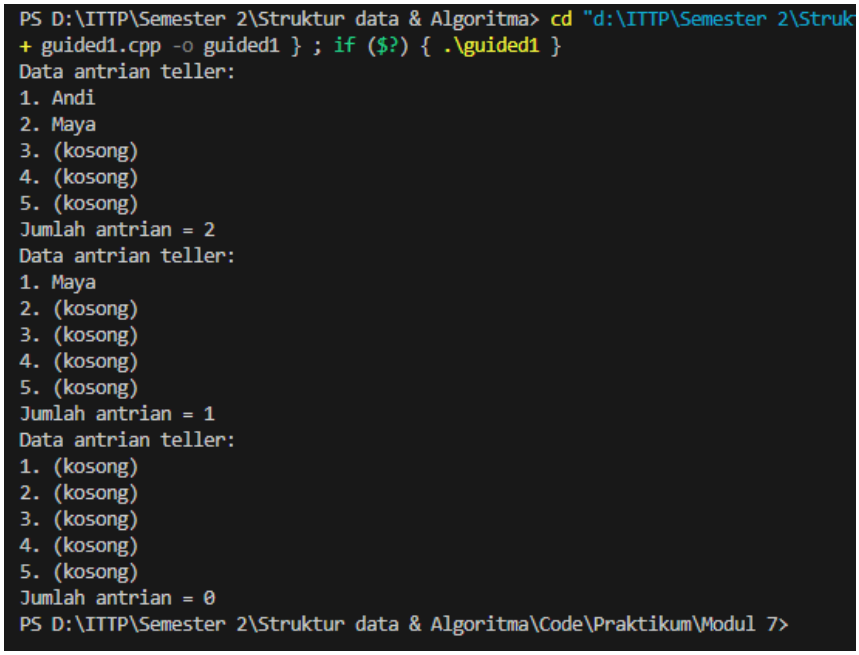
```
#include <iostream>
using namespace std;
const int maksimalQueue = 5; // Maksimal antrian
int front = 0;               // Penanda antrian
int back = 0;                // Penanda
string queueTeller[5];       // Fungsi pengecekan
bool isFull()
{ // Pengecekan antrian penuh atau tidak
    if (back == 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 implementasi antrian (queue) sederhana menggunakan array dalam bahasa C++. Program ini mencakup fungsi-fungsi untuk mengelola antrian, seperti menambah elemen (enqueue), menghapus elemen (dequeue), memeriksa apakah antrian penuh (isFull) atau kosong (isEmpty), menghitung jumlah elemen (countQueue), menghapus semua elemen (clearQueue), dan menampilkan elemen dalam antrian (viewQueue). Antrian memiliki kapasitas maksimal yang ditentukan oleh maksimalQueue, dan elemen-elemen disimpan dalam array queueTeller. Fungsi utama (main) mendemonstrasikan penggunaan fungsi-fungsi ini dengan menambahkan elemen ke antrian, menampilkan antrian, menghitung jumlah elemen, menghapus elemen, dan mengosongkan antrian, serta menampilkan hasil pada setiap tahap.

LATIHAN KELAS - UNGUIDED

1. **Unguided 1**

Source code

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

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

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

public:
    Queue()
    {
```

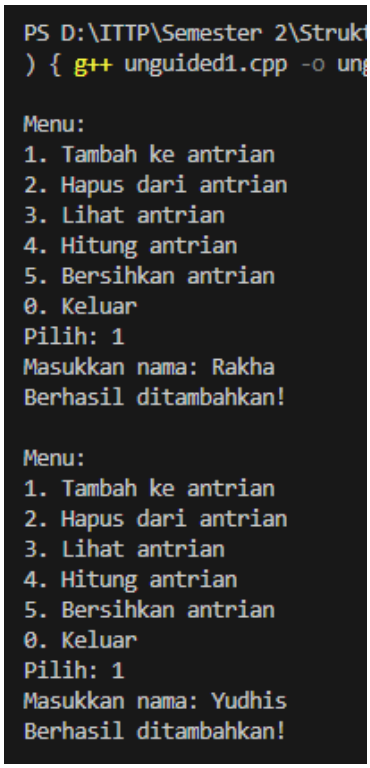
```
        front = nullptr;
        back = nullptr;
    }

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

    void enqueue(string data)
    {
        Node *temp = new Node();
        temp->data = data;
        temp->next = nullptr;

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

    void dequeue()
    {
        if (isEmpty())
        {
            cout << "Antrian kosong" << endl;
        }
        else
```

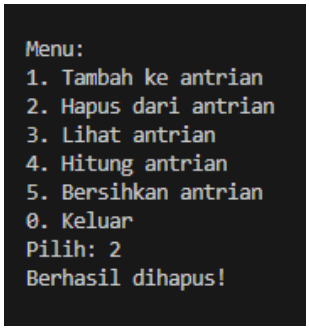
```
        {
            Node *temp = front;
            front = front->next;
            delete temp;
            if (front == nullptr)
            {
                back = nullptr;
            }
        }
    }

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

    void clearQueue()
    {
        while (!isEmpty())
        {
            dequeue();
        }
    }

    void viewQueue()
    {
        cout << "Data antrian pembeli:" << endl;
```

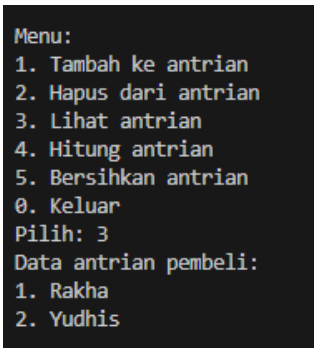
```
        Node *current = front;
        int index = 1;
        while (current != nullptr)
        {
            cout << index << ". " << current->data << endl;
            current = current->next;
            index++;
        }
        if (isEmpty())
        {
            cout << "(kosong)" << endl;
        }
    }
};

int main()
{
    Queue q;
    int choice;
    string name;

    do
    {
        cout << "\nMenu:\n";
        cout << "1. Tambah ke antrian \n";
        cout << "2. Hapus dari antrian \n";
        cout << "3. Lihat antrian\n";
        cout << "4. Hitung antrian\n";
        cout << "5. Bersihkan antrian\n";
        cout << "0. Keluar\n";
        cout << "Pilih: ";
        cin >> choice;

        switch (choice)
```

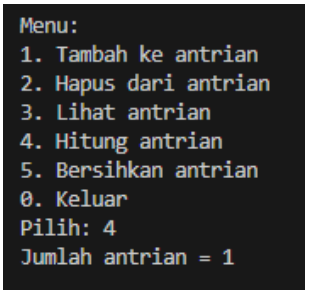
```
        {
        case 1:
            cout << "Masukkan nama: ";
            cin >> name;
            q.enqueue(name);
            cout << "Berhasil ditambahkan!" << endl;
            break;
        case 2:
            q.dequeue();
            cout << "Berhasil dihapus!" << endl;
            break;
        case 3:
            q.viewQueue();
            break;
        case 4:
            cout << "Jumlah antrian = " << q.countQueue() <<
endl;
            break;
        case 5:
            q.clearQueue();
            break;
        case 0:
            cout << "Keluar dari program." << endl;
            break;
        default:
            cout << "Pilihan tidak valid. Silakan coba lagi." <<
endl;
        }
    } while (choice != 0);

    return 0;
}
```

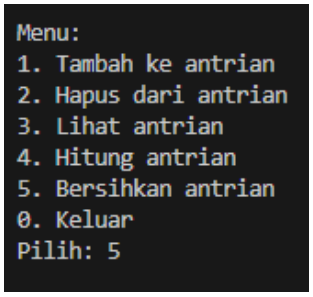
Screenshoot program











Deskripsi program

Program di atas mengimplementasikan antrian (queue) menggunakan struktur data linked list dalam bahasa C++. Program ini terdiri dari dua kelas utama: Node dan QueueList. Kelas Node merepresentasikan setiap elemen antrian dengan atribut data dan next. Kelas QueueList mengelola antrian dengan atribut front dan back, yang menunjuk ke elemen pertama dan terakhir. Metode yang tersedia meliputi enqueue (menambah elemen), dequeue (menghapus elemen), countQueue (menghitung jumlah elemen), clearQueue (menghapus semua elemen), dan viewQueue (menampilkan semua elemen). Dalam fungsi main, program menyediakan menu interaktif untuk pengguna, memungkinkan operasi seperti menambah, menghapus, menghitung, mengosongkan, melihat elemen dalam antrian, dan keluar dari program. Antrian ini memanfaatkan linked list untuk manajemen data yang dinamis dan efisien.

2. **Unguided 2**

Source code

```
#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;
    }

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

    void enqueue(string nama, string NIM)
```

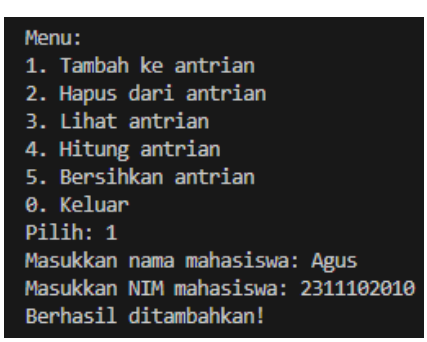
```
    {
        Node *temp = new Node();
        temp->nama = nama;
        temp->NIM = NIM;
        temp->next = nullptr;

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

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

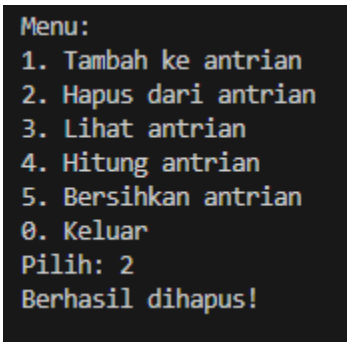
```
    }

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

    void clearQueue()
    {
        while (!isEmpty())
        {
            dequeue();
        }
    }

    void viewQueue()
    {
        cout << "Data antrian mahasiswa:" << endl;
        Node *current = front;
        int index = 1;
        while (current != nullptr)
        {
            cout << index << ". Nama: " << current->nama << ",
NIM: " << current->NIM << endl;
            current = current->next;
            index++;
        }
```

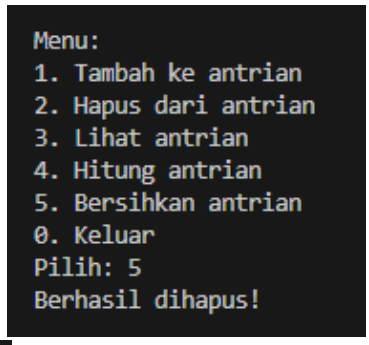
```
        if (isEmpty())
        {
            cout << "(kosong)" << endl;
        }
    }
};

int main()
{
    Queue q;
    int choice;
    string nama, NIM;

    do
    {
        cout << "\nMenu:\n";
        cout << "1. Tambah ke antrian \n";
        cout << "2. Hapus dari antrian \n";
        cout << "3. Lihat antrian\n";
        cout << "4. Hitung antrian\n";
        cout << "5. Bersihkan antrian\n";
        cout << "0. Keluar\n";
        cout << "Pilih: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "Masukkan nama mahasiswa: ";
            cin.ignore();
            getline(cin, nama);
            cout << "Masukkan NIM mahasiswa: ";
            cin >> NIM;
            q.enqueue(nama, NIM);
```

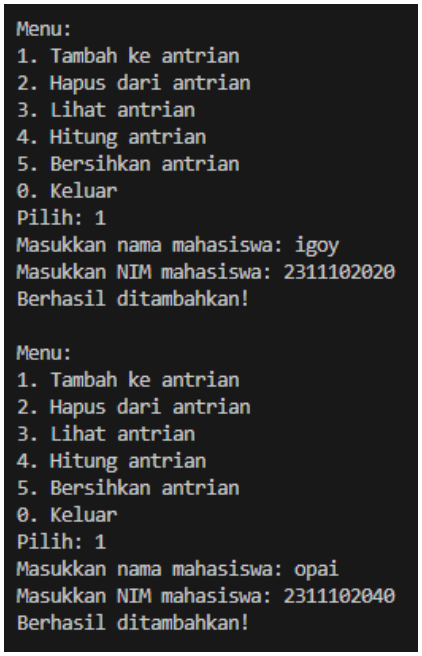
```
            cout << "Berhasil ditambahkan!" << endl;
            break;
        case 2:
            q.dequeue();
            cout << "Berhasil dihapus!" << endl;
            break;
        case 3:
            q.viewQueue();
            break;
        case 4:
            cout << "Jumlah antrian = " << q.countQueue() <<
endl;
            break;
        case 5:
            q.clearQueue();
            cout << "Berhasil dihapus!" << endl;
            break;
        case 0:
            cout << "Keluar dari program." << endl;
            break;
        default:
            cout << "Pilihan tidak valid. Silakan coba lagi." <<
endl;
        }
    } while (choice != 0);

    return 0;
}
```

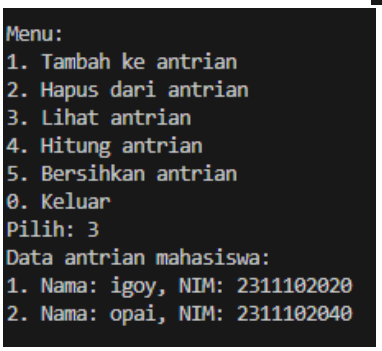
Screenshoot program

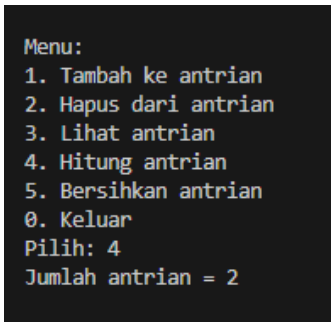












Deskripsi program

Program di atas mengimplementasikan antrian (queue) menggunakan linked list dalam bahasa C++. Kelas Node merepresentasikan elemen antrian dengan atribut nim (Nomor Induk Mahasiswa), data, dan next yang menunjuk ke node berikutnya. Kelas QueueList mengelola antrian dengan atribut front dan back yang menunjuk ke elemen pertama dan terakhir. Metode yang disediakan meliputi enqueue (menambah elemen baru), dequeue (menghapus elemen dari depan), countQueue (menghitung elemen), clearQueue (menghapus semua elemen), dan

viewQueue (menampilkan elemen). Fungsi main menyediakan menu interaktif untuk pengguna, memungkinkan operasi seperti menambah elemen dengan NIM dan data, menghapus elemen, menghitung elemen, mengosongkan antrian, melihat elemen, dan keluar dari program. Linked list digunakan untuk manajemen data yang dinamis dan efisien dalam penggunaan memori.

BAB IV

KESIMPULAN

Struktur data queue (antrian) adalah sebuah konsep fundamental dalam ilmu komputer yang menerapkan prinsip FIFO (First In, First Out), di mana elemen yang pertama kali dimasukkan akan menjadi elemen pertama yang dikeluarkan. Beberapa karakteristik utama dan metode yang umum digunakan dalam queue meliputi:

1. Enqueue: Menambahkan elemen baru ke akhir antrian.
2. Dequeue: Menghapus elemen dari depan antrian.
3. IsEmpty: Memeriksa apakah antrian kosong.
4. IsFull: Memeriksa apakah antrian penuh (terutama relevan untuk implementasi dengan array).
5. CountQueue: Menghitung jumlah elemen dalam antrian.
6. ClearQueue: Menghapus semua elemen dalam antrian.
7. ViewQueue: Menampilkan semua elemen dalam antrian.

Queue dapat diimplementasikan menggunakan array atau linked list. Implementasi dengan array memiliki keterbatasan kapasitas tetap, sementara linked list menawarkan fleksibilitas dan efisiensi memori karena dapat tumbuh dan menyusut secara dinamis sesuai kebutuhan.

Queue digunakan dalam berbagai aplikasi, termasuk manajemen tugas di sistem operasi, penanganan antrian di printer, routing paket dalam jaringan, dan pemrosesan data dalam algoritma breadth-first search (BFS). Memahami dan mengimplementasikan queue dengan baik adalah penting untuk pengembangan perangkat lunak yang efisien dan responsif.

DAFTAR PUSTAKA

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

Soden,Syarif. (05 Mei 2020). Pengertian Queue Dalam C++. Diakses pada 28 Mei 2024 dari https://www.kaskus.co.id/thread/5ec54d20facb95558a5496e1/pengertian-queue-dalam-c