LAPORAN PRAKTIKUM

MODUL VII
QUEUE

Disusun oleh:
Nadhif Atha Zaki
NIM: 2311102007

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

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2024

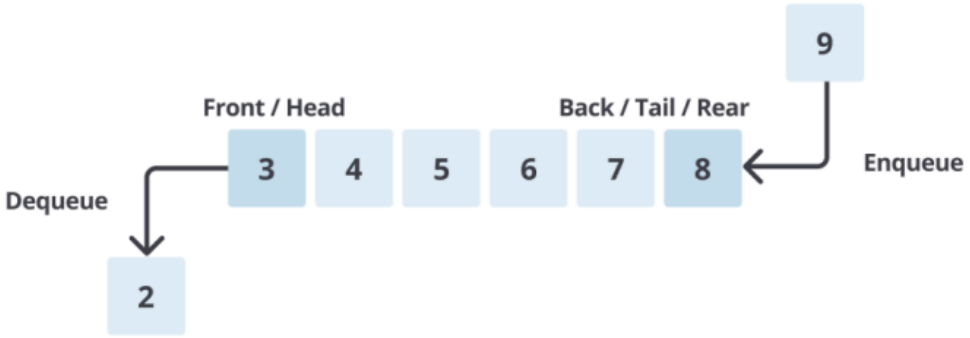
BAB I
TUJUAN PRAKTIKUM

a. Mahasiswa mampu menjelaskan definisi dan konsep dari double queue
b. Mahasiswa mampu menerapkan operasi tambah, menghapus pada queue
c. Mahasiswa 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:

```
PS D:\Nathhh\matkul\smst 2\praktikum struktur data\modul 7 - queue> cd "
ul\smst 2\praktikum struktur data\modul 7 - queue\" ; if ($?) { g++ guid
ded1 } ; if ($?) { .\guided1 }
Data antrian teller:
1. Andi
2. Maya
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 2
Data antrian teller:
1. Maya
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
PS D:\Nathhh\matkul\smst 2\praktikum struktur data\modul 7 - queue>
```

Deskripsi program:

Program di atas adalah implementasi antrian (queue) menggunakan array dengan ukuran maksimal lima elemen. Fungsi `isFull()` memeriksa apakah antrian sudah penuh, sedangkan `isEmpty()` memeriksa apakah antrian kosong. Fungsi `enqueueAntrian()` menambahkan elemen baru ke antrian, dan `dequeueAntrian()` menghapus elemen paling depan dari antrian. Fungsi `countQueue()` menghitung jumlah elemen dalam antrian, dan `clearQueue()` menghapus semua elemen di dalam antrian. Fungsi `viewQueue()` menampilkan isi antrian, menunjukkan elemen mana yang kosong dan mana yang terisi..

LATIHAN KELAS - 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;
};

Node *front = nullptr; // Pointer ke node depan
Node *back = nullptr;  // Pointer ke node belakang

bool isFull()
{ // Dalam implementasi linked list, antrian tidak pernah penuh
    return false;
}

bool isEmpty()
{ // Memeriksa apakah antrian kosong
    return front == nullptr;
}

void enqueueAntrian(string data)
{ // Fungsi untuk menambahkan elemen ke antrian
    Node *newNode = new Node();
    newNode->data = data;
    newNode->next = nullptr;
    if (isEmpty())
    {
        front = newNode;
        back = newNode;
```

```
    }
    else
    {
        back->next = newNode;
        back = newNode;
    }
}

void dequeueAntrian()
{ // Fungsi untuk mengurangi elemen dari antrian
    if (isEmpty())
    {
        cout << "Antrian kosong" << endl;
    }
    else
    {
        Node *temp = front;
        front = front->next;
        if (front == nullptr)
        {
            back = nullptr; // Jika antrian sekarang kosong,
perbarui pointer back
        }
        delete temp;
    }
}

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

void clearQueue()
{ // Fungsi untuk menghapus semua elemen dari antrian
```

```
    while (!isEmpty())
    {
        dequeueAntrian();
    }
}

void viewQueue()
{ // Fungsi untuk melihat elemen dalam antrian
    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("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

```
PS D:\Nathhh\matkul\smst 2\praktikum struktur data\modul 7 - queue> cd "d:\Nathhh\matkul\smst 2\praktikum struktur data\modul 7 - queue\" ; if ($?) { g++ unguided1.cpp -o unguided1 } ; if ($?) { .\unguided1 }
Data antrian teller:
1. Andi
2. Maya
Jumlah antrian = 2
Data antrian teller:
1. Maya
Jumlah antrian = 1
Data antrian teller:
Antrian kosong
Jumlah antrian = 0
PS D:\Nathhh\matkul\smst 2\praktikum struktur data\modul 7 - queue>
```

Deskripsi dan Penjelasan Program:

Program di atas adalah implementasi antrian (queue) menggunakan linked list dalam bahasa C++. Pada awalnya, dua pointer global `front` dan `back` diinisialisasi untuk menunjuk ke elemen pertama dan terakhir dari antrian. Struktur `Node` digunakan untuk menyimpan data dan pointer ke node berikutnya. Fungsi `isFull()` selalu mengembalikan false karena dalam linked list, antrian tidak memiliki batas ukuran yang tetap. Fungsi `isEmpty()` memeriksa apakah antrian kosong dengan melihat apakah pointer `front` bernilai `nullptr`.

Fungsi `enqueueAntrian()` menambahkan elemen baru ke akhir antrian. Jika antrian kosong, node baru ini menjadi `front` dan `back`. Jika tidak, node baru ditambahkan di belakang dan pointer `back` diperbarui. Fungsi `dequeueAntrian()` menghapus elemen dari depan antrian dan menghapus memori node tersebut. Jika antrian menjadi kosong setelah penghapusan, pointer `back` juga diatur menjadi `nullptr`.

Fungsi `countQueue()` menghitung jumlah elemen dalam antrian dengan melakukan traversal dari `front` ke `back`. Fungsi `clearQueue()` menghapus semua elemen dalam antrian dengan memanggil `dequeueAntrian()` hingga antrian kosong. Fungsi `viewQueue()` menampilkan semua elemen dalam antrian, menunjukkan data

masing-masing node dari depan hingga belakang. Pada fungsi `main()`, beberapa operasi antrian dilakukan, termasuk menambahkan elemen, menampilkan elemen, menghitung jumlah elemen, menghapus elemen, dan menghapus semua elemen dari antrian.

2. .

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

Souce Code:

```
#include <iostream>

using namespace std;

struct Mahasiswa
{
    string nama;
    string nim;
};

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

Node *front = nullptr; // Pointer ke node depan
Node *back = nullptr;  // Pointer ke node belakang
```

```
bool isFull()
{ // Dalam implementasi linked list, antrian tidak pernah penuh
    return false;
}

bool isEmpty()
{ // Memeriksa apakah antrian kosong
    return front == nullptr;
}

void enqueueAntrian(string nama, string nim)
{ // Fungsi untuk menambahkan elemen ke antrian
    Node *newNode = new Node();
    newNode->data.nama = nama;
    newNode->data.nim = nim;
    newNode->next = nullptr;
    if (isEmpty())
    {
        front = newNode;
        back = newNode;
    }
    else
    {
        back->next = newNode;
        back = newNode;
    }
}

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

```
            back = nullptr; // Jika antrian sekarang kosong,
perbarui pointer back
        }
        delete temp;
    }
}

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

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

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

```
        cout << "Antrian kosong" << endl;
    }
}

int main()
{
    enqueueAntrian("Andi", "123456789");
    enqueueAntrian("Maya", "987654321");
    viewQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;
    dequeueAntrian();
    viewQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;
    clearQueue();
    viewQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;
    return 0;
}
```

Screenshot Program:

```
PS D:\Nathhh\matkul\smst 2\praktikum struktur data\modul 7 - queue> cd "d:\Nathhh
ul\smst 2\praktikum struktur data\modul 7 - queue\" ; if ($?) { g++ unguided2.cpp
nguided2 } ; if ($?) { .\unguided2 }
Data antrian mahasiswa:
1. Nama: Andi, NIM: 123456789
2. Nama: Maya, NIM: 987654321
Jumlah antrian = 2
Data antrian mahasiswa:
1. Nama: Maya, NIM: 987654321
Jumlah antrian = 1
Data antrian mahasiswa:
Antrian kosong
Jumlah antrian = 0
PS D:\Nathhh\matkul\smst 2\praktikum struktur data\modul 7 - queue>
```

Deskripsi dan Penjelasan Program:

Program di atas adalah implementasi antrian (queue) menggunakan linked list dalam bahasa C++. Struktur Mahasiswa didefinisikan untuk menyimpan informasi mahasiswa berupa nama dan nim. Struktur Node digunakan untuk menyimpan data tipe Mahasiswa dan pointer ke node berikutnya. Dua pointer global, front dan back, digunakan untuk melacak elemen pertama dan terakhir dalam antrian.

Fungsi isFull() selalu mengembalikan false karena antrian linked list tidak memiliki batas ukuran yang tetap. Fungsi isEmpty() memeriksa apakah antrian kosong dengan memeriksa apakah front bernilai nullptr. Fungsi enqueueAntrian() menambahkan elemen baru ke akhir antrian. Jika antrian kosong, node baru ini menjadi front dan back. Jika tidak, node baru ditambahkan di belakang dan pointer back diperbarui.

Fungsi dequeueAntrian() menghapus elemen dari depan antrian dan menghapus memori node tersebut. Jika antrian menjadi kosong setelah penghapusan, pointer back juga diatur menjadi nullptr. Fungsi countQueue() menghitung jumlah elemen dalam antrian dengan melakukan traversal dari front ke back. Fungsi clearQueue() menghapus semua elemen dalam antrian dengan memanggil dequeueAntrian() hingga antrian kosong. Fungsi viewQueue() menampilkan semua elemen dalam antrian, menunjukkan data nama dan nim masing-masing node dari depan hingga belakang.

BAB IV

KESIMPULAN

Queue adalah struktur data linear yang mengikuti prinsip FIFO (First In, First Out), dimana elemen yang pertama kali dimasukkan akan menjadi elemen pertama yang keluar. Dalam implementasi queue, dua operasi utama adalah enqueue (menambahkan elemen ke belakang antrian) dan dequeue (menghapus elemen dari depan antrian). Queue dapat diimplementasikan menggunakan berbagai cara, termasuk array statis, array dinamis, atau linked list.

1. **Array Statis**:
    - Kelebihan: Implementasi sederhana dan mudah dipahami.
    - Kekurangan: Ukuran tetap, sehingga kapasitas maksimum harus ditentukan terlebih dahulu, yang dapat menyebabkan inefisiensi memori atau keterbatasan ruang.
2. **Array Dinamis**:
    - Kelebihan: Dapat menyesuaikan ukuran saat dibutuhkan, lebih efisien dalam penggunaan memori dibandingkan array statis.
    - Kekurangan: Memerlukan overhead tambahan untuk mengelola ukuran dinamis, dan mungkin melibatkan operasi penyalinan data saat resizing.
3. **Linked List**:
    - Kelebihan: Ukuran tidak terbatas, memori dialokasikan secara dinamis sesuai kebutuhan, tidak memerlukan pergeseran elemen saat enqueue atau dequeue.
    - Kekurangan: Memerlukan lebih banyak memori per elemen karena penyimpanan pointer, sedikit lebih kompleks untuk diimplementasikan dibandingkan dengan array.

Queue memiliki banyak aplikasi dalam kehidupan nyata dan ilmu komputer, seperti:

- **Sistem Antrian**: Contoh nyata seperti antrian di bank, bioskop, atau layanan pelanggan.
- **Penjadwalan Tugas**: Sistem operasi menggunakan antrian untuk mengelola proses yang menunggu untuk dijalankan.
- **Transfer Data**: Buffering data yang dikirimkan melalui jaringan atau antara proses yang berbeda.
- **Algoritma dan Struktur Data Lainnya**: Digunakan dalam algoritma breadth-first search (BFS) pada graf, penjadwalan CPU, dan printer spooling.

Dengan memahami konsep dasar dan implementasi queue, programmer dapat memilih metode yang paling sesuai untuk kebutuhan aplikasi mereka dan memanfaatkan efisiensi yang ditawarkan oleh struktur data ini dalam berbagai konteks.