

LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA

MODUL 7

QUEUE



DISUSUN OLEH:

PRIESTY AMEILIANA MAULIDAH

2311102175

S1 IF-11-E

DOSEN:

Muhammad Afrizal Amrustian, S. Kom.

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO PURWOKERTO

2024

A.DASAR TEORI

Queue adalah struktur data yang digunakan untuk menyimpan data dengan metode FIFO (First-In First-Out).

Implementasi queue bisa menggunakan array atau linked list. Struktur data queue terdiri dari dua pointer yaitu front dan rear. Perbedaan dengan stack, pada queue, operasi penambahan dan penghapusan elemen dilakukan di tempat berbeda.

Pada queue, ada prosedur Enqueue untuk menambahkan elemen dan Dequeue untuk mengeluarkan elemen.

Operasi pada queue meliputi

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

B.Guided

Guided 1

```
// priesty ameiliana maulidah
// 2311102175
#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;
```

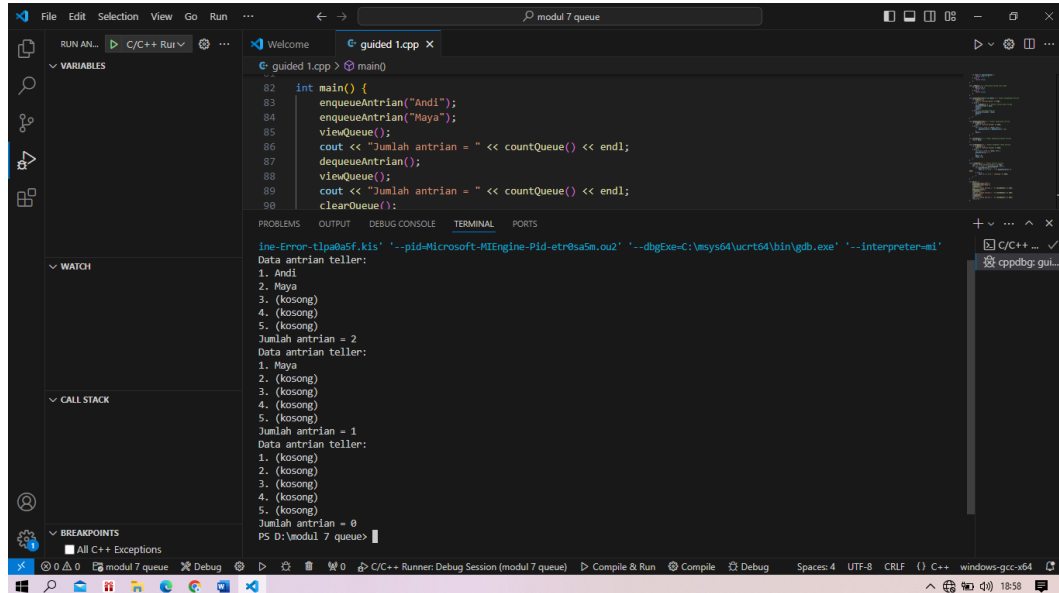
```
        }
```

```
    }
```

```
}
```

```
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;  
}
```

Screenshots output



The screenshot shows a C++ program running in a debugger. The program is a queue simulation. The output in the terminal window is as follows:

```
ine-Error-1tpa8a5f.kis' '--pid=Microsoft-MIEngine-Pid-etr8a5e.ou2' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
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:\modul 7 queue>
```

Deskripsi :

Fungsi-Fungsi:

- **main():** Fungsi utama program yang menjalankan simulasi antrian teller.
- **enqueueAntrian(string data):** Menambahkan nama nasabah (data) ke antrian.
- **dequeueAntrian():** Mengeluarkan nama nasabah terdepan dari antrian.
- **countQueue():** Menghitung jumlah nasabah dalam antrian.
- **isFull():** Mengecek apakah antrian penuh.
- **isEmpty():** Mengecek apakah antrian kosong.

- **clearQueue()**: Menghapus semua nasabah dari antrian.
- **viewQueue()**: Menampilkan daftar nama nasabah dalam antrian.

Variabel:

- **maksimalQueue**: Konstanta yang menentukan batas maksimum antrian (5 nasabah).
- **front**: Penunjuk indeks data terdepan dalam antrian (dimulai dari 0).
- **back**: Penunjuk indeks data terakhir dalam antrian (dimulai dari 1).
- **queueTeller**: Array string untuk menyimpan nama-nama nasabah dalam antrian (ukuran 5).

Alur Kerja:

1. Program dimulai dengan mendefinisikan fungsi-fungsi antrian dan variabel-variabelnya.
2. Dua nama nasabah ("Andi" dan "Maya") ditambahkan ke antrian.
3. Antrian ditampilkan beserta jumlah nasabahnya.
4. Satu nasabah dikeluarkan dari antrian.
5. Antrian ditampilkan kembali beserta jumlah nasabahnya.

6. Semua nasabah dihapus dari antrian.
7. Antrian ditampilkan (kosong) beserta jumlah nasabahnya (0).
8. Program diakhiri.

Penjelasan Fungsi:

- **enqueueAntrian(string data):** Fungsi ini menambahkan nama nasabah (data) ke dalam antrian. Pertama, fungsi mengecek apakah antrian penuh. Jika penuh, maka akan ditampilkan pesan "Antrian penuh". Jika tidak penuh, maka nama nasabah disimpan di array queueTeller pada indeks back. Kemudian, nilai back dan front diincrement.
- **dequeueAntrian():** Fungsi ini mengeluarkan nama nasabah terdepan dari antrian. Pertama, fungsi mengecek apakah antrian kosong. Jika kosong, maka akan ditampilkan pesan "Antrian kosong". Jika tidak kosong, maka data nama nasabah pada indeks front dihapus. Kemudian, semua data nama nasabah yang tersisa digeser ke depan satu indeks. Terakhir, nilai back didecrement.
- **countQueue():** Fungsi ini menghitung jumlah nasabah dalam antrian. Fungsi ini hanya

mengembalikan nilai back, yang menunjukkan jumlah data yang terisi dalam array queueTeller.

- **isFull()**: Fungsi ini mengecek apakah antrian penuh. Fungsi ini membandingkan nilai back dengan maksimalQueue. Jika back sama dengan maksimalQueue, maka antrian penuh dan fungsi mengembalikan nilai true. Sebaliknya, jika back masih kurang dari maksimalQueue, maka antrian tidak penuh dan fungsi mengembalikan nilai false.
- **isEmpty()**: Fungsi ini mengecek apakah antrian kosong. Fungsi ini membandingkan nilai back dengan 0. Jika back sama dengan 0, maka antrian kosong dan fungsi mengembalikan nilai true. Sebaliknya, jika back lebih besar dari 0, maka antrian tidak kosong dan fungsi mengembalikan nilai false.
- **clearQueue()**: Fungsi ini menghapus semua nasabah dari antrian. Pertama, fungsi mengecek apakah antrian kosong. Jika kosong, maka akan ditampilkan pesan "Antrian kosong". Jika tidak kosong, maka semua elemen dalam array queueTeller diubah menjadi string kosong (""). Kemudian, nilai back dan front direset menjadi 0.

- **viewQueue()**: Fungsi ini menampilkan daftar nama nasabah dalam antrian. Pertama, judul "Data antrian teller:" ditampilkan. Kemudian, for-loop digunakan untuk mengiterasi seluruh elemen dalam array queueTeller. Jika elemen tidak kosong, maka nama nasabah dan nomor antriannya ditampilkan. Jika elemen kosong, maka ditampilkan pesan "(kosong)".

c. unguided/tugas

unguided 1

```
// priesty ameiliana mualidah
// 2311102175
#include <iostream>

using namespace std;

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

Node* front = NULL; // Penunjuk ke elemen pertama
Node* rear = NULL; // Penunjuk ke elemen terakhir

bool isFull() {
    return rear != NULL; // Queue penuh jika rear tidak NULL
}

bool isEmpty() {
    return front == NULL; // Queue kosong jika front NULL
}
```

```
void enqueueAntrian(string data) {
```

```
    Node* newNode = new Node;
```

```
    newNode->data = data;
```

```
    newNode->next = NULL;
```

```
    if (isEmpty()) {
```

```
        front = newNode;
```

```
        rear = newNode;
```

```
    } else {
```

```
        rear->next = newNode;
```

```
        rear = newNode;
```

```
    }
```

```
}
```

```
void dequeueAntrian() {
```

```
    if (isEmpty()) {
```

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

```
        return;
```

```
    }
```

```
    Node* temp = front;
```

```
    front = front->next;
```

```
    delete temp;
```

```
    if (front == NULL) {
```

```
        rear = NULL;
```

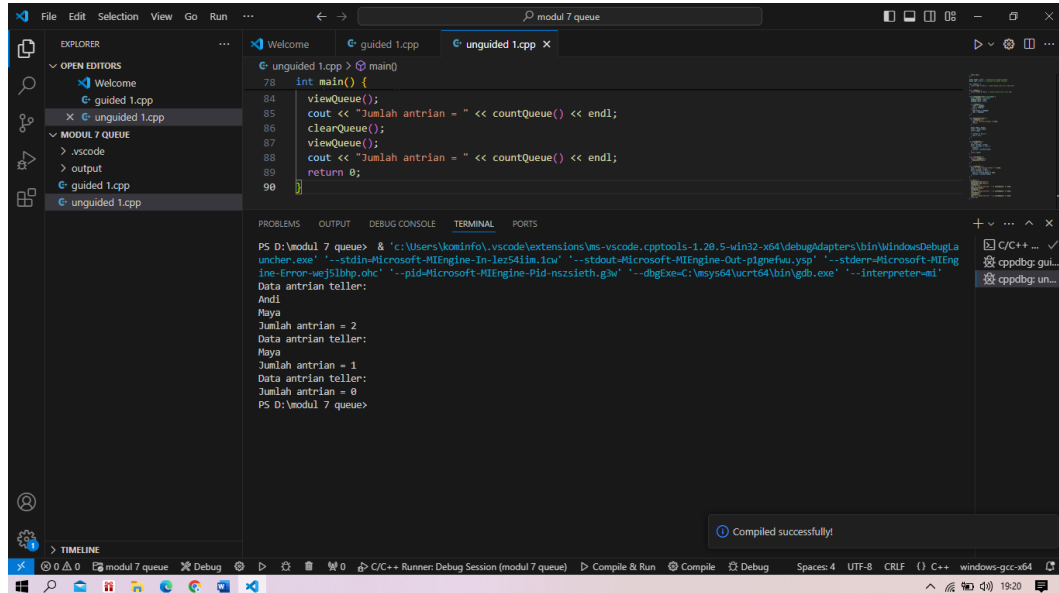
```
    }
```

```
}
```

```
int countQueue() {  
    int count = 0;  
    Node* current = front;  
    while (current != NULL) {  
        count++;  
        current = current->next;  
    }  
    return count;  
}  
  
void clearQueue() {  
    while (!isEmpty()) {  
        dequeueAntrian();  
    }  
}  
  
void viewQueue() {  
    cout << "Data antrian teller:" << endl;  
    Node* current = front;  
    while (current != NULL) {  
        cout << current->data << endl;  
        current = current->next;  
    }  
}
```

```
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;  
}
```

Screenshots output:



```
78 int main() {
79     viewQueue();
80     cout << "Jumlah antrian = " << countQueue() << endl;
81     clearQueue();
82     viewQueue();
83     cout << "Jumlah antrian = " << countQueue() << endl;
84     return 0;
85 }
```

PS D:\modul 7 queue> & "c:\Users\kominfo\.vscode\extensions\ms-vscode.cpptools-1.28.5-win32-x64\debugAdapters\bin\WindowsDebugLa
uncher.exe" --stdin=Microsoft-MIEngine-In-1ez541la.1cw' --stdout=Microsoft-MIEngine-Out-pigrefau.ysp' --stderr=Microsoft-MIEng
ine-Error-wejslhhp.ohc' --pid=Microsoft-MIEngine-Pid-nzslsth.gw' --dbgExe=c:\wsys64\ucrt64\bin\gdb.exe' --interpreter=mi
Data antrian teller:
Andi
Maya
Jumlah antrian = 2
Data antrian teller:
Maya
Jumlah antrian = 1
Data antrian teller:
Jumlah antrian = 0
PS D:\modul 7 queue>

Compiled successfully!

Deskripsi :

Struktur Data:

- **Node:** Digunakan untuk menyimpan data antrian (nama) dan pointer ke elemen berikutnya dalam linked list.
- **Queue:** Ditentukan oleh dua pointer, front dan rear.
 - front: Menunjuk ke elemen pertama dalam linked list (atau NULL jika queue kosong).
 - rear: Menunjuk ke elemen terakhir dalam linked list (atau NULL jika queue kosong).

Operasi Queue:

- **isFull():** Mengembalikan true jika queue penuh (artinya rear tidak NULL).
- **isEmpty():** Mengembalikan true jika queue kosong (artinya front adalah NULL).
- **enqueueAntrian(string data):** Menambahkan elemen baru (nama) ke bagian belakang queue.
- **dequeueAntrian():** Menghapus elemen pertama dari queue.
- **countQueue():** Menghitung jumlah elemen dalam queue.
- **clearQueue():** Menghapus semua elemen dari queue.
- **viewQueue():** Menampilkan semua elemen (nama) dalam queue.

Main Function:

- Menambahkan dua nama ("Andi" dan "Maya") ke dalam queue.
- Menampilkan isi queue dan menghitung jumlah elemen.
- Menghapus elemen pertama dari queue.
- Menampilkan isi queue dan menghitung jumlah elemen.
- Menghapus semua elemen dari queue.

- Menampilkan isi queue dan menghitung jumlah elemen (seharusnya kosong).

Keuntungan Menggunakan Linked List:

- **Memori Dinamis:** Linked list hanya mengalokasikan memori untuk elemen yang benar-benar dibutuhkan, sehingga lebih hemat memori dibandingkan array yang harus mengalokasikan memori untuk semua elemen meskipun tidak terisi.
- **Kinerja Operasi:** Operasi enqueue dan dequeue pada linked list memiliki kinerja konstan $O(1)$, tidak terpengaruh oleh jumlah elemen dalam queue.

Unguided 2

```
// priesty ameiliana maulidah
// 2311102175

#include <iostream>

using namespace std;

struct Mahasiswa {
    string Nama;
    string NIM;
};

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

Node* front = NULL; // Pointer to the first element
Node* rear = NULL; // Pointer to the last element

bool isEmpty() {
    return front == NULL; // Queue is empty if front is NULL
}
```

```
void enqueueAntrian(Mahasiswa data) {
```

```
    Node* newNode = new Node;
```

```
    newNode->data = data;
```

```
    newNode->next = NULL;
```

```
    if (isEmpty()) {
```

```
        front = newNode;
```

```
        rear = newNode;
```

```
    } else {
```

```
        rear->next = newNode;
```

```
        rear = newNode;
```

```
    }
```

```
}
```

```
void dequeueAntrian() {
```

```
    if (isEmpty()) {
```

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

```
        return;
```

```
    }
```

```
    Node* temp = front;
```

```
    front = front->next;
```

```
    delete temp;
```

```
    if (front == NULL) {
```

```
        rear = NULL;
```

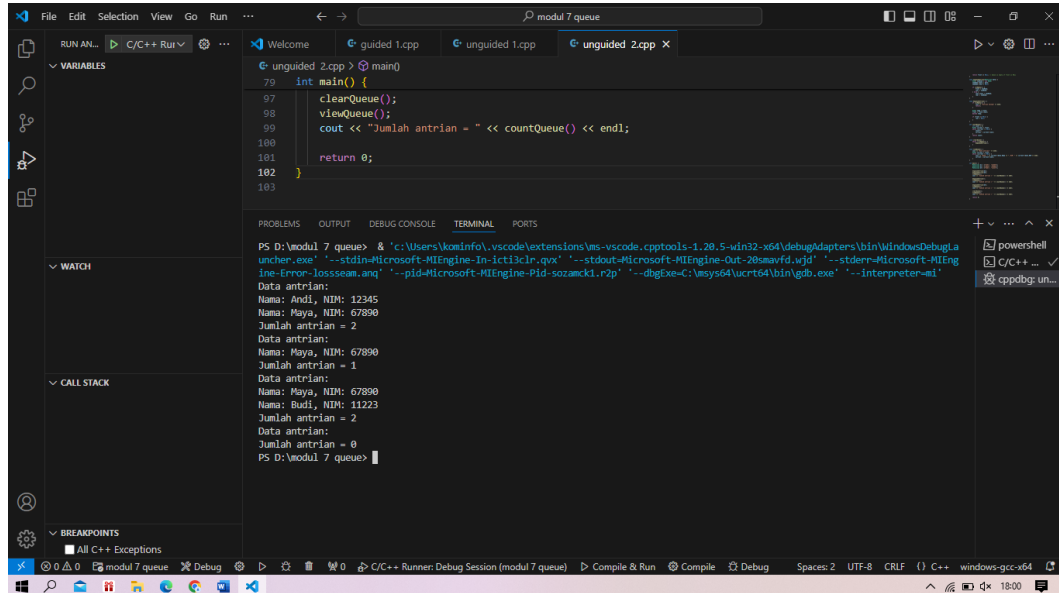
```
    }
```

```
}
```

```
int countQueue() {  
    int count = 0;  
    Node* current = front;  
    while (current != NULL) {  
        count++;  
        current = current->next;  
    }  
    return count;  
}  
  
void clearQueue() {  
    while (!isEmpty()) {  
        dequeueAntrian();  
    }  
}  
  
void viewQueue() {  
    cout << "Data antrian:" << endl;  
    Node* current = front;  
    while (current != NULL) {  
        cout << "Nama: " << current->data>Nama << ", NIM: " << current->data.NIM <<  
endl;  
        current = current->next;  
    }  
}
```

```
int main() {  
    Mahasiswa m1 = {"Andi", "12345"};  
    Mahasiswa m2 = {"Maya", "67890"};  
    Mahasiswa m3 = {"Budi", "11223"};  
  
    enqueueAntrian(m1);  
    enqueueAntrian(m2);  
    viewQueue();  
    cout << "Jumlah antrian = " << countQueue() << endl;  
  
    dequeueAntrian();  
    viewQueue();  
    cout << "Jumlah antrian = " << countQueue() << endl;  
  
    enqueueAntrian(m3);  
    viewQueue();  
    cout << "Jumlah antrian = " << countQueue() << endl;  
  
    clearQueue();  
    viewQueue();  
    cout << "Jumlah antrian = " << countQueue() << endl;  
  
    return 0;  
}
```

Screenshots output:



The screenshot shows the Visual Studio Code interface with a C++ file named 'unguided 2.cpp' open. The code defines a queue structure and a main function. The output window shows the execution results, including the number of elements in the queue and the names of the students.

```
79 int main() {
80     clearQueue();
81     viewQueue();
82     cout << "Jumlah antrian = " << countQueue() << endl;
83     return 0;
84 }
```

Output:

```
PS D:\modul 7 queue> & "c:\Users\kominfo\.vscode\extensions\ms-vscode.cpptools-1.28.5-win32-x64\debugAdapters\bin\WindowsDebugLa
uncher.exe" --stdin-Microsoft-MIEngine-In-1ct13clp.qnx' '--stdout-Microsoft-MIEngine-Out-26sawfd.wjd' '--stderr-Microsoft-MIEng
ine-Error-Josseam.ang' '--pid-Microsoft-MIEngine-Pid-sozawckl.r2p' '--dbgExe-C:\wsys64\ucrnt64\bin\gdb.exe' '--interpreter-mi
Data antrian:
Nama: Andi, NIM: 12345
Nama: Maya, NIM: 67890
Jumlah antrian = 2
Data antrian:
Nama: Maya, NIM: 67890
Jumlah antrian = 1
Data antrian:
Nama: Maya, NIM: 67890
Nama: Budi, NIM: 11223
Jumlah antrian = 2
Data antrian:
Jumlah antrian = 0
PS D:\modul 7 queue>
```

Deskripsi :

1. Pemanggilan Header:

- `#include <iostream>`: Memasukkan pustaka input/output standar untuk interaksi konsol.

2. Ruang Nama:

- `using namespace std;` : Menghindari penulisan `std::` berulang kali sebelum elemen pustaka standar.

3. Struktur Mahasiswa (Mahasiswa)

- `struct Mahasiswa {`: Mendefinisikan struktur bernama Mahasiswa untuk menyimpan informasi mahasiswa.

- `string Nama;`: Mendeklarasikan variabel anggota Nama bertipe string untuk menyimpan nama mahasiswa.
- `string NIM;`: Mendeklarasikan variabel anggota NIM bertipe string untuk menyimpan ID mahasiswa.

4. Struktur Node Antrian (Node)

- `struct Node {`: Mendefinisikan struktur bernama Node untuk merepresentasikan node dalam antrian.
 - `Mahasiswa data;`: Menyimpan instansi struktur Mahasiswa untuk menyimpan data mahasiswa di dalam node.
 - `Node* next;`: Pointer ke node berikutnya dalam antrian (struktur daftar tertaut).

5. Variabel Global:

- `Node* front = NULL;`: Pointer ke elemen pertama (depan) antrian, awalnya NULL (kosong).
- `Node* rear = NULL;`: Pointer ke elemen terakhir (belakang) antrian, awalnya NULL (kosong).

6. Fungsi isEmpty():

- `bool isEmpty() {`: Memeriksa apakah antrian kosong.

- `return front == NULL;` Mengembalikan true jika front adalah NULL (antrian kosong), false sebaliknya.

7. Fungsi `enqueueAntrian()`:

- `void enqueueAntrian(Mahasiswa data) {`
Menambahkan mahasiswa baru ke bagian belakang (belakang) antrian.
 - `Node* newNode = new Node;` Membuat node baru menggunakan alokasi memori dinamis.
 - `newNode->data = data;` Menetapkan data Mahasiswa yang disediakan ke node baru.
 - `newNode->next = NULL;` Mengatur pointer next dari node baru ke NULL (node terakhir).
 - **Menangani Antrian Kosong:**
 - `if (isEmpty()) {` Jika antrian kosong (front adalah NULL).
 - `front = newNode;` Mengatur front dan rear ke node baru.
 - **Menangani Antrian yang Ada:**
 - `else {` Jika antrian tidak kosong.

- rear->next = newNode;;
Menghubungkan node rear saat ini ke node baru.
- rear = newNode;; Memperbarui rear untuk menunjuk ke node terakhir yang baru ditambahkan.

8. Fungsi dequeueAntrian():

- void dequeueAntrian() {: Menghapus mahasiswa di depan (awal) antrian.
 - **Pemeriksaan Antrian Kosong:**
 - if (isEmpty()) {: Jika antrian kosong.
 - cout << "Antrian kosong" << endl;;
Mencetak pesan error.
 - return;; Keluar dari fungsi tanpa dequeue.
 - **Dequeuing:**
 - Node* temp = front;; Membuat pointer sementara untuk menyimpan node yang akan dihapus.
 - front = front->next;; Memindahkan front ke node berikutnya (front baru).

- delete temp;; Membebaskan memori yang sebelumnya ditempati oleh node yang dihapus (temp).
- **Menangani Antrian Satu Elemen:**
 - if (front == NULL) {: Jika front menjadi NULL setelah dequeue (satu-satunya elemen tersisa).
 - rear = NULL;; Mengatur rear ke NULL juga (antrian kosong).

9. Fungsi countQueue():

- int countQueue() {: Menghitung jumlah mahasiswa dalam antrian.
 - int count = 0;; Menginisialisasi variabel count ke 0.
 - Node* current = front;; Membuat pointer current untuk melintasi antrian.
 - while (current != NULL) {: Iterasi selama current tidak NULL (akhir antrian).
 - count

E. Referensi

<https://jurnal.polinema.ac.id/index.php/jip/article/view/2633>