LAPORAN PRAKTIKUM

MODUL VI
QUEUE

Disusun oleh:
Maulana Ghani Rolanda
NIM: 2311102012

Dosen Pengampu:
Muhammad Afrizal Amrustian, S. Kom., M. Kom

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2023

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

Queue (antrian) adalah struktur data linier yang mengikuti prinsip FIFO (First In, First Out). Artinya, elemen yang pertama kali dimasukkan ke dalam antrian akan menjadi elemen pertama yang dikeluarkan.

1. **Struktur Dasar**:
    - Queue terdiri dari dua ujung, yaitu depan (front) dan belakang (rear).
    - Elemen baru ditambahkan di ujung belakang (enqueue), dan elemen diambil dari ujung depan (dequeue).
2. **Operasi Dasar**:
    - **Enqueue**: Menambahkan elemen ke belakang antrian.
    - **Dequeue**: Menghapus elemen dari depan antrian.
    - **Peek/Front**: Melihat elemen di depan tanpa menghapusnya.
    - **IsEmpty**: Memeriksa apakah antrian kosong.
    - **IsFull**: Memeriksa apakah antrian penuh (khusus untuk implementasi dengan batasan kapasitas).
3. **Implementasi Queue**:

- **Array**: Queue bisa diimplementasikan menggunakan array dengan menggunakan dua indeks untuk melacak front dan rear.
- **Linked List**: Queue juga bisa diimplementasikan menggunakan linked list, di mana setiap node menyimpan elemen dan pointer ke node berikutnya.

4. **Jenis-Jenis Queue**:
    - **Simple Queue**: Mengikuti aturan FIFO dasar.
    - **Circular Queue**: Memperbaiki masalah array fixed size dengan menghubungkan ujung belakang ke ujung depan, membentuk lingkaran.
    - **Priority Queue**: Elemen ditambahkan dengan prioritas, elemen dengan prioritas lebih tinggi dikeluarkan lebih dulu.
    - **Deque (Double-Ended Queue)**: Elemen bisa ditambahkan atau dihapus dari kedua ujung (depan dan belakang)

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 C:\olan\vscode\Modul7> cd "c:\olan\vscode\Modul7\" ; if ($?) {
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 C:\olan\vscode\Modul7>
```

Deskripsi program

Kode tersebut adalah implementasi dasar dari struktur data queue menggunakan array. Pada bagian awal, terdapat deklarasi dan inisialisasi variabel. `maksimalQueue` ditetapkan sebagai kapasitas maksimal antrian yang berjumlah 5 elemen. Variabel `front` dan `back` digunakan untuk melacak indeks depan dan belakang antrian, masing-masing diinisialisasi dengan nilai 0. Array `queueTeller` berukuran 5 digunakan untuk menyimpan elemen-elemen antrian. Fungsi-fungsi yang diimplementasikan termasuk `isFull` untuk memeriksa apakah antrian

penuh, `isEmpty` untuk memeriksa apakah antrian kosong, `enqueueAntrian` untuk menambahkan elemen ke dalam antrian, dan `dequeueAntrian` untuk menghapus elemen dari antrian. Fungsi `countQueue` mengembalikan jumlah elemen dalam antrian, `clearQueue` menghapus semua elemen dalam antrian, dan `viewQueue` menampilkan semua elemen dalam antrian. Fungsi `main` digunakan untuk menguji fungsi-fungsi tersebut dengan menambahkan, menghapus, dan menampilkan elemen-elemen dalam antrian, serta mengosongkan antrian. Implementasi ini memberikan contoh dasar bagaimana menggunakan queue dengan operasi enqueue, dequeue, dan beberapa fungsi tambahan untuk pengelolaan antrian.

LATIHAN KELAS - UNGUIDED

1. **Unguided 1**

   Source code

```cpp
#include <iostream>
#include <string>

using namespace std;

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

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

```
bool isFull() {

    return false;
}

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

void enqueueAntrian(string data) {
    Node* newNode = new Node{data, nullptr};
    if (isEmpty()) {
        front = back = newNode;
    } else {
        back->next = newNode;
        back = newNode;
    }
}

void dequeueAntrian() {
    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* temp = front;
    while (temp != nullptr) {
        count++;
        temp = temp->next;
    }
    return count;
}
```

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

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("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 C:\olan\vscode\Modul7> cd "c:\olan\vscode\Modul7\" ; if ($?)
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 C:\olan\vscode\Modul7>
```

Deskripsi Program

Untuk mengubah implementasi antrian dari pendekatan berbasis array menjadi berbasis linked list, beberapa perubahan dilakukan pada struktur data dan operasi antrian. Pertama, sebuah struktur `Node` diperkenalkan untuk mewakili setiap elemen dalam antrian, di mana setiap `Node` berisi `data` (untuk menyimpan elemen antrian) dan pointer `next` (untuk menunjuk ke node berikutnya dalam antrian). Alih-alih menggunakan array dan indeks integer (`front` dan `back`), sekarang digunakan dua pointer: `front` dan `back`. `front` menunjuk ke node pertama dalam antrian, dan `back` menunjuk ke node terakhir dalam antrian. Dalam fungsi `isFull()`, karena antrian berbasis linked list tidak pernah penuh kecuali sistem kehabisan memori, fungsi ini selalu mengembalikan `false`. Fungsi `isEmpty()` memeriksa apakah `front` adalah `nullptr` untuk menentukan apakah antrian kosong. Fungsi `enqueueAntrian()` menambahkan node baru ke ujung antrian. Jika antrian kosong, `front` dan `back` keduanya menunjuk ke node baru, sedangkan jika tidak kosong, node baru ditambahkan setelah `back`, dan `back` diperbarui untuk menunjuk ke node baru. Fungsi `dequeueAntrian()` menghapus node dari depan antrian dan jika antrian menjadi kosong setelah penghapusan, `back` juga diatur ke

`nullptr`. Fungsi `countQueue()` menghitung jumlah node dalam antrian dengan mengiterasi melalui linked list, sementara `clearQueue()` menghapus semua node dalam antrian dengan mengiterasi dan memanggil `dequeueAntrian()` sampai antrian kosong. Fungsi `viewQueue()` menampilkan isi antrian dengan mengiterasi melalui linked list dan mencetak data dari setiap node. Dalam program utama, fungsi-fungsi ini digunakan untuk menambahkan, menghapus, dan menampilkan elemen-elemen dalam antrian serta mengosongkan antrian. Dengan perubahan ini, implementasi antrian menggunakan linked list memungkinkan manajemen antrian yang lebih fleksibel tanpa batasan ukuran tetap seperti pada implementasi berbasis array.

2. Unguided 2

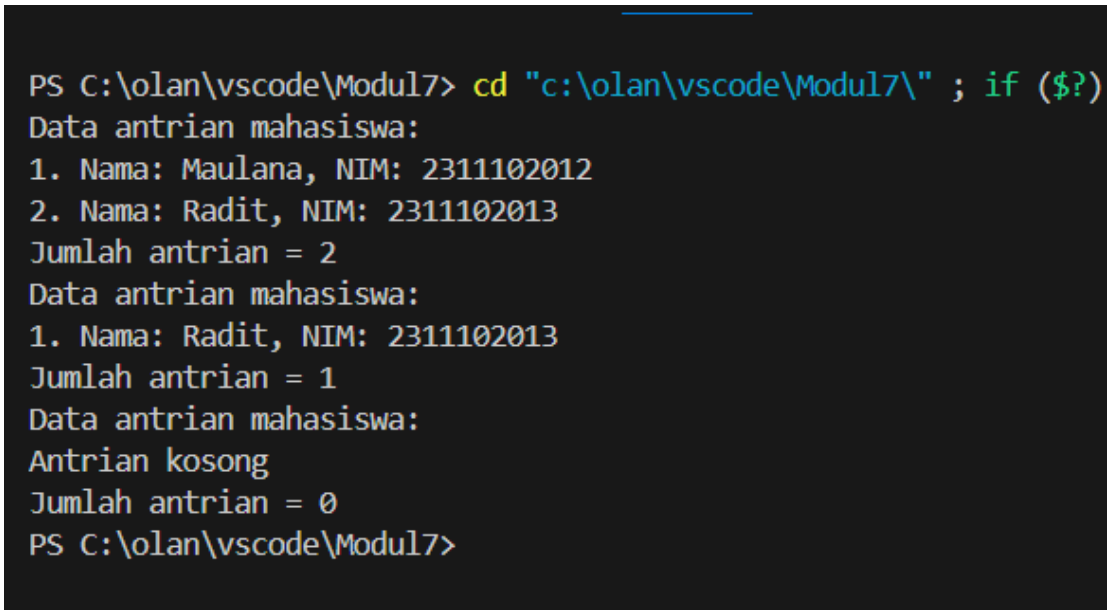
Source code

```cpp
#include <iostream>
#include <string>

using namespace std;

struct Node {
    string nama;
    string nim;
    Node* next;
};

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

bool isFull() {
    // Linked list tidak pernah penuh kecuali memori habis
    return false;
}

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

```
void enqueueAntrian(string nama, string nim) {
    Node* newNode = new Node{nama, nim, nullptr};
    if (isEmpty()) {
        front = back = newNode;
    } else {
        back->next = newNode;
        back = newNode;
    }
}

void dequeueAntrian() {
    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* temp = front;
    while (temp != nullptr) {
        count++;
        temp = temp->next;
    }
    return count;
}

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

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("Maulana", "2311102012");
    enqueueAntrian("Radit", "2311102013");
    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

Implementasi antrian menggunakan linked list untuk menyimpan data mahasiswa memiliki struktur data yang terdiri dari sebuah `Node`. Setiap `Node` dalam struktur ini memiliki dua atribut: `nama` dan `nim` mahasiswa, serta pointer `next` yang menunjuk ke node berikutnya dalam antrian. Terdapat dua pointer global, `front` dan `back`, yang menandai elemen pertama dan terakhir dalam antrian. Fungsi `isFull()` selalu mengembalikan `false`, mengingat linked list tidak memiliki batasan ukuran tetap. Fungsi `isEmpty()` memeriksa apakah antrian kosong dengan memeriksa apakah `front` adalah `nullptr`. Fungsi `enqueueAntrian()` menambahkan node baru ke ujung antrian, baik dengan mengatur `front` dan `back` jika antrian kosong, atau hanya mengatur `back` jika sudah ada elemen dalam antrian. Fungsi `dequeueAntrian()` menghapus node pertama dalam antrian, dengan memperbarui `front` ke node berikutnya jika perlu dan mengatur `back` ke `nullptr` jika antrian menjadi kosong setelah penghapusan. Fungsi `countQueue()` menghitung jumlah node dalam antrian dengan mengiterasi dari `front` ke `back`. Fungsi `clearQueue()` menghapus semua node dalam antrian dengan memanggil `dequeueAntrian()` berulang kali hingga antrian kosong. Fungsi `viewQueue()` menampilkan semua elemen dalam antrian dengan mencetak `nama` dan `nim` dari setiap node. Program utama menguji fungsi-fungsi tersebut dengan menambahkan beberapa mahasiswa ke

antrian, menghapus satu elemen dari antrian, mengosongkan antrian, dan menampilkan hasil akhir dari antrian.

BAB IV

KESIMPULAN

Laporan praktikum ini memberikan pengenalan terhadap struktur data queue dan implementasinya melalui dua pendekatan utama: menggunakan array dan linked list. Konsep dasar queue yang mengikuti prinsip FIFO (First In, First Out) dijelaskan, di mana elemen pertama yang dimasukkan akan menjadi elemen pertama yang dikeluarkan. Dalam implementasi dengan array, terdapat batasan ukuran tetap dan penyesuaian indeks saat operasi enqueue dan dequeue. Sementara itu, penggunaan linked list memungkinkan manajemen antrian yang lebih fleksibel tanpa batasan ukuran tetap. Praktikum ini juga mencakup operasi dasar pada queue, seperti enqueue (menambahkan elemen), dequeue (menghapus elemen), dan view (menampilkan elemen). Dengan studi kasus pada pengelolaan data mahasiswa, penggunaan queue dengan linked list memungkinkan penyimpanan informasi nama dan NIM mahasiswa dalam setiap node. Diharapkan bahwa melalui praktikum ini, mahasiswa dapat memahami konsep dasar struktur data queue, mampu menerapkan operasi dasar, dan mengenali penerapan serta implikasi dari penggunaan queue dalam pemrograman dan pengembangan aplikasi.

DAFTAR PUSTAKA

Geeks For Geeks 11 Mei, 2024. Queue Data Structure, Diakses pada 27 Mei 2024 dari https://www.geeksforgeeks.org/queue-data-structure/