LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 8
QUEUE

Disusun Oleh :

NAMA : LAHRA BUDI SAPUTRA

NIM : 103112430054

Dosen :

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Queue (antrean) merupakan struktur data linear yang mengadopsi prinsip *First In First Out* (FIFO), yang berarti elemen yang pertama kali masuk ke dalam antrean akan menjadi elemen pertama yang diproses atau dikeluarkan. Struktur ini memiliki dua operasi utama, yaitu penyisipan (*Enqueue/Insert*) yang selalu dilakukan pada posisi belakang (*Tail*) dan pengambilan (*Dequeue/Delete*) yang dilakukan pada posisi depan (*Head*). Implementasi Queue dapat dilakukan menggunakan *Linked List* atau representasi Tabel (*Array*). Dalam representasi tabel, memori bersifat statis dengan jumlah elemen terbatas, dan pengelolaannya dapat dibagi menjadi beberapa mekanisme. Mekanisme tersebut meliputi: **Alternatif 1**, di mana *Head* selalu tetap di posisi awal dan elemen digeser (*shifting*) setiap kali terjadi penghapusan; **Alternatif 2**, di mana *Head* dan *Tail* bergerak maju untuk menghindari pergeseran terus-menerus, namun memerlukan penanganan khusus saat terjadi kondisi "penuh semu" (*pseudo-full*) dengan menggeser elemen hanya ketika *Tail* mencapai batas maksimum ; serta **Alternatif 3**, yang menggunakan konsep *circular buffer* (berputar) di mana indeks *Head* dan *Tail* akan kembali ke posisi awal setelah mencapai batas maksimum tabel untuk efisiensi memori.

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

“queue.h”

```
#ifndef QUEUE_H
#define QUEUE_H
#define MAX_QUEUE 5

struct Queue {
    int info [MAX_QUEUE];
    int head;
    int tail;
    int count;
};

void createQueue(Queue &Q);
bool isEmpty(Queue Q);
```

```
bool isFull(Queue Q);
void enqueue(Queue &Q, int data);
int dequeue(Queue &Q);
void printInfo(Queue Q);

#endif
```

“queue.cpp”

```
 #include "queue.h"
#include <iostream>
using namespace std;

void createQueue(Queue &Q){
   Q.head = Q.tail = Q.count = 0;
}

bool isEmpty(Queue Q){
   return Q.count == 0;
}

bool isFull(Queue Q){
   return Q.count == MAX_QUEUE;
}

void enqueue(Queue &Q, int x){
   if(!isFull(Q)){
      Q.info[Q.tail] = x;
      Q.tail = (Q.tail + 1) % MAX_QUEUE;
      Q.count++;
   } else {
      cout << "Queue penuh!" << endl;
   }
}
```

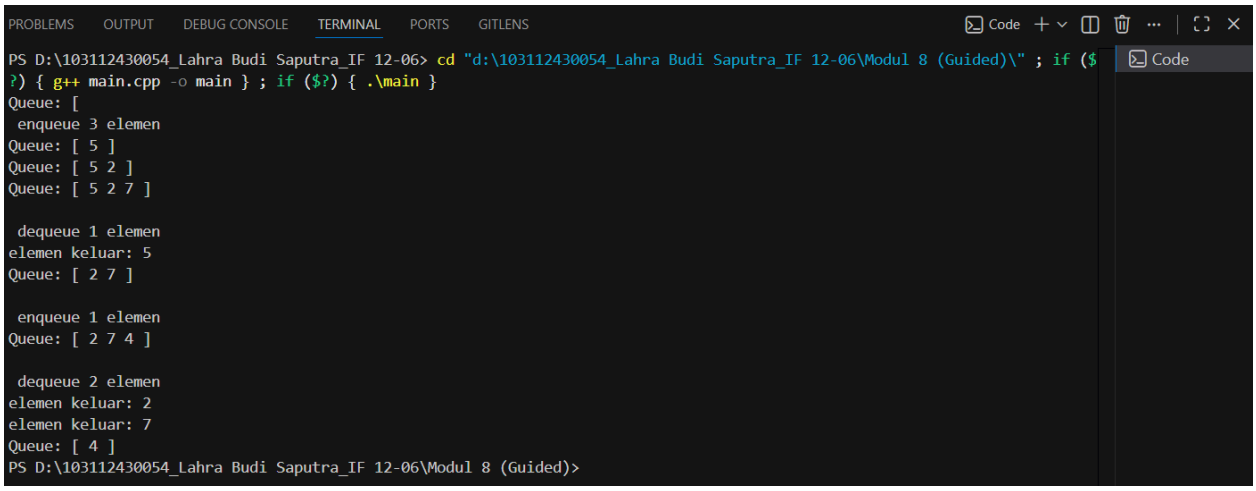
```
int dequeue(Queue &Q){
    if(!isEmpty(Q)){
        int x = Q.info[Q.head];
        Q.head = (Q.head + 1) % MAX_QUEUE;
        Q.count--;
        return x;
    } else {
        cout << "Queue kosong!" << endl;
        return -1;
    }
}

void printInfo(Queue Q){
    cout << "Queue: [ ";
    if(!isEmpty(Q)){
        int i = Q.head;
        int n = 0;
        while(n < Q.count){
            cout << Q.info[i] << " ";
            i = (i + 1) % MAX_QUEUE;
            n++;
        }
    cout << "]" << endl;
    }
}
```

“main.cpp”

```
#include "queue.h"
#include "queue.cpp"
#include <iostream>

using namespace std;
```

```
int main() {
    Queue Q;
    createQueue(Q);
    printInfo(Q);

    cout << "\n enqueue 3 elemen" << endl;
    enqueue(Q, 5);
    printInfo(Q);

    enqueue(Q, 2);
    printInfo(Q);

    enqueue(Q, 7);
    printInfo(Q);

    cout << "\n dequeue 1 elemen" << endl;
    cout << "elemen keluar: " << dequeue(Q) << endl;
    printInfo(Q);

    cout << "\n enqueue 1 elemen" << endl;
    enqueue(Q, 4);
    printInfo(Q);

    cout << "\n dequeue 2 elemen" << endl;
    cout << "elemen keluar: " << dequeue(Q) << endl;
    cout << "elemen keluar: " << dequeue(Q) << endl;
    printInfo(Q);

    return 0;
}
```

Screenshots Output



Deskripsi:

Program ini terdiri dari tiga berkas. ***File queue.h*** berisi deklarasi struktur queue dan fungsi-fungsi dasar seperti membuat antrian, mengecek kosong/penuh, menambah, menghapus, dan menampilkan elemen. ***File queue.cpp*** mengimplementasikan seluruh fungsi tersebut menggunakan konsep *circular array* untuk mengatur pergerakan head dan tail. Sementara itu, ***main.cpp*** berfungsi sebagai program utama yang menguji operasi queue dengan melakukan proses enqueue, dequeue, dan menampilkan isi antrian agar terlihat hasil dari setiap perubahan.

C. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Unguided (SOAL 1)

(“queue.h”)

```
// Nama : Lahra Budi Saputra
// NIM  : 103112430054
// Kelas: IF 12-06 Teknik Informatika A'24
#ifndef QUEUE_H
#define QUEUE_H
#include <iostream>
```

```
using namespace std;

#define MAX 5

typedef int infotype;

struct Queue {

    infotype info[MAX];

    int head;

    int tail;

};


void createQueue(Queue &Q);

bool isEmptyQueue(Queue Q);

bool isFullQueue(Queue Q);

void enqueue(Queue &Q, infotype x);

infotype dequeue(Queue &Q);

void printInfo(Queue Q);

#endif
```

(“queue.cpp”)

```
  // Nama : Lahra Budi Saputra
// NIM  : 103112430054
// Kelas: IF 12-06 Teknik Informatika A'24
#include "queue.h"
#include <iostream>

using namespace std;

void createQueue(Queue &Q) {
    Q.head = -1;
    Q.tail = -1;
}

bool isEmptyQueue(Queue Q) {
    return Q.head == -1;
```

```
}

bool isFullQueue(Queue Q) {
   return Q.tail == MAX - 1;
}

void enqueue(Queue &Q, infotype x) {
   if (isFullQueue(Q)) {
      cout << "Antrean Penuh" << endl;
   } else {
      if (isEmptyQueue(Q)) {
         Q.head = 0;
         Q.tail = 0;
         Q.info[0] = x;
      } else {
         Q.tail++;
         Q.info[Q.tail] = x;
      }
   }
}

infotype dequeue(Queue &Q) {
   infotype val = -1;
   if (isEmptyQueue(Q)) {
      cout << "Antrean Kosong" << endl;
   } else {
      val = Q.info[Q.head];
      if (Q.head == Q.tail) {
         createQueue(Q);
      } else {
         for (int i = Q.head; i < Q.tail; i++) {
            Q.info[i] = Q.info[i + 1];
         }
         Q.tail--;
      }
   }
   return val;
}

void printInfo(Queue Q) {

   if (Q.head >= 0) {
      cout << " ";
```

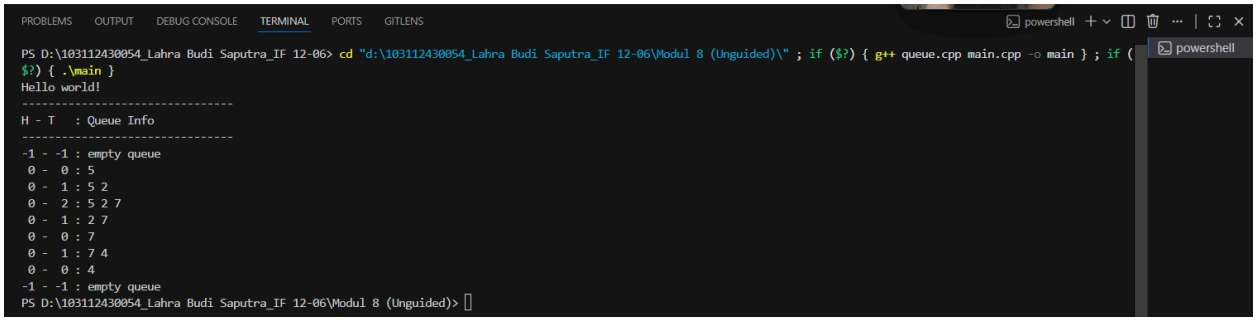
```
    }
    cout << Q.head << " - ";

    if (Q.tail >= 0) {
        cout << " ";
    }
    cout << Q.tail << " : ";

    if (isEmptyQueue(Q)) {
        cout << "empty queue" << endl;
    } else {
        for (int i = Q.head; i <= Q.tail; i++) {
            cout << Q.info[i] << " ";
        }
        cout << endl;
    }
}
```

(main.cpp)

```
// Nama : Lahra Budi Saputra

// NIM  : 103112430054

// Kelas: IF 12-06 Teknik Informatika A'24

#include "queue.h"

using namespace std;

int main() {

    cout << "Hello world!" << endl;

    cout << "--------------------------------" << endl;

    cout << "H - T \t: Queue Info" << endl;

    cout << "--------------------------------" << endl;

    Queue Q;

    createQueue(Q);

    printInfo(Q);
```

```
    enqueue(Q, 5);
    printInfo(Q);
    enqueue(Q, 2);
    printInfo(Q);
    enqueue(Q, 7);
    printInfo(Q);
    dequeue(Q);
    printInfo(Q);
    dequeue(Q);
    printInfo(Q);
    enqueue(Q, 4);
    printInfo(Q);
    dequeue(Q);
    printInfo(Q);
    dequeue(Q);
    printInfo(Q);
    return 0;
}
```

Screenshots Output:



Deskripsi:

File-file kode tersebut merupakan implementasi Abstract Data Type (ADT) Queue menggunakan representasi array statis (tabel) dengan kapasitas maksimal 5 elemen. ***File queue.h*** *berfungsi sebagai header yang mendeklarasikan struktur data Queue (terdiri dari array info, head, dan tail) beserta prototipe fungsi-fungsi primitifnya. Implementasi logika terdapat pada* ***file queue.cpp,*** *yang menerapkan mekanisme Alternatif 1 (Head diam, Tail bergerak), di mana operasi dequeue dilakukan dengan menggeser seluruh elemen yang tersisa ke posisi depan (shifting) agar indeks head selalu terjaga di posisi awal.* ***Terakhir, file main.cpp*** *bertindak sebagai program utama (driver) untuk menguji kebenaran logika ADT tersebut melalui serangkaian operasi penyisipan (enqueue) dan penghapusan (dequeue) data, sembari menampilkan perubahan posisi Head, Tail, dan isi antrean di setiap langkahnya.*

Unguided (SOAL 2)

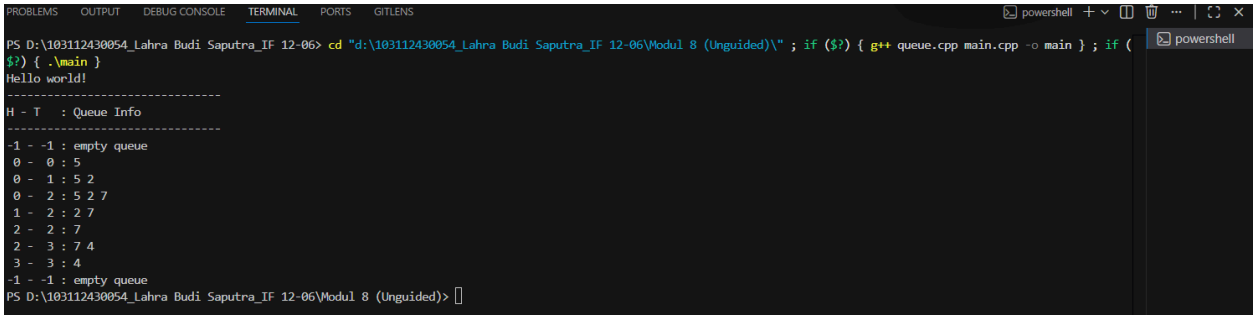
(queue.cpp)

```
 //* SOAL NOMOR 2*//
// Queue Alternatif 2 (head bergerak, tail bergerak).
#include "queue.h"
#include <iostream>

using namespace std;
void createQueue(Queue &Q) {
    Q.head = -1;
    Q.tail = -1;
}

bool isEmptyQueue(Queue Q) {
    return Q.head == -1;
}

bool isFullQueue(Queue Q) {
    return (Q.head == 0 && Q.tail == MAX - 1);
}
```

```
void enqueue(Queue &Q, infotype x) {
    if (isFullQueue(Q)) {
        cout << "Antrean Penuh" << endl;
    } else {
        if (isEmptyQueue(Q)) {
            Q.head = 0;
            Q.tail = 0;
            Q.info[0] = x;
        } else {
            if (Q.tail == MAX - 1) {
                for (int i = Q.head; i <= Q.tail; i++) {
                    Q.info[i - Q.head] = Q.info[i];
                }
                // Sesuaikan posisi Tail dan Head setelah digeser
                Q.tail = Q.tail - Q.head;
                Q.head = 0;

                // Setelah geser, barulah tambahkan elemen baru
                Q.tail++;
                Q.info[Q.tail] = x;
            } else {
                // Jika belum mentok, Tail cukup maju biasa
                Q.tail++;
                Q.info[Q.tail] = x;
            }
        }
    }
}

infotype dequeue(Queue &Q) {
    infotype val = -1;
    if (isEmptyQueue(Q)) {
        cout << "Antrean Kosong" << endl;
    } else {
        val = Q.info[Q.head];
        if (Q.head == Q.tail) {
            createQueue(Q);
        } else {
            Q.head++;
        }
    }
    return val;
}
```

```
void printInfo(Queue Q) {
    if (Q.head != -1 && Q.head < 10) cout << " ";
    cout << Q.head << " - ";
    if (Q.tail != -1 && Q.tail < 10) cout << " ";
    cout << Q.tail << " : ";

    if (isEmptyQueue(Q)) {
        cout << "empty queue" << endl;
    } else {
        for (int i = Q.head; i <= Q.tail; i++) {
            cout << Q.info[i] << " ";
        }
        cout << endl;
    }
}
```

Screenshots Output:



Deskripsi:

Kode ini mengimplementasikan struktur data Queue menggunakan array statis dengan mekanisme Head dan Tail yang bergerak maju untuk meningkatkan efisiensi waktu eksekusi. Dalam implementasi ini, operasi dequeue dilakukan hanya dengan menaikkan indeks *Head* tanpa menggeser elemen lain, sehingga proses penghapusan berjalan sangat cepat. Untuk mengatasi masalah "penuh semu" (di mana *Tail* mencapai batas akhir array tetapi masih ada ruang kosong di depan), operasi enqueue dilengkapi logika cerdas yang hanya melakukan pergeseran elemen (*shifting*) ke posisi awal secara massal saat kondisi tersebut terpenuhi. Dengan demikian, algoritma ini menyeimbangkan efisiensi operasi harian dengan optimalisasi penggunaan ruang memori array.

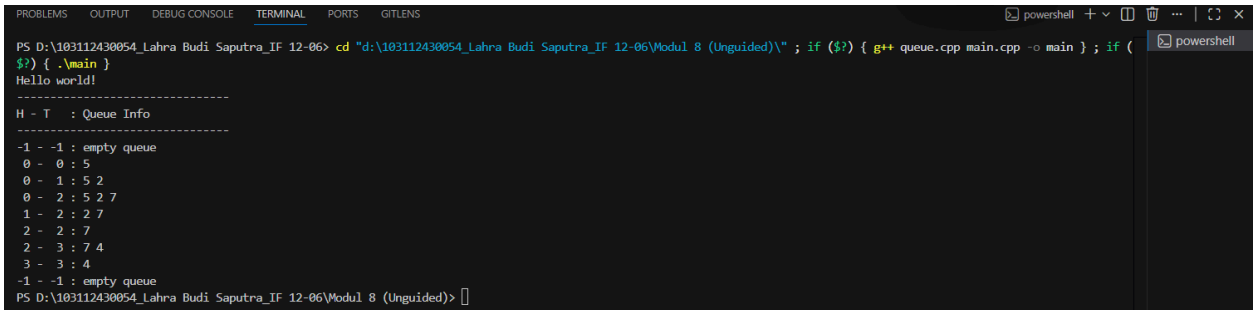
Unguided (SOAL 3)

(queue.cpp)

```
 //* SOAL NOMOR 3*//
//queue Alternatif 3 (head dan tail berputar)
#include "queue.h"
#include <iostream>
using namespace std;

void createQueue(Queue &Q) {
   Q.head = -1;
   Q.tail = -1;
}
bool isEmptyQueue(Queue Q) {
   return Q.head == -1;
}
bool isFullQueue(Queue Q) {
   return ((Q.tail + 1) % MAX == Q.head);
}
void enqueue(Queue &Q, infotype x) {
   if (isFullQueue(Q)) {
      cout << "Antrean Penuh" << endl;
   } else {
      if (isEmptyQueue(Q)) {
         Q.head = 0;
         Q.tail = 0;
         Q.info[0] = x;
      } else {
         Q.tail = (Q.tail + 1) % MAX;
         Q.info[Q.tail] = x;
      }
   }
}
```

```
infotype dequeue(Queue &Q) {
    infotype val = -1;
    if (isEmptyQueue(Q)) {
        cout << "Antrean Kosong" << endl;
    } else {
        val = Q.info[Q.head];
        if (Q.head == Q.tail) {
            createQueue(Q);
        } else {
            Q.head = (Q.head + 1) % MAX;
        }
    }
    return val;
}

void printInfo(Queue Q) {
    if (Q.head != -1 && Q.head < 10) cout << " ";
    cout << Q.head << " - ";
    if (Q.tail != -1 && Q.tail < 10) cout << " ";
    cout << Q.tail << " : ";

    if (isEmptyQueue(Q)) {
        cout << "empty queue" << endl;
    } else {
        int i = Q.head;
        while (true) {
            cout << Q.info[i] << " ";
            if (i == Q.tail) break; // Berhenti jika sudah di posisi Tail
            i = (i + 1) % MAX;      // Pindah index memutar
        }
        cout << endl;
    }
}
```

Screenshots Output:



Deskripsi:

*Kode ini menerapkan ADT Queue menggunakan metode **Circular Buffer** (antrean berputar), di mana pergerakan indeks Head dan Tail bersifat melingkar memanfaatkan operasi matematika modulo. Pendekatan ini adalah yang paling efisien karena sepenuhnya menghilangkan kebutuhan untuk menggeser elemen (shifting); ketika Tail mencapai batas maksimum array, ia akan otomatis kembali ke indeks awal (0) untuk mengisi slot kosong yang ditinggalkan oleh Head. Penentuan kondisi penuh dan kosong didasarkan pada jarak relatif antara Head dan Tail dalam siklus lingkaran tersebut, menjadikan penggunaan memori optimal tanpa mengorbankan kinerja komputasi.*

D. Kesimpulan

Secara keseluruhan, Queue merupakan struktur data linear dengan prinsip *First In First Out* (FIFO) yang memiliki operasi penyisipan di *Tail* dan penghapusan di *Head*, di mana implementasinya menggunakan tabel (Array) menuntut pemilihan algoritma yang tepat demi efisiensi memori dan waktu. Dari tiga metode representasi tabel yang dipelajari, **Alternatif 1** (Head diam) dinilai paling tidak efisien karena mengharuskan pergeseran elemen (*shifting*) setiap kali data diambil, sedangkan **Alternatif 2** (Head bergerak) menawarkan kinerja lebih cepat namun menyisakan masalah "penuh semu" yang memaksa pergeseran elemen saat antrean mencapai batas akhir. Sebagai solusi terbaik, **Alternatif 3** atau konsep *circular buffer* (berputar) dianggap paling optimal karena meniadakan kebutuhan pergeseran elemen sepenuhnya dengan memanfaatkan rotasi indeks, sehingga manajemen memori menjadi jauh lebih efektif

E. Referensi

Pratama, R. A., & Santoso, B. (2024). Optimasi Struktur Data Stack dan Queue Menggunakan Array Dinamis pada Sistem Berkas. *Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK)*, *11*(1), 45-52.

Wijaya, H. (2025). Implementasi Struktur Data Antrian Queue dalam Sistem Penjadwalan Proses pada Sistem Operasi. *Jurnal Publikasi Teknik Informatika*, *4*(2), 112-119.

Saputra, A. (2025). Implementasi Queue Berbasis Linked List Pada Aplikasi Web Manajemen Antrian Print Mahasiswa. *Jurnal Ilmu Komputer dan Informatika*, *2*(2), 88-95.