LAPORAN PRAKTIKUM STRUKTUR DATA

MODUL VIII
QUEUE

Disusun Oleh :

NAMA : ADIKA AUNURFIKRI NOVIYANTO

NIM : 103112400195

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

C++ adalah pengembangan dari dari bahasa c yang dibuat oleh Bjarne Stroustrup sekitar tahun 1980-an. C++ disebut bahasa multi-paradigma, artinya bisa dipakai dengan gaya prosedural (pakai fungsi biasa), beriorientasi objek (pakai class dan object), atau bahkan gabungan keduanya. C++ punya dasar-dasar seperti variabel, operator percabangan (if, switch), perulangan (for, while), dan bisa memakai class untuk membuat objek.

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

Guided 1
queue.h

```
#ifndef QUEUE_H
#define QUEUE_H

const int 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

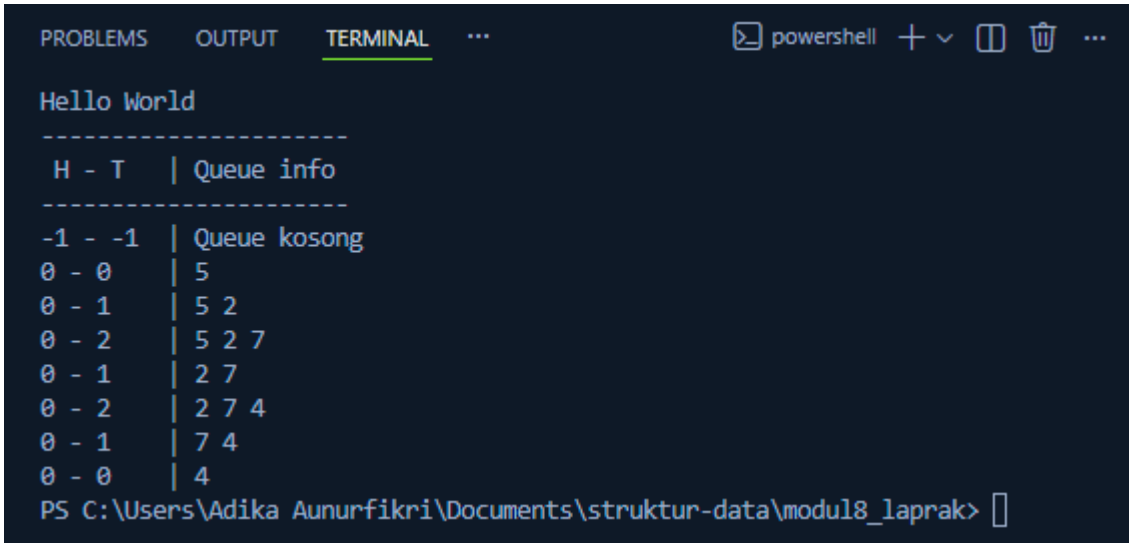
```
#include <iostream>
#include "queue.h"
using namespace std;

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 << "Queue penuh" << endl;
    } else {
        if(isEmptyQueue(Q)){
            Q.head = 0;
            Q.tail = 0;
        } else {
            Q.tail++;
        }
        Q.info[Q.tail] = x;
    }
}

infotype dequeue(Queue &Q){
    infotype x = -1;
```

```
    if(isEmptyQueue(Q)){
        cout << "Queue kosong" << endl;
    } else {
        x = Q.info[Q.head];

        for(int i = Q.head; i < Q.tail; i++){
            Q.info[i] = Q.info[i+1];
        }

        Q.tail--;

        if(Q.tail < Q.head){
            createQueue(Q);
        }
    }
    return x;
}

void printInfo(Queue Q){
    cout << Q.head << " - " << Q.tail << "\t | ";

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

Main.cpp

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

int main() {
    cout << "Hello World" << endl;
    Queue Q;
    createQueue(Q);

    cout<<"----------------------"<<endl;
    cout<<" H - T \t | Queue info"<<endl;
    cout<<"----------------------"<<endl;

    printInfo(Q);
    enqueue(Q,5); printInfo(Q);
    enqueue(Q,2); printInfo(Q);
    enqueue(Q,7); printInfo(Q);
    dequeue(Q);   printInfo(Q);
    enqueue(Q,4); printInfo(Q);
    dequeue(Q);   printInfo(Q);
    dequeue(Q);   printInfo(Q);

    return 0;
}
```

Screenshots Output:



Guided 2
queue.h

```
#ifndef QUEUE_H
#define QUEUE_H

const int 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

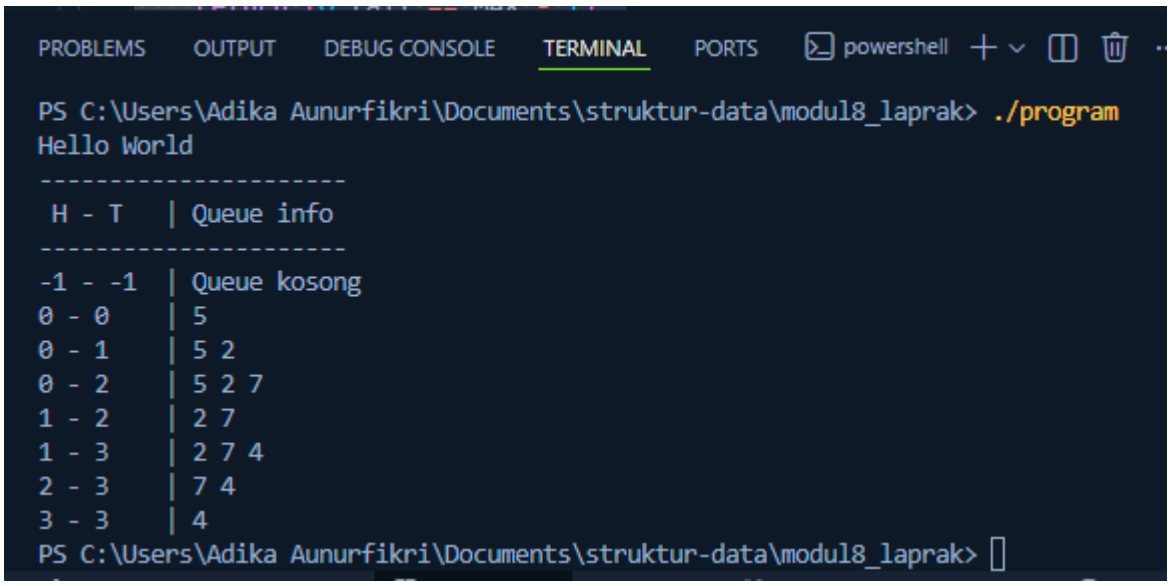
```
#include <iostream>
#include "queue.h"
using namespace std;

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 << "Queue penuh" << endl;
    } else {
        if (isEmptyQueue(Q)) {
            Q.head = 0;
            Q.tail = 0;
        } else {
            Q.tail++;
        }
        Q.info[Q.tail] = x;
    }
}

infotype dequeue(Queue &Q) {
    if (isEmptyQueue(Q)) {
        cout << "Queue kosong" << endl;
```

```
    return -1;
  }

  infotype x = Q.info[Q.head];
  Q.head++;

  if (Q.head > Q.tail) {
    createQueue(Q);
  }

  return x;
}

void printInfo(Queue Q) {
  cout << Q.head << " - " << Q.tail << "\t | ";

  if (isEmptyQueue(Q)) {
    cout << "Queue kosong";
  } else {
    for (int i = Q.head; i <= Q.tail; i++) {
      cout << Q.info[i] << " ";
    }
  }

  cout << endl;
}
```

Main.cpp

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

int main() {
    cout << "Hello World" << endl;
    Queue Q;
    createQueue(Q);

    cout<<"----------------------"<<endl;
    cout<<" H - T \t | Queue info"<<endl;
    cout<<"----------------------"<<endl;

    printInfo(Q);
    enqueue(Q,5); printInfo(Q);
    enqueue(Q,2); printInfo(Q);
    enqueue(Q,7); printInfo(Q);
    dequeue(Q);   printInfo(Q);
    enqueue(Q,4); printInfo(Q);
    dequeue(Q);   printInfo(Q);
    dequeue(Q);   printInfo(Q);

    return 0;
}
```

Screenshots Output:



Guided 3
queue.h

```
#ifndef QUEUE_H
#define QUEUE_H

const int 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

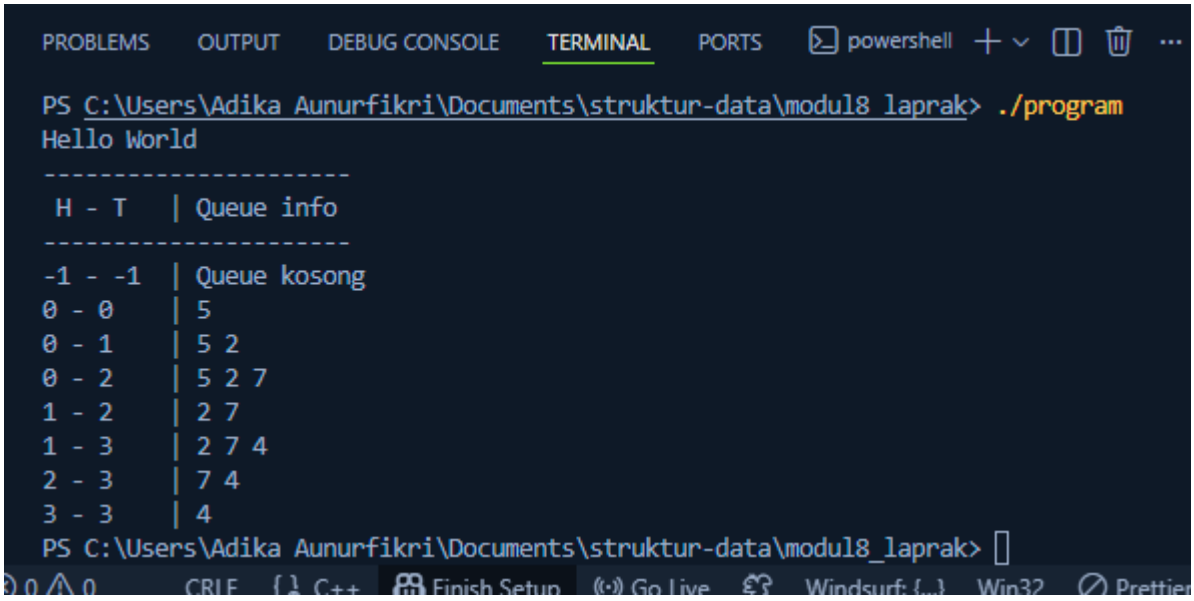
```
#include <iostream>
#include "queue.h"
using namespace std;

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 << "Queue penuh" << endl;
    } else {
        if (isEmptyQueue(Q)) {
            Q.head = 0;
            Q.tail = 0;
        } else {
            Q.tail = (Q.tail + 1) % MAX;
        }
        Q.info[Q.tail] = x;
    }
}

infotype dequeue(Queue &Q) {
```

```
    if (isEmptyQueue(Q)) {
        cout << "Queue kosong" << endl;
        return -1;
    }

    infotype x = Q.info[Q.head];

    if (Q.head == Q.tail) {
        createQueue(Q);
    } else {
        Q.head = (Q.head + 1) % MAX;
    }

    return x;
}

void printInfo(Queue Q) {
    cout << Q.head << " - " << Q.tail << "\t | ";

    if (isEmptyQueue(Q)) {
        cout << "Queue kosong";
    } else {
        int i = Q.head;
        while (true) {
            cout << Q.info[i] << " ";
            if (i == Q.tail) break;
            i = (i + 1) % MAX;
        }
    }

    cout << endl;
}
```

Main.cpp

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

int main() {
    cout << "Hello World" << endl;
    Queue Q;
    createQueue(Q);

    cout<<"----------------------"<<endl;
    cout<<" H - T \t | Queue info"<<endl;
    cout<<"----------------------"<<endl;

    printInfo(Q);
    enqueue(Q,5); printInfo(Q);
    enqueue(Q,2); printInfo(Q);
    enqueue(Q,7); printInfo(Q);
    dequeue(Q);   printInfo(Q);
    enqueue(Q,4); printInfo(Q);
    dequeue(Q);   printInfo(Q);
    dequeue(Q);   printInfo(Q);

    return 0;
}
```

Screenshots Output:



Deskripsi:

Pada praktikum ini diimplementasikan tiga bentuk mekanisme queue menggunakan array, yaitu Alternatif 1, Alternatif 2, dan Alternatif 3. Ketiganya menerapkan prinsip dasar FIFO, namun berbeda pada cara menangani perpindahan head, tail, dan pemanfaatan ruang array. Pada Alternatif 1, head selalu berada pada indeks awal dan setiap operasi dequeue mengharuskan pergeseran seluruh elemen sehingga kurang efisien. Alternatif 2 memperbaiki kelemahan tersebut dengan memajukan head setiap kali dequeue, namun ruang kosong yang sudah dilewati head tidak dapat digunakan kembali, menyebabkan queue dapat menjadi penuh meskipun masih ada slot kosong. Alternatif 3 menjadi bentuk yang paling optimal dengan menggunakan metode circular buffer, di mana head dan tail dapat berputar dari akhir kembali ke awal array, sehingga seluruh kapasitas dapat dimanfaatkan tanpa pergeseran elemen dan tetap mempertahankan operasi enqueue dan dequeue yang efisien.

C. Kesimpulan

Dari ketiga implementasi queue, dapat disimpulkan bahwa perbedaan mekanisme perpindahan head dan tail berpengaruh besar terhadap efisiensi dan pemanfaatan memori. Alternatif 1 memiliki struktur paling sederhana namun paling tidak efisien, Alternatif 2 meningkatkan efisiensi tetapi masih menyisakan ruang tidak terpakai, sedangkan Alternatif 3 merupakan implementasi terbaik karena memanfaatkan seluruh array secara optimal dan menjaga operasi queue tetap memiliki kompleksitas waktu O(1). Dengan demikian, circular queue (Alternatif 3) menjadi solusi paling efektif untuk penggunaan queue berbasis array.

D. Referensi

- W3Resource. (2020). *C++ String Exercises: Convert digit/number to words*.
- GeeksforGeeks. (2020). *Loops in C++ (for, while, do-while)*.