LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL VIII
QUEUE

Disusun Oleh :

NAMA : Taufik Hafit Zakaria

NIM : 103112430093

Dosen

WAHYU ANDI SAPUTRA

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Queue (antrean) merupakan salah satu bentuk struktur data linear yang menerapkan prinsip operasi FIFO (*First In First Out*). Sesuai dengan analoginya seperti antrean di dunia nyata, elemen yang pertama kali dimasukkan ke dalam antrean adalah elemen yang pertama kali akan dikeluarkan atau dilayani. Seluruh operasi pada *queue*, seperti penyisipan (*enqueue*) dan pengambilan (*dequeue*) data, dilakukan pada ujung yang berbeda. Penyisipan elemen baru selalu dilakukan di ujung belakang yang disebut *tail* (ekor), sementara pengambilan elemen dilakukan dari ujung depan yang disebut *head* (kepala). *Queue* dapat diimplementasikan menggunakan dua representasi utama: representasi *pointer* (menggunakan konsep *linked list*) yang dinamis, atau representasi tabel (menggunakan *array*) yang memiliki ukuran terbatas.

Karakteristik Queue

- **Prinsip FIFO**: Operasi didasarkan pada prinsip *First In First Out*, di mana elemen yang pertama masuk akan menjadi yang pertama keluar.
- **Akses di Dua Ujung**: Elemen baru ditambahkan di bagian belakang (*tail*) dan elemen dikeluarkan dari bagian depan (*head*).
- **Operasi Utama (Enqueue & Dequeue)**: Memiliki dua operasi utama: *Enqueue* (menambah elemen ke *tail*) dan *Dequeue* (mengambil elemen dari *head*).

Keuntungan Queue

- **Manajemen Sumber Daya yang Adil**: Sangat ideal untuk mengelola sumber daya atau tugas secara berurutan dan adil, seperti pada sistem *printer spooling* atau penjadwalan CPU, karena memastikan pemrosesan berdasarkan urutan kedatangan.
- **Pemrosesan Data Berurutan**: Efektif digunakan dalam skenario yang membutuhkan pemrosesan data secara berurutan sesuai urutan masuknya, contohnya pada *breadth-first search* (BFS) dalam algoritma graf.
- **Manajemen Lalu Lintas Data**: Berguna dalam menangani aliran data yang tidak sinkron antara dua proses (misalnya, *buffer* data) untuk mencegah kehilangan data.

Keterbatasan Queue

- **Akses Tidak Fleksibel**: Elemen yang berada di tengah antrean tidak dapat diakses secara langsung tanpa terlebih dahulu mengeluarkan elemen-elemen di depannya.
- **Ukuran Statis (Array)**: Implementasi menggunakan *array* memiliki ukuran yang tetap, sehingga berisiko mengalami kondisi *overflow* (antrean penuh) jika jumlah elemen melebihi kapasitas yang ditentukan.
- **Risiko *Underflow***: Terdapat risiko mengalami *underflow* jika mencoba mengambil elemen dari *queue* yang sudah dalam keadaan kosong.

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

Guided 1

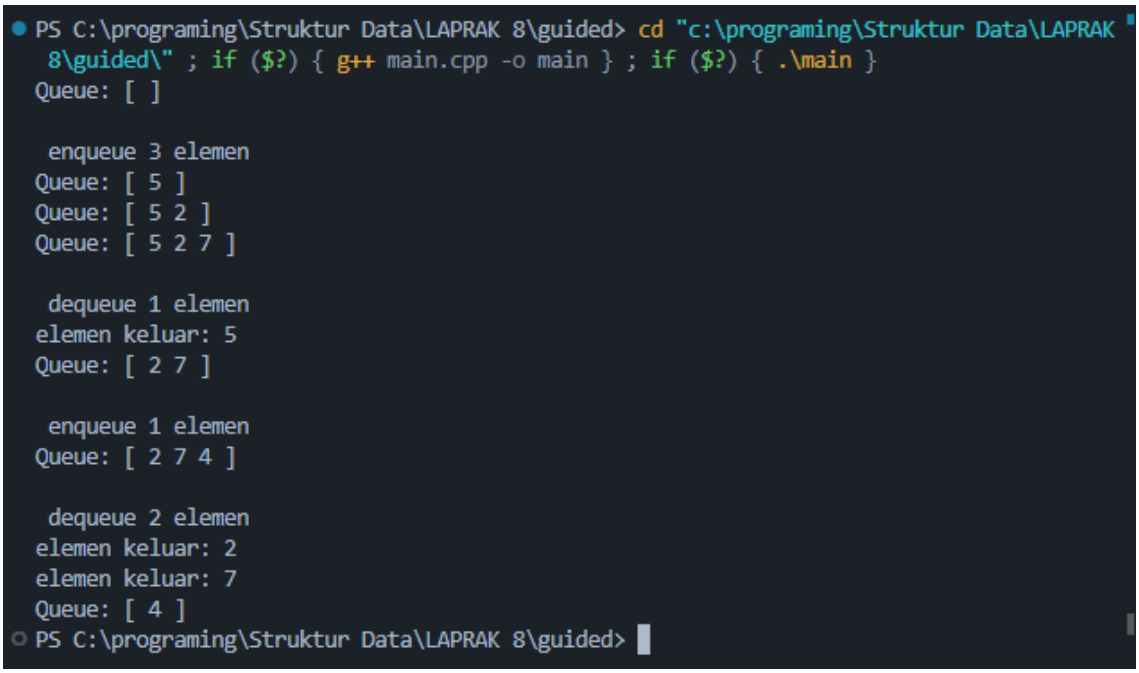
```
//Taufik Hafit Zakaria
//103112430093
//queue.h

#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
```

```cpp
//Taufik Hafit Zakaria
//103112430093
//queue.cpp

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

// Prosedur untuk membuat antrian kosong
void createQueue(Queue &Q){
    Q.head = Q.tail = Q.count = 0;
}

// Fungsi untuk memeriksa apakah antrian kosong
bool isEmpty(Queue Q){
    return Q.count == 0;
}

// Fungsi untuk memeriksa apakah antrian penuh
bool isFull(Queue Q){
    return Q.count == MAX_QUEUE;
}

// Prosedur untuk menambahkan elemen ke dalam antrian
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;
    }
}

```

```
// Fungsi untuk menghapus elemen dari antrian
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;
    }
}

// Prosedur untuk mencetak isi antrian
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;
}
```

```cpp
//Taufik Hafit Zakaria
//103112430093
//main.cpp

#include "queue.h" // Hanya include file header, bukan .cpp
#include "queue.cpp"
#include <iostream>
using namespace std;

int main() {
    // Inisialisasi atau membuat queue kosong
    Queue Q;
    createQueue(Q);
    printInfo(Q);

    // Menambahkan 3 elemen pertama ke dalam queue
    cout << "\n enqueue 3 elemen" << endl;
    enqueue(Q, 5);
    printInfo(Q);
    enqueue(Q, 2);
    printInfo(Q);
    enqueue(Q, 7);
    printInfo(Q);

    // Mengeluarkan 1 elemen dari depan queue
    cout << "\n dequeue 1 elemen" << endl;
    cout << "elemen keluar: " << dequeue(Q) << endl;
    printInfo(Q);

    // Menambahkan 1 elemen baru ke belakang queue
    cout << "\n enqueue 1 elemen" << endl;
    enqueue(Q, 4);
    printInfo(Q);

    // Mengeluarkan 2 elemen lagi dari depan queue
    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 adalah implementasi struktur data *Queue* (antrean) menggunakan C++ dengan pendekatan representasi tabel, atau lebih spesifiknya, menggunakan mekanisme *circular array* (array yang berputar). Intinya, program ini menerapkan prinsip FIFO (*First In First Out*), di mana data pertama yang masuk akan menjadi yang pertama kali keluar. Kode program diorganisir secara modular ke dalam tiga file: queue.h sebagai *header* yang berisi deklarasi struktur dan prototipe fungsi, queue.cpp yang berisi implementasi detail dari setiap fungsi, dan main.cpp yang berfungsi untuk mendemonstrasikan cara kerja *queue*.

Alur program di fungsi main secara langsung mendemonstrasikan cara kerjanya. Awalnya, sebuah *queue* kosong dibuat, lalu program melakukan tiga kali operasi enqueue untuk memasukkan nilai (5, 2, dan 7) ke dalam antrean. Setelah itu, satu elemen terdepan (yaitu 5) dikeluarkan menggunakan dequeue. Selanjutnya, satu elemen baru (nilai 4) ditambahkan lagi ke belakang antrean, dan terakhir, dua elemen kembali dikeluarkan dari depan. Hasil akhir dengan jelas menunjukkan bahwa elemen-elemen diproses sesuai urutan masuknya (5 masuk pertama dan keluar pertama), yang membuktikan bahwa prinsip FIFO berjalan dengan benar dan implementasi *circular array* ini bekerja sesuai harapan.

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

Unguided 1

```h
//Taofik Hafit Zakaria
//103112430093
//queue_unguided.h

#ifndef QUEUE_H_INCLUDE
#define QUEUE_H_INCLUDE

typedef int infotype;

struct Queue {
    infotype info[5];
    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
```

```cpp
//Taofik Hafit Zakaria
//103112430093
//main.cpp

  #include <iostream>
  #include "queue_unguided.h"
  #include "queue_unguided.cpp"
  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);
      dequeue(Q); printInfo(Q);
      enqueue(Q, 4); printInfo(Q);
      dequeue(Q); printInfo(Q);
      dequeue(Q); printInfo(Q);

      return 0;
  }
```

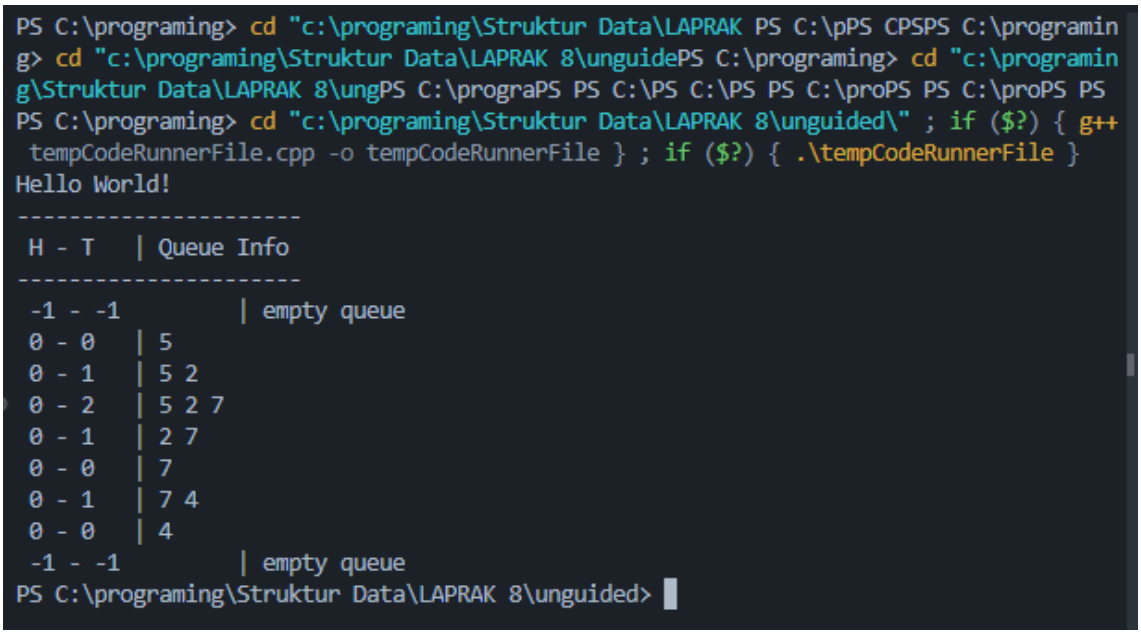
```cpp
//Taofik Hafit Zakaria
//103112430093
//queue_unguided.cpp

#include <iostream>
#include "queue_unguided.h"

using namespace std;

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

bool isEmptyQueue(Queue Q) {
    return (Q.head == 0);
}

bool isFullQueue(Queue Q) {
    return (Q.tail == 5);                    // soal 1

    //return (Q.tail == 5 && Q.head == 1);  // soal 2

     /*if (isEmptyQueue(Q)) {                // soal 3
         return false;
     }
     int nextTail;
     if (Q.tail == 5) {
         nextTail = 1;
     } else {
         nextTail = Q.tail + 1;
     }
     return (nextTail == Q.head);*/
}
```

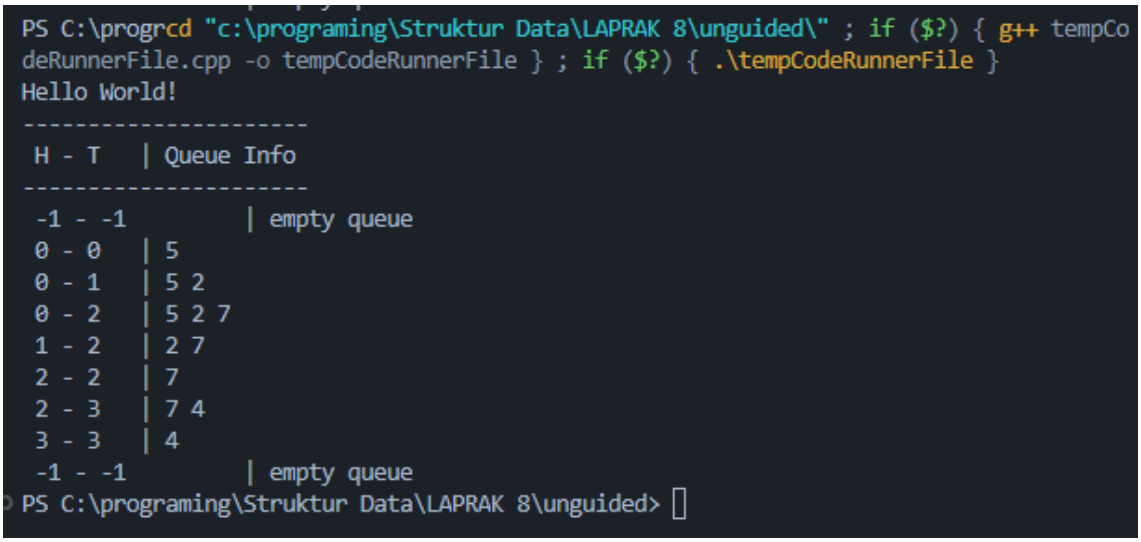
```
void enqueue(Queue &Q, infotype x) {
    // soal 1
    if (isFullQueue(Q)) {
        cout << "Queue penuh!" << endl;
    } else {
        if (isEmptyQueue(Q)) {
            Q.head = 1;
            Q.tail = 1;
            Q.info[Q.tail - 1] = x;
        } else {
            Q.tail++;
            Q.info[Q.tail - 1] = x;
        }
    }

     //soal 2
     /*if (isFullQueue(Q)) {
         cout << "Queue penuh!" << endl;
     } else {
         if (isEmptyQueue(Q)) {
             Q.head = 1;
             Q.tail = 1;
             Q.info[Q.tail - 1] = x;
         } else {
             // Cek apakah perlu geser elemen
             if (Q.tail == 5 && Q.head > 1) {
                 // Geser semua elemen ke depan
                 int j = 1;
                 for (int i = Q.head; i <= Q.tail; i++) {
                     Q.info[j - 1] = Q.info[i - 1];
                     j++;
                 }
                 Q.tail = Q.tail - Q.head + 2;
                 Q.head = 1;
                 Q.info[Q.tail - 1] = x;
             } else {
                 Q.tail++;
                 Q.info[Q.tail - 1] = x;
             }
         }
     }*/

     //soal 3
     /*if (isFullQueue(Q)) {
         cout << "Queue penuh!" << endl;
     } else {
         if (isEmptyQueue(Q)) {
             Q.head = 1;
             Q.tail = 1;
             Q.info[Q.tail - 1] = x;
         } else {
             // Tail bergerak circular
             if (Q.tail == 5) {
                 Q.tail = 1;
             } else {
                 Q.tail++;
             }
             Q.info[Q.tail - 1] = x;
         }
     }*/
}

```

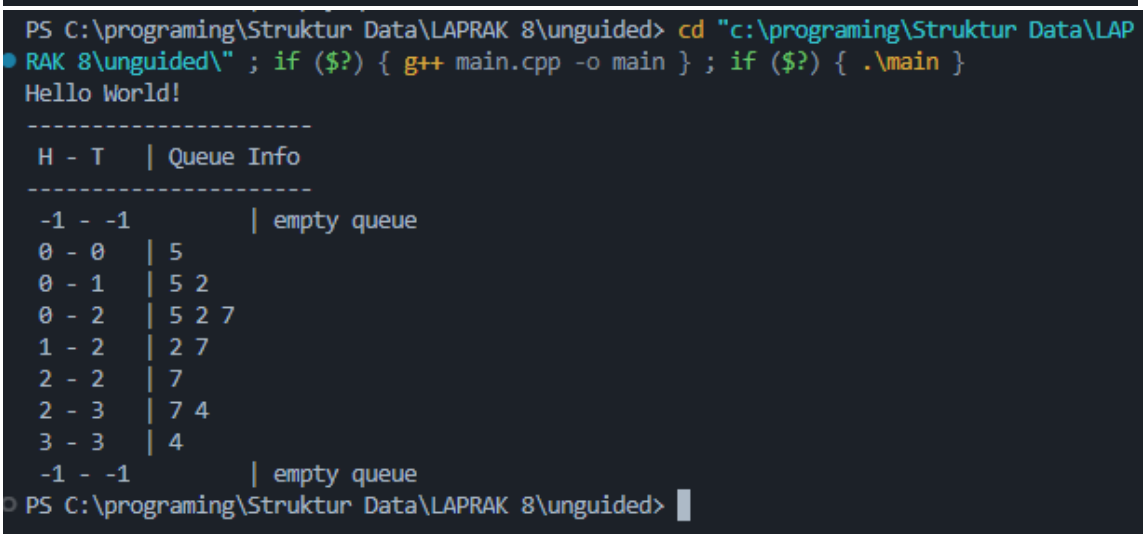
```
infotype dequeue(Queue &Q) {

    //soal 1
    infotype x;

    if (isEmptyQueue(Q)) {
        cout << "Queue kosong!" << endl;
        return -1;
    } else {
        x = Q.info[Q.head - 1];

        if (Q.head == Q.tail) {
            createQueue(Q);
        } else {
            // Geser semua elemen ke depan
            for (int i = Q.head; i < Q.tail; i++) {
                Q.info[i - 1] = Q.info[i];
            }
            Q.tail--;
        }

        return x;
    }

    // soal 2
    /* infotype x;

     if (isEmptyQueue(Q)) {
         cout << "Queue kosong!" << endl;
         return -1;
     } else {
         x = Q.info[Q.head - 1];

         if (Q.head == Q.tail) {
             createQueue(Q);
         } else {
             // Head bergerak maju (TIDAK ada pergeseran)
             Q.head++;
         }

         return x;
     }*/

    // soal 3
     /*infotype x;
     if (isEmptyQueue(Q)) {
         cout << "Queue kosong!" << endl;
         return -1;
     } else {
         x = Q.info[Q.head - 1];

         if (Q.head == Q.tail) {
             createQueue(Q);
         } else {
             // Head bergerak circular
             if (Q.head == 5) {
                 Q.head = 1;
             } else {
                 Q.head++;
             }
         }

         return x;
     }*/
}
```

```
162
163
164 void printInfo(Queue Q) {
165
166     //soal 1 & 2
167     if (isEmptyQueue(Q)) {
168         cout << " " << Q.head - 1 << " - " << Q.tail - 1 << " \t | empty queue" << endl;
169     } else {
170         cout << " " << Q.head - 1 << " - " << Q.tail - 1 << " \t | ";
171
172         for (int i = Q.head; i <= Q.tail; i++) {
173             cout << Q.info[i - 1];
174             if (i < Q.tail) {
175                 cout << " ";
176             }
177         }
178         cout << endl;
179     }
180
181     // soal 3
182      /*if (isEmptyQueue(Q)) {
183          cout << " " << Q.head - 1 << " - " << Q.tail - 1 << " \t | empty queue" << endl;
184      } else {
185          cout << " " << Q.head - 1 << " - " << Q.tail - 1 << " \t | ";
186
187          // Print dengan konsep circular
188          int i = Q.head;
189          bool first = true;
190
191          while (true) {
192              if (!first) cout << " ";
193              cout << Q.info[i - 1];
194              first = false;
195
196              if (i == Q.tail) break;
197
198              // Bergerak circular
199              if (i == 5) {
200                  i = 1;
201              } else {
202                  i++;
203              }
204          }
205          cout << endl;
206      }*/
207 }
```

Screenshots Output







Deskripsi:

Program ini adalah implementasi struktur data *Queue* menggunakan C++ dengan pendekatan representasi tabel (array), yang secara spesifik dikembangkan melalui tiga alternatif mekanisme yang berbeda. Intinya, program ini menerapkan prinsip FIFO (*First In First Out*), di mana data yang pertama kali masuk akan menjadi yang pertama kali keluar. Kode program diorganisir ke dalam tiga file (.h, .cpp, dan main), di mana file .cpp secara unik menyimpan jejak tiga pendekatan implementasi (soal 1, soal 2, dan soal 3) dalam bentuk komentar, dengan versi *circular array* yang paling efisien dipilih sebagai implementasi yang aktif.

Alur program di fungsi main secara langsung mendemonstrasikan cara kerjanya. Awalnya, program melakukan serangkaian operasi enqueue untuk memasukkan nilai (5, 2, 7), yang diikuti dengan beberapa operasi dequeue untuk mengeluarkan elemen dari depan. Kemudian, sebuah nilai baru (4) ditambahkan lagi ke antrean sebelum sisa elemen dikeluarkan hingga antrean kembali kosong. Rangkaian proses ini secara efektif menguji pergerakan *head* dan *tail*. Hasil akhir dengan jelas menunjukkan bahwa setiap elemen dikeluarkan sesuai urutan kedatangannya, membuktikan bahwa prinsip FIFO berjalan dengan benar dan implementasi *circular array* yang efisien ini bekerja sesuai harapan.

D. Kesimpulan

Dari hasil praktikum yang telah dilakukan, dapat disimpulkan bahwa struktur data *Queue* memungkinkan pengelolaan data secara efisien berdasarkan prinsip FIFO (*First In First Out*). Melalui implementasi program latihan ini, konsep dasar operasi *queue* seperti createQueue, enqueue, dan dequeue, serta berbagai mekanisme representasi tabel (array) dapat dipahami dan diterapkan dengan baik. Praktikum ini juga secara khusus mendemonstrasikan perbandingan antara beberapa alternatif implementasi array, hingga pada akhirnya menerapkan mekanisme *circular array* yang paling efisien karena dapat menggunakan kembali ruang kosong tanpa perlu menggeser elemen. Akses data pada *queue* selalu terjadi di dua ujung yang berbeda, yaitu penyisipan di *tail* (belakang) dan penghapusan di *head* (depan).

Selain itu, program ini membuktikan bahwa penggunaan *Queue* sangat berguna dalam kasus-kasus yang memerlukan pemrosesan data secara berurutan sesuai urutan kedatangan, seperti pada simulasi antrean, penjadwalan proses pada sistem operasi, atau manajemen *buffer* data. Dengan memahami prinsip dasar FIFO dan cara kerjanya, mahasiswa dapat secara efektif menerapkan *Queue* untuk memecahkan berbagai masalah algoritmik yang menuntut pemrosesan yang adil dan terurut. Pemahaman ini juga menjadi fondasi penting untuk mempelajari struktur data lain yang lebih kompleks seperti *priority queue* atau *deque*, serta algoritma yang memanfaatkan *Queue* dalam implementasinya, seperti *Breadth-First Search* (BFS) pada *graph*.

E. Referensi

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++* (2nd ed.). John Wiley & Sons.

GeeksforGeeks. (2024). *Queue Data Structure*. Diakses pada tahun 2025 dari https://www.geeksforgeeks.org/queue-data-structure/

Programiz. (2024). *Queue Data Structure*. Diakses pada tahun 2025 dari https://www.programiz.com/dsa/queue