LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 8
QUEUE

Disusun Oleh :
NAMA : FANDIKA PRIMADANI
NIM : 103112400231

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Stack adalah struktur data linear yang menerapkan prinsip **LIFO (Last In, First Out)**, di mana elemen yang terakhir dimasukkan akan menjadi elemen pertama yang dikeluarkan. Operasi dasarnya meliputi **push** (menambah data), **pop** (menghapus data teratas), dan **printInfo** (menampilkan isi stack).

B. SOAL 1

queue.h

```h
#ifndef QUEUE_H
#define QUEUE_H

const int MAX = 5;

typedef int infotype;

struct Queue {
    infotype info[MAX];
    int head, 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

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

void createQueue(Queue &Q) {
    Q.head = 0;
    Q.tail = -1;
}

bool isEmptyQueue(Queue Q) {
```

```
    return (Q.tail == -1);
}

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

void enqueue(Queue &Q, infotype x) {
    if (isFullQueue(Q)) {
        cout << "Queue penuh!" << endl;
        return;
    }

    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];

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

    Q.tail--;

    if (Q.tail < Q.head)
        Q.tail = -1;

    return x;
}

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

    if (isEmptyQueue(Q)) {
        cout << "empty queue" << endl;
        return;
```

```
    }

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

main.cpp

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

    return 0;
}
```

SOAL 2

queue1.h

```
#ifndef QUEUE_H
#define QUEUE_H

const int MAX = 5;

typedef int infotype;

struct Queue {
    infotype info[MAX];
    int head, 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
```

queue1.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 && Q.tail == -1);
}

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

```
void enqueue(Queue &Q, infotype x) {
    if (isFullQueue(Q)) {
        cout << "Queue penuh!" << endl;
        return;
    }

    if (isEmptyQueue(Q)) {
        Q.head = 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 hasil = Q.info[Q.head];

    if (Q.head == Q.tail) {
        Q.head = Q.tail = -1;
    } else {
        Q.head++;
    }

    return hasil;
}

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

    if (isEmptyQueue(Q)) {
        cout << "empty queue" << endl;
        return;
    }

    for (int i = Q.head; i <= Q.tail; i++) {
        cout << Q.info[i] << " ";
```

```
    }
    cout << endl;
```

main1.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;
}
```

SOAL 3

stack2.h

```
#ifndef QUEUE_H
#define QUEUE_H

const int MAX = 5;

typedef int infotype;
```

```
struct Queue {
    infotype info[MAX];
    int head, 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
```

queue2.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;
        return;
    }

    if (isEmptyQueue(Q)) {
        Q.head = 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 data = Q.info[Q.head];

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

    return data;
}

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

    if (isEmptyQueue(Q)) {
        cout << "empty queue" << endl;
        return;
    }

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

main2.cpp

```
#include <iostream>
```

```cpp
#include "queue.h"
#include "queue2.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);
    enqueue(Q, 4);  printInfo(Q);
    enqueue(Q, 9);  printInfo(Q);

    dequeue(Q);     printInfo(Q);
    dequeue(Q);     printInfo(Q);

    enqueue(Q, 8);  printInfo(Q);
    enqueue(Q, 1);  printInfo(Q);

    enqueue(Q, 6);  printInfo(Q);

    return 0;
}
```

Screenshots Output

1. Output 1

```
PS D:\SEMESTER 3\Pratikum Struktur Data\Pratikum Modul9>
cpp -o main } ; if ($?) { .\main }
Hello World
------------------------
H - T \t | Queue info
------------------------
0 - -1 | empty queue
0 - 0 | 5
0 - 1 | 5 2
0 - 2 | 5 2 7
0 - 1 | 2 7
0 - 2 | 2 7 4
0 - 1 | 7 4
0 - 0 | 4
```

2. Output 2 Carilah elemen dengan nomor polisi B001P dengan membuat fungsi baru.

```
PS D:\SEMESTER 3\Pratikum Struktur Data\Pratikum Modul9>
cpp -o main } ; if ($?) { .\main }
Hello World
-------------------------
H - T \t | Queue Info
-------------------------
-1 - -1 | empty queue
0 - 0 | 5
0 - 1 | 5 2
0 - 2 | 5 2 7
1 - 2 | 2 7
1 - 3 | 2 7 4
2 - 3 | 7 4
3 - 3 | 4
```

3. Output 3 hapus elemen dengan nomor polisi B002P dengan procedure delete.

```
PS D:\SEMESTER 3\Pratikum Struktur Data\Pratikum Modul9>
.cpp -o main2 } ; if ($?) { .\main2 }
Hello World
-------------------------
H - T    | Queue Info
-------------------------
-1 - -1 | empty queue
0 - 0 | 5
0 - 1 | 5 2
0 - 2 | 5 2 7
0 - 3 | 5 2 7 4
0 - 4 | 5 2 7 4 9
1 - 4 | 2 7 4 9
2 - 4 | 7 4 9
2 - 0 | 7 4 9 8
2 - 1 | 7 4 9 8 1
Queue penuh!
2 - 1 | 7 4 9 8 1
PS D:\SEMESTER 3\Pratikum Struktur Data\Pratikum Modul9>
```

Deskripsi:

Soal 1 : Pada implementasi queue pertama, digunakan array statis dengan mekanisme head tetap di indeks 0 sementara tail bergerak ke kanan setiap kali dilakukan enqueue. Queue dianggap kosong jika tail bernilai –1 dan penuh jika tail mencapai indeks maksimum. Saat enqueue, tail dinaikkan dan elemen baru disimpan di posisi tersebut. Saat dequeue, elemen pertama diambil lalu seluruh elemen digeser satu posisi ke kiri agar head tetap berada di indeks 0. Karena selalu menggeser elemen, metode ini mudah dipahami tetapi kurang efisien untuk operasi dequeue.

Soal 2 : Pada implementasi queue alternatif 2, head dan tail sama-sama dapat bergerak, sehingga elemen tidak perlu digeser ketika dilakukan dequeue. Queue dianggap kosong jika head dan tail bernilai –1, dan dianggap penuh jika tail mencapai indeks akhir array. Saat enqueue, jika queue kosong maka head dan tail diatur ke 0, namun jika tidak, tail cukup dinaikkan lalu elemen baru disimpan pada posisi tersebut. Saat dequeue, elemen diambil dari posisi head, kemudian head dinaikkan satu langkah; jika setelah penghapusan head melewati tail, maka queue kembali kosong dengan mengatur head dan tail ke –1. Metode ini lebih efisien dibanding alternatif pertama karena tidak ada shifting elemen, meskipun ruang array yang sudah dilewati head tidak bisa dipakai kembali.

Soal 3 : Pada implementasi queue alternatif 3, digunakan konsep *circular array* di mana head dan tail bergerak melingkar menggunakan operasi modulus sehingga seluruh ruang array dapat digunakan tanpa perlu menggeser elemen. Queue dianggap kosong jika head bernilai –1, dan dianggap penuh jika posisi (tail + 1) % MAX sama dengan head. Saat enqueue, jika queue kosong maka head dan tail diatur ke 0; jika tidak, tail digeser ke posisi berikutnya secara melingkar menggunakan (tail + 1) % MAX, lalu elemen baru disimpan di posisi tersebut. Saat dequeue, elemen diambil dari posisi head, lalu head dipindah ke indeks berikutnya secara melingkar; jika head bertemu dengan tail, queue kembali kosong dan kedua indeks diset menjadi –1. Metode ini merupakan versi paling efisien karena tidak ada shifting dan seluruh kapasitas array dapat dimanfaatkan penuh.

C. Kesimpulan

Kesimpulannya, ketiga alternatif implementasi queue menunjukkan perbedaan strategi dalam mengelola posisi head dan tail pada struktur array. Alternatif 1 menggunakan head yang tetap dan melakukan pergeseran elemen setiap kali dequeue sehingga mudah dipahami namun kurang efisien. Alternatif 2 meningkatkan efisiensi dengan menggerakkan head dan tail tanpa melakukan shifting, tetapi ruang array yang sudah dilewati head tidak dapat digunakan kembali. Alternatif 3 menjadi solusi terbaik melalui teknik circular queue, di mana head dan tail bergerak melingkar dengan operasi modulus sehingga seluruh kapasitas array dapat dimanfaatkan tanpa shifting dan tanpa membuang ruang. Dengan demikian, setiap alternatif memiliki karakteristik dan kegunaannya sendiri, namun circular queue merupakan pilihan paling optimal untuk implementasi queue berbasis array.

D. Referensi

Drozdek, Adam. *Data Structures and Algorithms in C++.* Cengage Learning, 2012
Malik, D.S. *Data Structures Using C++.* Course Technology, 2010.