

## MODUL 4

### QUEUE ARRAY



#### A. Definisi

Dalam kehidupan sehari-hari, sering kali kita menemukan adanya antrian (*queue*). Antrian merupakan suatu kondisi dimana suatu objek harus menunggu giliran untuk melakukan suatu tindakan. Konsep dasar dari suatu antrian adalah objek yang telah ada atau menunggu terlebih dahulu akan mendapat kesempatan terlebih dahulu dibandingkan objek yang ada atau menunggu setelahnya.

Materi *queue* pada mata kuliah Informasi dan Struktur Data menerapkan konsep yang sama dengan konsep *queue* atau antrian yang ada di kehidupan sehari-hari. Konsep *queue* secara umum adalah **First In First Out (FIFO)** atau **Last In Last Out (LILO)**, dimana objek yang masuk terlebih dahulu akan keluar terlebih dahulu pula. Pada materi ini, konsep *queue* akan diterapkan pada struktur data *array*.

#### B. Komponen Struktur Data Queue Array

Terdapat beberapa komponen pada struktur data dengan konsep *queue*, seperti:

1. **Head:** index dari data yang pertama kali masuk yang masih berada dalam *queue array*. Head juga menjadi index data yang akan dihapus (dequeue).
2. **Tail:** index dari data yang terakhir kali masuk ke dalam *queue array*. Tail juga menjadi index tempat memasukkan data baru (enqueue).

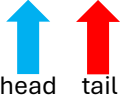
### C. Ilustrasi Sirkulasi Data pada Queue Array

Berikut adalah ilustrasi sirkulasi data pada *queue array*:

1. Kondisi *queue array* yang masih kosong (baru saja diinisialisasi)

$head = -1, tail = -1$

Data			
Index	0	1	2



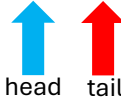
The diagram shows two arrows pointing to the index -1 position. A blue arrow labeled 'head' and a red arrow labeled 'tail' both point to the position before index 0.

Pada kondisi *queue array* yang masih kosong, index *head* dan *tail* diatur dengan nilai -1 yang artinya berada di luar *array* (index *array* dimulai dari 0).

2. Dilakukan enqueue data “A”

$head = 0, tail = 0$

Data	A		
Index	0	1	2



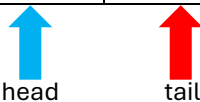
The diagram shows two arrows pointing to index 0. A blue arrow labeled 'head' and a red arrow labeled 'tail' both point to the cell containing 'A' at index 0.

Ketika *queue array* dalam keadaan kosong, enqueue dilakukan dengan mengatur nilai *head* dan *tail* dengan nilai 0 kemudian data dimasukkan pada index *tail*.

3. Dilakukan enqueue data “B”

$head = 0, tail = 1$

Data	A	B	
Index	0	1	2




The diagram shows two arrows pointing to index 0 and index 1. A blue arrow labeled 'head' points to the cell containing 'A' at index 0. A red arrow labeled 'tail' points to the cell containing 'B' at index 1.

Ketika *queue array* tidak dalam keadaan kosong, tidak dalam keadaan penuh, dan nilai *tail* tidak sama dengan ukuran *array* dikurang 1, dilakukan inkremen pada nilai *tail* kemudian data dimasukkan pada index *tail*.

## 4. Dilakukan enqueue data “C”

$head = 0, tail = 2$

Data	A	B	C
Index	0	1	2




Ketika *queue array* tidak dalam keadaan kosong, tidak dalam keadaan penuh, dan nilai *tail* tidak sama dengan ukuran *array* dikurang 1, dilakukan inkremen pada nilai *tail* kemudian data dimasukkan pada index *tail*.

5. Dilakukan enqueue data “D”, namun gagal karena *queue array* telah penuh.

$head = 0, tail = 2$

Data	A	B	C
Index	0	1	2

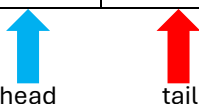


Ketika *queue array* dalam keadaan penuh, proses enqueue tidak dapat melewati pengecekan fungsi *isFull* dan data tidak dapat dimasukkan ke dalam *array*.

## 6. Dilakukan dequeue

$head = 1, tail = 2$

Data		B	C
Index	0	1	2

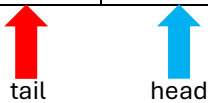


Ketika *queue array* tidak dalam keadaan kosong dan tidak hanya terdapat satu data pada *array*, dilakukan pengecekan apakah nilai *head* sama dengan ukuran *array* dikurang 1. Karena nilai *head* tidak sama dengan ukuran *array* dikurang 1, maka nilai *head* diinkremen dan data pada index *head* sebelum diinkremen dianggap kosong.

## 7. Dilakukan enqueue data “D”

$head = 1, tail = 0$

Data	D	B	C
Index	0	1	2




Ketika *queue array* tidak dalam keadaan kosong, tidak dalam keadaan penuh, dan nilai *tail* sama dengan ukuran *array* dikurang 1, nilai *tail* diatur dengan nilai 0 kemudian data dimasukkan pada index *tail*.

## 8. Dilakukan dequeue

$head = 2, tail = 0$

Data	D		C
Index	0	1	2

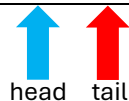


Ketika *queue array* tidak dalam keadaan kosong dan tidak hanya terdapat satu data pada *array*, dilakukan pengecekan apakah nilai *head* sama dengan ukuran *array* dikurang 1. Karena nilai *head* tidak sama dengan ukuran *array* dikurang 1, maka nilai *head* diinkremen dan data pada index *head* sebelum diinkremen dianggap kosong.

## 9. Dilakukan dequeue

$head = 0, tail = 0$

Data	D		
Index	0	1	2



Ketika *queue array* tidak dalam keadaan kosong dan tidak hanya terdapat satu data pada *array*, dilakukan pengecekan apakah nilai *head* sama dengan ukuran *array* dikurang 1. Karena nilai *head* sama dengan ukuran *array* dikurang 1, maka nilai *head* diatur dengan nilai 0 dan data pada index *head* sebelum diinkremen dianggap kosong.

## 10. Dilakukan dequeue

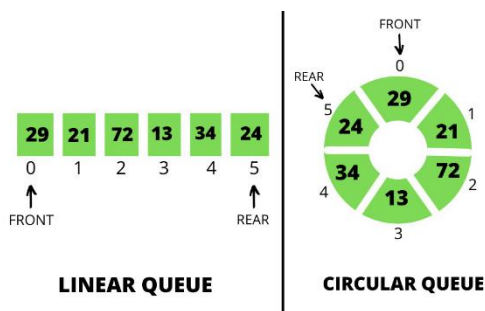
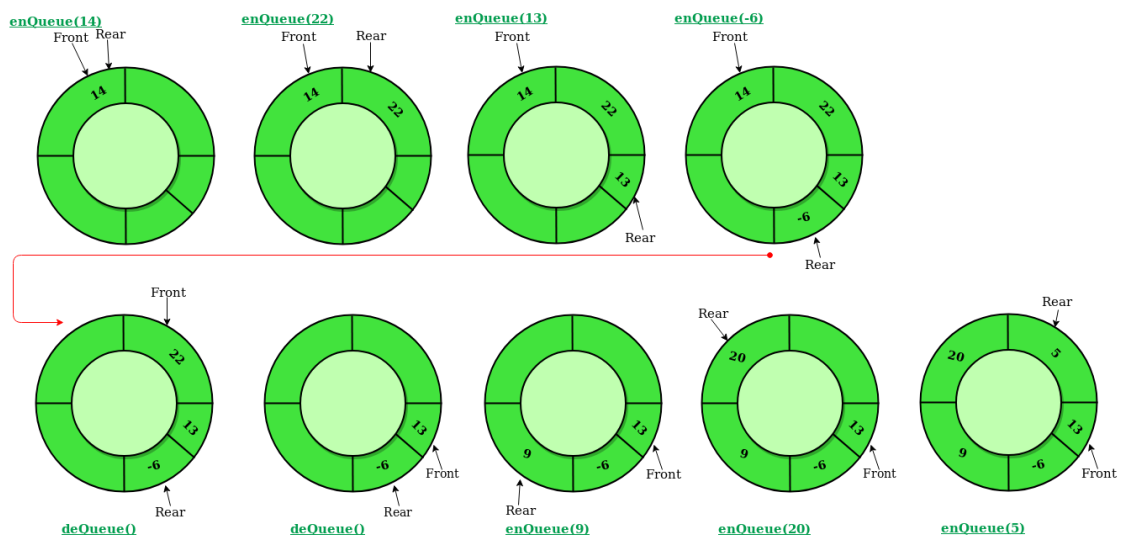
$head = -1, tail = -1$

Data			
Index	0	1	2

↑ head    ↑ tail

Ketika *queue array* tidak dalam keadaan kosong dan hanya terdapat satu data pada *array*, dilakukan prosedur *createEmpty* sehingga *queue array* kembali kepada kondisi seperti pertama kali diinisialisasi.

Struktur data *queue array* bekerja secara sirkular, sehingga juga dapat diilustrasikan sebagai berikut:

1. Ilustrasi *queue array* secara sirkular2. Ilustrasi enqueue dan dequeue pada *queue array* secara sirkular

#### D. Operasi pada Struktur Data *Queue Array*

1. **createEmpty**: prosedur createEmpty digunakan untuk menginisialisasi sebuah *queue array*.

```
void createEmpty(Queue *Q){
    (*Q).head = -1;
    (*Q).tail = -1;
}
```

Prosedur createEmpty digunakan untuk menginisialisai *queue array* dalam keadaan kosong dengan nilai head dan tail diset -1 karena index *array* dimulai dari 0 (*index* dibuat di luar *index* seharusnya (*out of bound*) untuk menyatakan *array* kosong).

2. **isFull**: fungsi isFull merupakan fungsi dengan tipe data balikan boolean untuk memeriksa apakah *queue array* penuh atau tidak. Fungsi isFull juga dapat dibuat dengan tipe data int.

```
bool isFull(Queue Q){
    if((Q.head < Q.tail && Q.tail - Q.head == MAX_SIZE-1)
    || (Q.head > Q.tail && Q.head - Q.tail == 1))
        return true;
    else
        return false;
}
```

Terdapat 2 kemungkinan kondisi untuk menyatakan sebuah *queue array* dalam keadaan penuh:

- Kemungkinan 1: jika nilai head < tail dan selisih nilai head dengan tail sama dengan nilai ukuran array dikurangi 1 maka *queue array* dalam keadaan penuh.
- Kemungkinan 2: jika nilai head > tail dan selisih nilai head dengan tail sama dengan 1 maka *queue array* dalam keadaan penuh.

Jika salah satu kemungkinan terpenuhi, fungsi isFull akan mengembalikan nilai *true*, sedangkan jika kedua kemungkinan tersebut tidak terpenuhi maka fungsi isFull akan mengembalikan nilai *false*.

3. **isEmpty:** fungsi isEmpty merupakan fungsi dengan tipe data balikan boolean untuk memeriksa apakah *queue array* kosong atau tidak. Fungsi isEmpty juga dapat dibuat dengan tipe data int.

```
bool isEmpty(Queue Q){
    if(Q.head == -1 && Q.tail == -1)
        return true;
    else
        return false;
}
```

*Queue array* dinyatakan kosong apabila nilai head dan tail = -1 (sesuai dengan nilai yang dibuat ketika inisialisasi (prosedur createEmpty)).

4. **isOneElement:** fungsi isOneElement merupakan fungsi dengan tipe data balikan boolean untuk memeriksa apakah hanya tersisa satu data pada *queue array* atau tidak. Fungsi isOneElement juga dapat dibuat dengan tipe data int.

```
bool isOneElement(Queue Q){
    if(Q.head == Q.tail && !isEmpty(Q))
        return true;
    else
        return false;
}
```

Suatu *queue array* dinyatakan hanya memiliki satu data apabila nilai head sama dengan nilai tail dan *queue array* tidak kosong.

5. **Enqueue:** operasi enqueue digunakan untuk menambahkan ke dalam *queue array*.

```
void Enqueue(Queue *Q, Data D){
    if(isFull(*Q))
        printf("\nAntrian penuh");
    else{
        if(isEmpty(*Q)){
            (*Q).head = (*Q).tail = 0;
            (*Q).P[(*Q).tail] = D;
        }else{
            if((*Q).tail == MAX_SIZE - 1)
                (*Q).tail = 0;
            else
                (*Q).tail++;

            (*Q).P[(*Q).tail] = D;
        }
        printf("\nBerhasil enqueue data");
    }
}
```

Prosedur enqueue memiliki pengecekan dengan fungsi `isFull` sehingga tidak terdapat kemungkinan terjadinya jumlah data yang lebih banyak dari ukuran *array*. Apabila *array* tidak penuh, fungsi enqueue akan mengecek apakah *array* masih kosong atau tidak. Jika *array* masih kosong maka *head* dan *tail* diberi nilai 0, kemudian data ditambahkan di *index* tail. Jika *array* tidak kosong akan dilakukan pengecekan, jika nilai *tail* sama dengan ukuran *array* dikurang 1 (nilai *tail* adalah index terakhir *array*) maka *tail* diberi nilai 0 (dikembalikan ke index pertama *array*), jika tidak maka nilai *tail* akan diinkremen. Kemudian data ditambahkan di *index* tail.

6. **Dequeue:** operasi dequeue digunakan untuk mengeluarkan data dari dalam *queue array*.

```
void Dequeue(Queue *Q){
    if(isEmpty(*Q))
        printf("\nAntrian kosong");
    else{
        if(isOneElement(*Q))
            createEmpty(&(*Q));
        else{
            if((*Q).head == MAX_SIZE - 1)
                (*Q).head = 0;
            else
                (*Q).head++;
        }
        printf("\nBerhasil dequeue data");
    }
}
```

Prosedur dequeue memiliki pengecekan dengan fungsi `isEmpty` sehingga tidak terdapat kemungkinan terjadinya dequeue pada *array* yang kosong. Apabila *array* tidak kosong, fungsi dequeue akan mengecek apakah hanya terdapat satu data pada *array* atau tidak. Jika hanya terdapat satu data pada *array* (pengecekan menggunakan fungsi `isOneElement`), prosedur akan memanggil prosedur `createEmpty` untuk menginisialisasi kembali *queue array* (jika data terakhir didequeue maka akan sama seperti *queue array* yang masih kosong dan baru diinisialisasi). Jika tidak hanya terdapat satu data pada *array*, dilakukan pengecekan apakah nilai *head* sama dengan ukuran *array* dikurang 1 (berada di index terakhir *array*). Jika nilai *head* sama dengan ukuran *array* dikurang 1 maka



head diberi nilai 0 (data yang dikeluarkan selanjutnya berada di index pertama array), jika tidak maka nilai head akan diinkremen.

7. **printQueue:** prosedur printQueue merupakan prosedur untuk menampilkan data pada *queue array* secara berurutan sesuai urutan masuk datanya.



```
void printQueue(Queue Q){
    if(isEmpty(Q))
        printf("\nAntrian kosong");
    else{
        int i;

        if(Q.head <= Q.tail){
            for(i = Q.head; i <= Q.tail; i++)
                printData(Q.D[i]);
        }else{
            for(i = Q.head; i <= MAX_SIZE - 1; i++)
                printData(Q.D[i]);

            for(i = 0; i <= Q.tail; i++)
                printData(Q.D[i]);
        }
    }
}
```

Prosedur printQueue memiliki pengecekan dengan fungsi isEmpty sehingga tidak akan menampilkan data apabila *queue array* kosong. Jika *array* tidak kosong, prosedur akan melakukan pengecekan apakah nilai head kurang dari sama dengan tail atau tidak. Jika nilai head kurang dari sama dengan tail maka dilakukan print data secara berurutan dari index head hingga index tail. Jika nilai head tidak kurang dari sama dengan tail maka dilakukan print data secara berurutan dari index head hingga ukuran array dikurang 1 kemudian dilanjutkan print data mulai dari index 0 hingga index tail.

8. **searchData:** fungsi searchData merupakan fungsi dengan tipe data balikan int untuk mencari data pada *queue array*.

```
int searchData(Queue Q, Data D){
    int i;

    if(Q.head <= Q.tail){
        for(i = Q.head; i <= Q.tail; i++){
            if(Q.D[i] == D)
                return i;
        }
    }else{
        for(i = Q.head; i <= MAX_SIZE - 1; i++){
            if(Q.D[i] == D)
                return i;
        }

        for(i = 0; i <= Q.tail; i++){
            if(Q.D[i] == D)
                return i;
        }
    }
    return -1;
}
```

Prosedur searchData memiliki pengecekan dengan fungsi isEmpty sehingga tidak akan mencari data apabila *queue array* kosong. Jika *array* tidak kosong, prosedur akan melakukan pengecekan apakah nilai head kurang dari sama dengan tail atau tidak. Jika nilai head kurang dari sama dengan tail maka dilakukan pengecekan, apabila data pada index tersebut sama dengan data yang dicari maka fungsi akan mengembalikan nilai index dari data tersebut. Jika nilai head tidak kurang dari sama dengan tail maka dilakukan pengecekan, apabila data pada index tersebut sama dengan data yang dicari maka fungsi akan mengembalikan nilai index dari data tersebut. Jika data belum ditemukan, dilanjutkan pencarian mulai dari index 0 hingga index tail. Jika data masih tidak ditemukan dapat disimpulkan data yang dicari tidak ada dan fungsi akan mengembalikan nilai -1.

## GUIDED

### header.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#define MAX_SIZE 5

// NAMA LENGKAP
// NPM
// KELAS

typedef char string[50];

typedef struct {
    string maskapai, tujuan;
} Penerbangan;

typedef struct {
    Penerbangan P[MAX_SIZE];
    int head, tail;
} Queue;

Penerbangan createPenerbangan(string maskapai, string tujuan);
void createEmpty(Queue *Q);
bool isEmpty(Queue Q);
bool isFull(Queue Q);
bool isOneElement(Queue Q);

void Enqueue(Queue *Q, Penerbangan P);
void Dequeue(Queue *Q);

void printPenerbangan(Penerbangan P);
void printQueue(Queue Q);
int searchPenerbangan(Queue Q, string maskapai);
```

**source.c**

```
#include "header.h"

// NAMA LENGKAP
// NPM
// KELAS

Penerbangan createPenerbangan(string maskapai, string tujuan){
    Penerbangan temp;

    strcpy(temp.maskapai, maskapai);
    strcpy(temp.tujuan, tujuan);

    return temp;

    /*createProduk merupakan fungsi untuk mempermudah dalam
    memasukkan data*/
}

void createEmpty(Queue *Q){
    (*Q).head = -1;
    (*Q).tail = -1;
}

bool isEmpty(Queue Q){
    if(Q.head == -1 && Q.tail == -1)
        return true;
    else
        return false;
}

/*terdapat bentuk lain penulisan code fungsi isEmpty:
bool isEmpty(Queue Q){
    return (Q.head == -1 && Q.tail == -1);
}
*/

bool isFull(Queue Q){
    if((Q.head < Q.tail && Q.tail - Q.head == MAX_SIZE-1) ||
        (Q.head > Q.tail && Q.head - Q.tail == 1))
        return true;
    else
        return false;
}

/*terdapat bentuk lain penulisan code fungsi isFull:
bool isFull(Queue Q){
    return ((Q.head < Q.tail && Q.tail-Q.head == MAX_SIZE-1) ||
        (Q.head > Q.tail && Q.head - Q.tail == 1));
}
*/

bool isOneElement(Queue Q){
    if(Q.head == Q.tail && !isEmpty(Q))
        return true;
    else
        return false;
}

/*terdapat bentuk lain penulisan code fungsi isOneElement:
bool isOneElement(Queue Q){
    return (Q.head == Q.tail && !isEmpty(Q));
}
*/
```

```

void Enqueue(Queue *Q, Penerbangan P){
    if(isFull(*Q))
        printf("\nAntrian lepas landas penuh");
    else{
        if(isEmpty(*Q)){
            (*Q).head = (*Q).tail = 0;
            (*Q).P[(*Q).tail] = P;
        }else{
            if((*Q).tail == MAX_SIZE - 1)
                (*Q).tail = 0;
            else
                (*Q).tail++;

            (*Q).P[(*Q).tail] = P;
        }
        printf("\nPesawat %s dengan tujuan %s ditambahkan ke
antrian lepas landas", P.maskapai, P.tujuan);
    }
}

void Dequeue(Queue *Q){
    Penerbangan temp = (*Q).P[(*Q).head];

    if(isEmpty(*Q))
        printf("\nAntrian lepas landas kosong");
    else{
        if(isOneElement(*Q))
            createEmpty(&(*Q));
        else{
            if((*Q).head == MAX_SIZE - 1)
                (*Q).head = 0;
            else
                (*Q).head++;
        }
        printf("\nPesawat %s dengan tujuan %s telah lepas landas",
temp.maskapai, temp.tujuan);
    }
}

void printPenerbangan(Penerbangan P){
    printf("\nMaskapai : %s", P.maskapai);
    printf("\nTujuan      : %s\n", P.tujuan);

    //menampilkan satu data penerbangan
}

void printQueue(Queue Q){
    if(isEmpty(Q))
        printf("\nAntrian lepas landas kosong");
    else{
        int i;

        if(Q.head <= Q.tail){
            for(i = Q.head; i <= Q.tail; i++)
                printPenerbangan(Q.P[i]);
        }else{
            for(i = Q.head; i <= MAX_SIZE - 1; i++)
                printPenerbangan(Q.P[i]);

            for(i = 0; i <= Q.tail; i++)
                printPenerbangan(Q.P[i]);
        }
    }
}

```

```
int searchPenerbangan(Queue Q, string maskapai){
    int i;

    if(Q.head <= Q.tail){
        for(i = Q.head; i <= Q.tail; i++){
            if(strcmpi(Q.P[i].maskapai, maskapai)==0)
                return i;
        }
    }else{
        for(i = Q.head; i <= MAX_SIZE - 1; i++){
            if(strcmpi(Q.P[i].maskapai, maskapai)==0)
                return i;
        }

        for(i = 0; i <= Q.tail; i++){
            if(strcmpi(Q.P[i].maskapai, maskapai)==0)
                return i;
        }
    }
    return -1;
}
```

## main.c

```
#include "header.h"

// NAMA LENGKAP
// NPM
// KELAS

int main(int argc, char *argv[]) {

    Queue Q;
    int menu;
    string maskapai, tujuan;

    createEmpty(&Q);
    do{
        system("cls");
        printf("\n\t=== WAKANDA AIRPORT ===\n");
        printf("\n[1] Tambah Antrian (Enqueue)");
        printf("\n[2] Lepas Landas (Dequeue)");
        printf("\n[3] Tampil Antrian");
        printf("\n[4] Cari Antrian");
        printf("\n[0] Exit");
        printf("\n>>> "); scanf("%d", &menu);

        switch(menu){
            case 1:
                printf("\n[Tambah Antrian]");
                printf("\nMaskapai : ");fflush(stdin);gets(maskapai);
                printf("\nTujuan : ");fflush(stdin);gets(tujuan);
                Enqueue(&Q, createPenerbangan(maskapai, tujuan));
                break;
            case 2:
                printf("\n[Lepas Landas]");
                Dequeue(&Q);
                break;
            case 3:
                printf("\n[Tampil Antrian]");
                printQueue(Q);
                break;
            case 4:
                printf("\n[Cari Antrian]");
                printf("\nMasukkan maskapai : ");fflush(stdin);gets(maskapai);
                if(searchPenerbangan(Q, maskapai) == -1)
                    printf("Maskapai tidak ditemukan");
                else
                    printPenerbangan(Q.P[searchPenerbangan(Q, maskapai)]);
                break;
            case 0:
                printf("Press any key to Exit");
                break;
            default:
                printf("\nMenu tidak tersedia");
                break;
        }
        getch();
    }while(menu != 0);

    return 0;
}
```

**Petunjuk pengumpulan Guided:**

1. Pastikan Anda telah membaca materi yang ada pada modul ini sebelum mengerjakan Guided.
2. Kerjakan Guided secara mandiri, dilarang melakukan copy paste code dari sumber manapun.
3. Pastikan Anda memahami setiap prosedur dan fungsi pada Guided struktur data Queue.
4. Kumpulkan seluruh file project Dev-C dengan memasukkannya ke dalam sebuah folder kemudian dikirimkan dalam format file zip.
5. Format penamaan project, folder, dan file zip: GD4\_X\_YYYYY

Keterangan:

X : kelas

Y : 5 digit terakhir NPM

6. Tidak ada toleransi keterlambatan dengan alasan apapun.
7. Ketentuan dan peraturan lainnya mengikuti peraturan praktikum mata kuliah Informasi dan Struktur Data.