



## Struktur Data

Pendidikan Teknik Informatika

Universitas Negeri Padang

TIM DOSEN ©2022

# JOBSHEET (JS-05)

## Queue

Fakultas	Proram Studi	Kode MK	Waktu
Teknik	Pendidikan Teknik Informatika	TIK1.61.2317	4 x 50 Menit

### **TUJUAN PRAKTIKUM**

1. Mahasiswa mampu mengaplikasikan konsep queue dalam menyelesaikan kasus.

### **HARDWARE & SOFTWARE**

1. Personal Computer
2. IDE: Dev C++

---

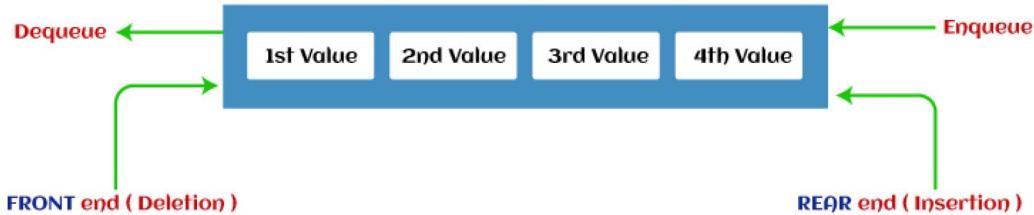
### **TEORI SINGKAT**

#### **A. Queue (Antrian)**

Antrian adalah struktur data yang mirip dengan antrian di dunia nyata. Antrian adalah struktur data di mana apa yang lebih dulu akan keluar lebih dulu, dan mengikuti kebijakan FIFO (First-In-First-Out). Antrian juga dapat didefinisikan sebagai daftar atau kumpulan di mana penyisipan dilakukan dari satu ujung yang dikenal sebagai ujung **belakang** atau **ekor** antrian, sedangkan penghapusan dilakukan dari ujung lain yang dikenal sebagai ujung **depan** atau **kepala** antrian.

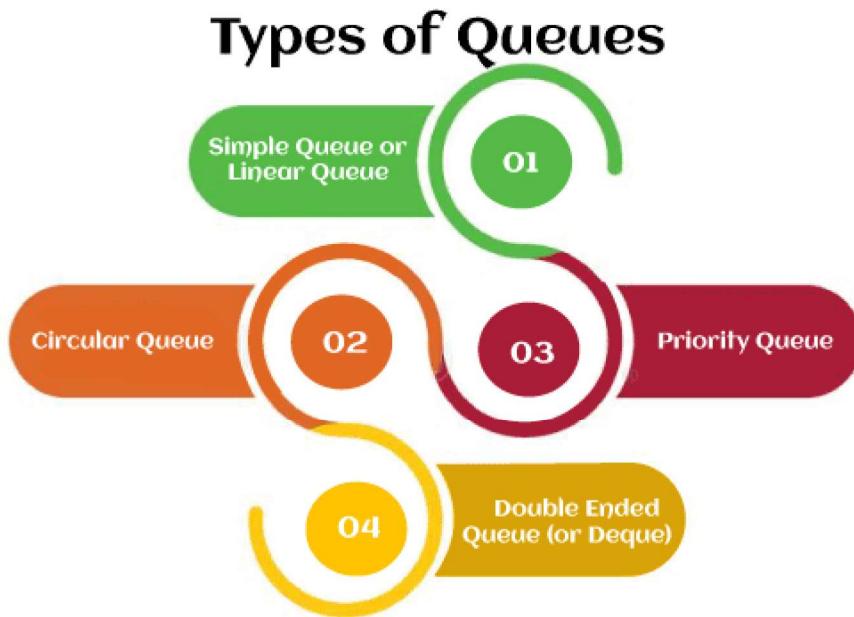
Contoh antrean di dunia nyata adalah antrean tiket di luar gedung bioskop, di mana orang yang masuk terlebih dahulu dalam antrean mendapatkan tiket terlebih dahulu, dan orang terakhir yang masuk dalam antrean akhirnya mendapatkan tiket. Pendekatan serupa diikuti dalam antrian dalam struktur data.

Representasi antrian ditunjukkan pada gambar di bawah ini :



## B. Jenis Antrian

Ada empat jenis antrian sebagai berikut :



1. Antrian Sederhana atau Antrian Linear
2. Antrian Melingkar
3. Antrian Prioritas
4. Antrian Berakhir Ganda (Deque)

## C. Antrian Sederhana atau Antrian Linear

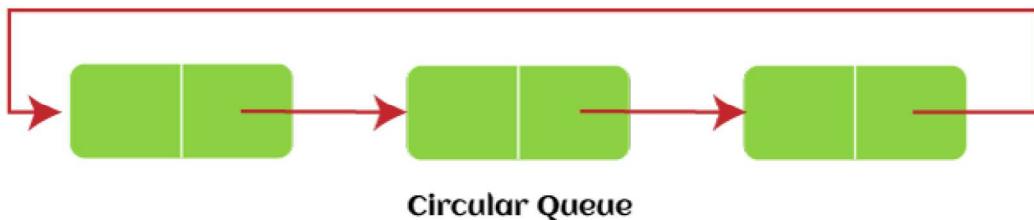
Dalam Linear Queue, penyisipan terjadi dari satu ujung sementara penghapusan terjadi dari ujung yang lain. Ujung di mana penyisipan terjadi dikenal sebagai ujung belakang, dan ujung di mana penghapusan terjadi dikenal sebagai ujung depan. Ini secara ketat mengikuti aturan FIFO.



Kelemahan utama menggunakan Antrian linier adalah bahwa penyisipan dilakukan hanya dari ujung belakang. Jika tiga elemen pertama dihapus dari Antrian, kita tidak dapat menyisipkan lebih banyak elemen meskipun ruang tersedia dalam Antrian Linier. Dalam hal ini, Antrian linier menunjukkan kondisi overflow karena bagian belakang menunjuk ke elemen Antrian terakhir.

#### D. Antrian Melingkar

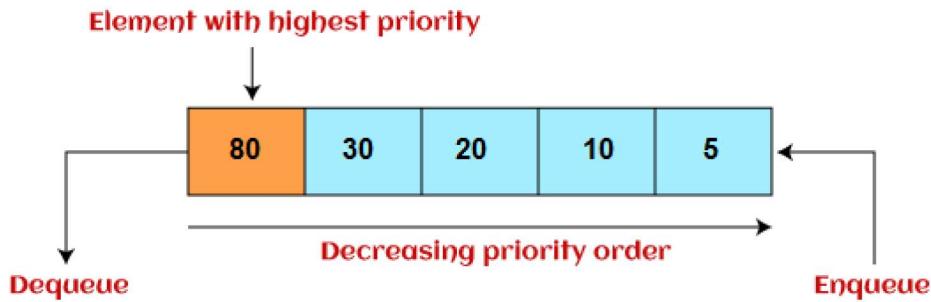
Dalam Circular Queue, semua node direpresentasikan sebagai lingkaran. Ini mirip dengan Antrian linier kecuali bahwa elemen terakhir dari antrian terhubung ke elemen pertama. Ini juga dikenal sebagai Ring Buffer, karena semua ujungnya terhubung ke ujung yang lain. Representasi antrian melingkar ditunjukkan pada gambar di bawah ini :



Kelemahan yang terjadi pada antrian linier diatasi dengan menggunakan antrian melingkar. Jika ruang kosong tersedia dalam antrian melingkar, elemen baru dapat ditambahkan di ruang kosong hanya dengan menambah nilai belakang. Keuntungan utama menggunakan antrian melingkar adalah pemanfaatan memori yang lebih baik.

#### E. Antrian Prioritas

Ini adalah jenis antrian khusus di mana elemen-elemennya disusun berdasarkan prioritas. Ini adalah tipe khusus dari struktur data antrian di mana setiap elemen memiliki prioritas yang terkait dengannya. Misalkan beberapa elemen terjadi dengan prioritas yang sama, mereka akan diatur menurut prinsip FIFO. Representasi antrian prioritas ditunjukkan pada gambar berikut ini :



Penyisipan dalam antrian prioritas terjadi berdasarkan kedatangan, sedangkan penghapusan dalam antrian prioritas terjadi berdasarkan prioritas. Antrian prioritas terutama digunakan untuk mengimplementasikan algoritma penjadwalan CPU.

Ada dua jenis antrian prioritas yang dibahas sebagai berikut :

1. **Antrian prioritas naik** - Dalam antrian prioritas naik, elemen dapat dimasukkan dalam urutan arbitrer, tetapi hanya yang terkecil yang dapat dihapus terlebih dahulu. Misalkan sebuah array dengan elemen 7, 5, dan 3 dalam urutan yang sama, maka penyisipan dapat dilakukan dengan urutan yang sama, tetapi urutan penghapusan elemen adalah 3, 5, 7.
2. **Antrian prioritas menurun** - Dalam antrian prioritas menurun, elemen dapat dimasukkan dalam urutan arbitrer, tetapi hanya elemen terbesar yang dapat dihapus terlebih dahulu. Misalkan sebuah array dengan elemen 7, 3, dan 5 dalam urutan yang sama, maka penyisipan dapat dilakukan dengan urutan yang sama, tetapi urutan penghapusan elemen adalah 7, 5, 3.

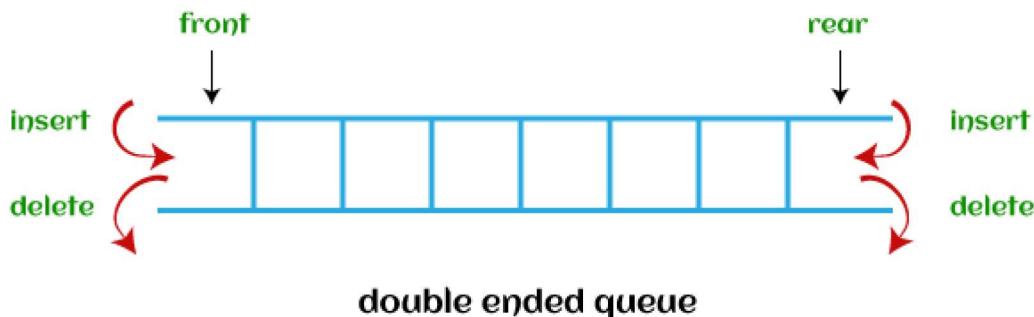
#### F. Deque (Antrian Berakhir Ganda)

Pada Deque atau Double Ended Queue, penyisipan dan penghapusan dapat dilakukan dari kedua ujung antrian baik dari depan maupun belakang. Ini berarti bahwa kita dapat menyisipkan dan menghapus elemen dari ujung depan dan belakang antrian. Deque dapat digunakan sebagai pemeriksa palindrom artinya jika kita membaca string dari kedua ujungnya, maka string tersebut akan sama.

Deque dapat digunakan baik sebagai tumpukan dan antrian karena memungkinkan operasi penyisipan dan penghapusan di kedua ujungnya. Deque dapat dianggap sebagai stack karena stack mengikuti prinsip LIFO (Last In First Out) dimana penyisipan dan penghapusan keduanya hanya dapat dilakukan dari satu ujung. Dan dalam deque,

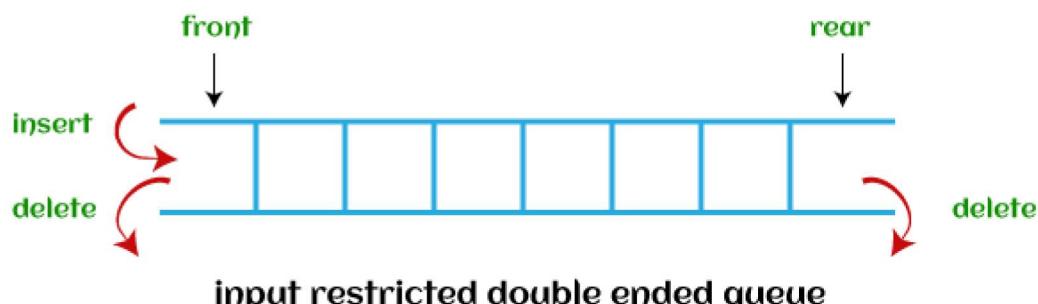
dimungkinkan untuk melakukan penyisipan dan penghapusan dari satu ujung, dan Deque tidak mengikuti prinsip FIFO.

Representasi deque ditunjukkan pada gambar di bawah ini :

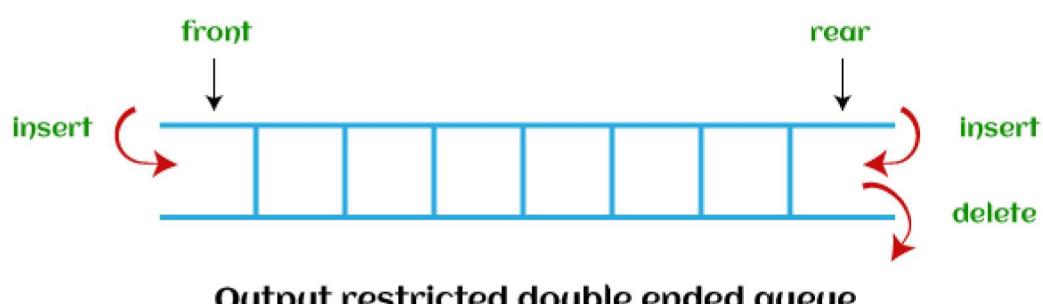


Ada dua jenis deque yang dibahas sebagai berikut :

1. **Input dibatasi deque** - Seperti namanya, dalam antrian input terbatas, operasi penyisipan dapat dilakukan hanya pada satu ujung, sedangkan penghapusan dapat dilakukan dari kedua ujungnya.



2. **Output dibatasi deque** - Seperti namanya, dalam antrian output terbatas, operasi penghapusan dapat dilakukan hanya pada satu ujung, sedangkan penyisipan dapat dilakukan dari kedua ujungnya.



## G. Operasi dilakukan pada antrian

Operasi dasar yang dapat dilakukan pada antrian terdaftar sebagai berikut -

1. **Enqueue:** Operasi Enqueue digunakan untuk menyisipkan elemen di bagian belakang antrian.
2. **Dequeue:** Ini melakukan penghapusan dari front-end antrian. Itu juga mengembalikan elemen yang telah dihapus dari front-end. Ini mengembalikan nilai integer.
3. **Peek:** Ini adalah operasi ketiga yang mengembalikan elemen, yang ditunjuk oleh pointer depan dalam antrian tetapi tidak menghapusnya.
4. **Queue overflow (isfull):** Ini menunjukkan kondisi overflow ketika antrian benar-benar penuh.
5. **Queue underflow (isempty):** Ini menunjukkan kondisi underflow saat Queue kosong, yaitu, tidak ada elemen di Queue.

## LATIHAN

### 1. Implementasi array pada queue

```
/* Nama File : implementasi array pada queue
Pembuat : tuliskan nama dan NIM anda
Tgl pembuatan : tuliskan tanggal hari ini*/\n\n#include<stdio.h>
#include<stdlib.h>
#define maxsize 5
void insert();
void hapus();
void display();
int front = -1, rear = -1;
int queue[maxsize];
int main ()
{
    int choice;
    while(choice != 4)
    {
        printf("\n=====Implement Queue Using
Array=====\\n");
        printf("=====\\n");
    }
}
```

```

=====
Main
Menu*****
=====

printf("\n=====
=====

printf("\n1.Insert an element\n2.Delete an
element\n3.Display the queue\n4.Exit\n");
printf("\nEnter your choice ?");
scanf("%d",&choice);
switch(choice)
{
    case 1:
        insert();
        break;
    case 2:
        hapus();
        break;
    case 3:
        display();
        break;
    case 4:
        exit(0);
        break;
    default:
        printf("\nEnter valid choice??\n");
}
}

void insert()
{
    int item;
    printf("\nEnter the element\n");
    scanf("\n%d",&item);
    if(rear == maxsize-1)
    {
        printf("\nOVERFLOW\n");
        return;
    }
    if(front == -1 && rear == -1)
    {
        front = 0;
        rear = 0;
    }
}

```

```
    else
    {
        rear = rear+1;
    }
    queue[rear] = item;
    printf("\nValue inserted ");
}
void hapus()
{
    int item;
    if (front == -1 || front > rear)
    {
        printf("\nUNDERFLOW\n");
        return;
    }
    else
    {
        item = queue[front];
        if(front == rear)
        {
            front = -1;
            rear = -1 ;
        }
        else
        {
            front = front + 1;
        }
        printf("\nvalue deleted ");
    }
}

void display()
{
    int i;
    if(rear == -1)
    {
        printf("\nEmpty queue\n");
    }
    else
    {   printf("\nprinting values ..... \n");
        for(i=front;i<=rear;i++)
    }
```

```

    {
        printf("\n%d\n", queue[i]);
    }
}
}

```

## 2. Implementasi linked list pada queue

```

/* Nama File : implementasi linked list pada queue
Pembuat : tuliskan nama dan NIM anda
Tgl pembuatan : tuliskan tanggal hari ini*/

#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *front;
struct node *rear;
void insert();
void hapus();
void display();
int main ()
{
    int choice;
    while(choice != 4)
    {
        printf("\n=====Implement Queue Using
Linked List=====\\n");
        printf("\n=====\\n");
        printf("\n*****Main
Menu*****\\n");
        printf("\n=====\\n");
        printf("\n1.Insert an element\\n2.Delete an
element\\n3.Display the queue\\n4.Exit\\n");
        printf("\nEnter your choice ?");
        scanf("%d", & choice);
        switch(choice)
        {

```

```
        case 1:  
            insert();  
            break;  
        case 2:  
            hapus();  
            break;  
        case 3:  
            display();  
            break;  
        case 4:  
            exit(0);  
            break;  
        default:  
            printf("\nEnter valid choice??\n");  
    }  
}  
}  
void insert()  
{  
    struct node *ptr;  
    int item;  
  
    ptr = (struct node *) malloc (sizeof(struct node));  
    if(ptr == NULL)  
    {  
        printf("\nOVERFLOW\n");  
        return;  
    }  
    else  
    {  
        printf("\nEnter value?\n");  
        scanf("%d",&item);  
        ptr -> data = item;  
        if(front == NULL)  
        {  
            front = ptr;  
            rear = ptr;  
            front -> next = NULL;  
            rear -> next = NULL;  
        }  
        else  
        {  
            rear -> next = ptr;  
            rear = ptr;  
    }  
}
```

```

        rear->next = NULL;
    }
}
}

void hapus()
{
    struct node *ptr;
    if(front == NULL)
    {
        printf("\nUNDERFLOW\n");
        return;
    }
    else
    {
        ptr = front;
        front = front -> next;
        free(ptr);
    }
}
void display()
{
    struct node *ptr;
    ptr = front;
    if(front == NULL)
    {
        printf("\nEmpty queue\n");
    }
    else
    {
        printf("\nprinting values ..... \n");
        while(ptr != NULL)
        {
            printf("\n%d\n",ptr -> data);
            ptr = ptr -> next;
        }
    }
}

```

### 3. Implementasi circular queue menggunakan array

```

/* Nama File : implementasi circular queue menggunakan array
Pembuat : tuliskan nama dan NIM anda
Tgl pembuatan : tuliskan tanggal hari ini*/

```

```
#include <stdio.h>

#define max 6
int queue[max]; // array declaration
int front=-1;
int rear=-1;
// function to insert an element in a circular queue
void enqueue(int element)
{
    if(front===-1 && rear===-1) // condition to check queue
        is empty
    {
        front=0;
        rear=0;
        queue[rear]=element;
    }
    else if((rear+1)%max==front) // condition to check
        queue is full
    {
        printf("Queue is overflow..");
    }
    else
    {
        rear=(rear+1)%max; // rear is incremented
        queue[rear]=element; // assigning a value to
        the queue at the rear position.
    }
}

// function to delete the element from the queue
int dequeue()
{
    if((front===-1) && (rear===-1)) // condition to check
        queue is empty
    {
        printf("\nQueue is underflow..");
    }
}
```

```

        }

    else if(front==rear)
    {
        printf("\nThe dequeued element is %d", queue[front]);
        front=-1;
        rear=-1;
    }
    else
    {
        printf("\nThe dequeued element is %d", queue[front]);
        front=(front+1)%max;
    }
}

// function to display the elements of a queue
void display()
{
    int i=front;
    if(front===-1 && rear===-1)
    {
        printf("\n Queue is empty..");
    }
    else
    {
        printf("\nElements in a Queue are :");
        while(i<=rear)
        {
            printf("%d,", queue[i]);
            i=(i+1)%max;
        }
    }
}

int main()
{
    int choice=1,x; // variables declaration

    while(choice<4 && choice!=0) // while loop

```

```
{  
    printf("\n=====Implementation of circular queue  
using Array=====\\n");  
  
    printf("=====\\n");  
    printf("=====Main  
Menu*****\\n");  
  
    printf("=====\\n");  
    printf("=====\\n");  
    printf("nPress 1: Insert an element");  
    printf("nPress 2: Delete an element");  
    printf("nPress 3: Display the element");  
    printf("nEnter your choice");  
    scanf("%d", &choice);  
  
    switch(choice)  
    {  
  
        case 1:  
  
            printf("Enter the element which is to be  
inserted");  
            scanf("%d", &x);  
            enqueue(x);  
            break;  
        case 2:  
            dequeue();  
            break;  
        case 3:  
            display();  
  
    } }  
    return 0;  
}
```

#### 4. Implementasi deque

```
/* Nama File : implementasi deque
Pembuat      : tuliskan nama dan NIM anda
Tgl pembuatan : tuliskan tanggal hari ini*/



#include <stdio.h>
#define size 5
int deque[size];
int f = -1, r = -1;
// insert_front function will insert the value from the
front
void insert_front(int x)
{
    if((f==0 && r==size-1) || (f==r+1))
    {
        printf("Overflow");
    }
    else if((f==-1) && (r==-1))
    {
        f=r=0;
        deque[f]=x;
    }
    else if(f==0)
    {
        f=size-1;
        deque[f]=x;
    }
    else
    {
        f=f-1;
        deque[f]=x;
    }
}

// insert_rear function will insert the value from the rear
void insert_rear(int x)
```

```
{  
    if( (f==0 && r==size-1) || (f==r+1) )  
    {  
        printf("Overflow");  
    }  
    else if( (f==-1) && (r==-1) )  
    {  
        r=0;  
        deque[r]=x;  
    }  
    else if(r==size-1)  
    {  
        r=0;  
        deque[r]=x;  
    }  
    else  
    {  
        r++;  
        deque[r]=x;  
    }  
}  
  
}  
  
// display function prints all the value of deque.  
void display()  
{  
    int i=f;  
    printf("\nElements in a deque are: ");  
  
    while(i!=r)  
    {  
        printf("%d ",deque[i]);  
        i=(i+1)%size;  
    }  
    printf("%d",deque[r]);  
}
```

```

// getfront function retrieves the first value of the deque.

void getfront()
{
    if((f===-1) && (r===-1))
    {
        printf("Deque is empty");
    }
    else
    {
        printf("\nThe value of the element at front is:
%d", deque[f]);
    }
}

// getrear function retrieves the last value of the deque.

void getrear()
{
    if((f===-1) && (r===-1))
    {
        printf("Deque is empty");
    }
    else
    {
        printf("\nThe value of the element at rear is %d",
deque[r]);
    }
}

// delete_front() function deletes the element from the
front

void delete_front()
{
    if((f===-1) && (r===-1))
    {

```

```
        printf("Deque is empty");
    }
else if(f==r)
{
    printf("\nThe deleted element is %d", deque[f]);
    f=-1;
    r=-1;

}
else if(f==(size-1))
{
    printf("\nThe deleted element is %d", deque[f]);
    f=0;
}
else
{
    printf("\nThe deleted element is %d", deque[f]);
    f=f+1;
}
}

// delete_rear() function deletes the element from the rear
void delete_rear()
{
    if((f==-1) && (r==-1))
    {
        printf("Deque is empty");
    }
    else if(f==r)
    {
        printf("\nThe deleted element is %d", deque[r]);
        f=-1;
        r=-1;
    }
    else if(r==0)
    {
```

```

        printf("\nThe deleted element is %d", deque[r]);
        r=size-1;
    }
    else
    {
        printf("\nThe deleted element is %d", deque[r]);
        r=r-1;
    }
}

int main()
{
    printf("\n=====Implementation of
Deque=====\\n");

    printf("=====\\n");
    insert_front(20);
    insert_front(10);
    insert_rear(30);
    insert_rear(50);
    insert_rear(80);
    display(); // Calling the display function to retrieve
the values of deque
    getfront(); // Retrieve the value at front-end
    getrear(); // Retrieve the value at rear-end
    delete_front();
    delete_rear();
    display(); // calling display function to retrieve
values after deletion
    return 0;
}

```

## TUGAS

Buatlah Program Queue Data Mahasiswa. Data mahasiswa yang diinputkan minimal 3 data (Nama, NIM dan Prodi). Tampilan Menu Utama sebagai berikut:

### PROGRAM QUEUE DATA MAHASISWA

#### MENU UTAMA:

- [1]. Enque
- [2]. Deque
- [3]. Clear
- [4]. Print
- [5]. Cek antrian
- [6]. Exit

Pilihan Anda?

### PROGRAM QUEUE DATA MAHASISWA

#### MENU UTAMA:

- [1]. Enque
- [2]. Deque
- [3]. Clear
- [4]. Print
- [5]. Cek antrian
- [6]. Exit

Pilihan Anda? 1

Masukkan Nama : Ririt Handayani  
Masukkan NIM : 11.01.53.11  
Masukkan Progdi : Teknik Informatika  
Data berhasil ditambahkan ke dalam antrian

```
PROGRAM QUEUE DATA MAHASISWA
MENU UTAMA:
[1]. Enque
[2]. Deque
[3]. Clear
[4]. Print
[5]. Cek antrian
[6]. Exit

Pilihan Anda? 1

Masukkan Nama : Santi Fitrinisa
Masukkan NIM : 11.01.44.11
Masukkan Progdi : Teknik Industri
Data berhasil ditambahkan ke dalam antrian
```

```
PROGRAM QUEUE DATA MAHASISWA
MENU UTAMA:
[1]. Enque
[2]. Deque
[3]. Clear
[4]. Print
[5]. Cek antrian
[6]. Exit

Pilihan Anda? 4

Data ke-1: Ririt Handayani ---> NIM 11.01.53.11 prodi Teknik Informatika
Data ke-2: Santi Fitrinisa ---> NIM 11.01.43.22 prodi Teknik Industri
```

## DAFTAR PUSTAKA

Kelley, Al and Pohl, Ira. 2003. *C by Dissection: The Essentials of C Programming*. Addison-Weasley.

Khannedy, Eko Kurniawan. 2007. *Diktat Pemrograman C*. Bandung : Unikom.

Liem, Ingriani. 2003. *Catatan Singkat Bahasa C*. Jurusan Teknik Informatika. Institut Teknologi Bandung.

Reema Thareja. 2014. *Data Structures Using C*. India : Oxford University Press.

Solichin, Achmad. 2003. *Pemrograman Bahasa C dengan Turbo C*. Kuliah Berseri ilmukomputer.com