LAPORAN PRAKTIKUM

Modul 8

“QUEUE”

Disusun Oleh:

RifqiMRamdani -2311104044

SE-07-02

Dosen :

Wahyu Andi Saputra, S.pd,M.Eng,

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

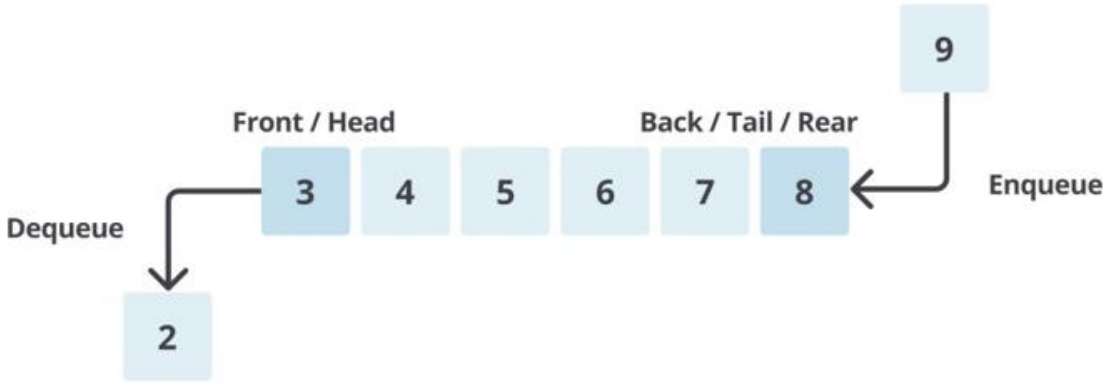
1. Tujuan

a. Mahasiswa mampu menjelaskan definisi dan konsep dari queue
b. Mahasiswa mampu menerapkan operasi tambah, menghapus pada queue
c. Mahasiswa mampu menerapkan operasi tampil data pada queue

2. Landasan Teori

Queue adalah struktur data yang digunakan untuk menyimpan data dengan metode FIFO (First-In First-Out). Data yang pertama dimasukkan ke dalam queue akan menjadi data yang pertama pula untuk dikeluarkan dari queue. Queue mirip dengan konsep antrian pada kehidupan sehari-hari, dimana konsumen yang datang lebih dulu akan dilayani terlebih dahulu.

Implementasi queue dapat dilakukan dengan menggunakan array atau linked list. Struktur data queue terdiri dari dua pointer yaitu front dan rear. Front/head adalah pointer ke elemen pertama dalam queue dan rear/tail/back adalah pointer ke elemen terakhir dalam queue.



FIRST IN FIRST OUT (FIFO)

Perbedaan antara *stack* dan *queue* terletak pada aturan penambahan dan penghapusan elemen. Pada *stack*, operasi penambahan dan penghapusan elemen dilakukan di satu ujung yang disebut *top* (ujung atas). Elemen yang terakhir kali dimasukkan ke dalam *stack* akan berada di posisi paling atas dan akan menjadi elemen pertama yang dihapus. Sifat ini dikenal dengan istilah *LIFO* (Last In, First Out). Contoh analogi sederhana dari *stack* adalah tumpukan piring, di mana piring terakhir yang

ditambahkan berada di posisi paling atas dan akan diambil atau dihapus terlebih dahulu

Sebaliknya, pada *queue*, operasi penambahan dan penghapusan elemen dilakukan di dua ujung yang berbeda. Elemen baru ditambahkan di ujung belakang (*rear* atau *tail*), dan elemen dihapus dari ujung depan (*front* atau *head*). Proses ini mengikuti prinsip *FIFO* (First In, First Out), yang berarti elemen pertama yang dimasukkan ke dalam *queue* akan menjadi elemen pertama yang dikeluarkan. Dalam konteks *queue*, operasi penambahan elemen dikenal sebagai ***Enqueue***, dan operasi penghapusan elemen disebut ***Dequeue***.

Pada *Enqueue*, elemen ditambahkan di belakang *queue* setelah elemen terakhir yang ada, sementara pada *Dequeue*, elemen paling depan (*head*) dihapus, dan posisi *head* akan bergeser ke elemen berikutnya. Contoh penggunaan *queue* dalam kehidupan sehari-hari adalah antrean di kasir, di mana orang pertama yang datang adalah yang pertama dilayani

Operasi pada Queue

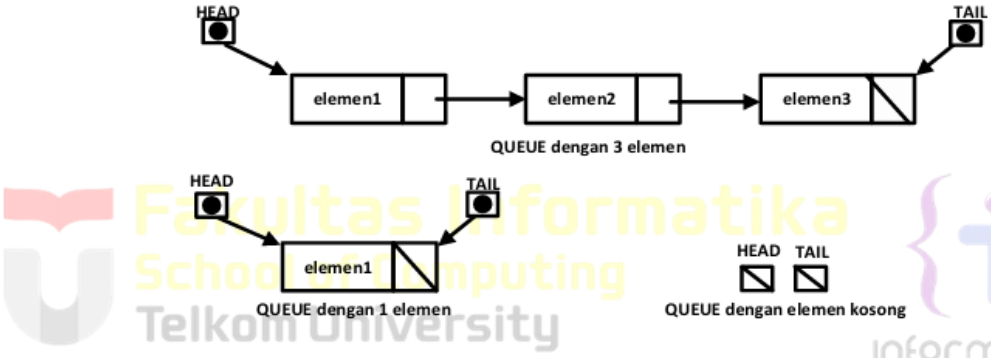
- enqueue() : menambahkan data ke dalam queue.
- dequeue() : mengeluarkan data dari queue.
- peek() : mengambil data dari queue tanpa menghapusnya.
- isEmpty() : mengecek apakah queue kosong atau tidak.
- isFull() : mengecek apakah queue penuh atau tidak.
- size() : menghitung jumlah elemen dalam queue.

3. Guided

Pengertian Queue

Queue (dibaca : kyu) merupakan struktur data yang dapat diumpamakan seperti sebuah antrian. Misalkan antrian pada loket pembelian tiket Kereta Api. Orang yang akan mendapatkan pelayanan yang pertama adalah orang pertamakali masuk dalam antrian tersebut dan yang terakhir masuk dia akan mendapatkan layanan yang terakhir pula. Jadi prinsip dasar dalam Queue adalah FIFO (First in Fisrt out), proses yang pertama masuk akan diakses terlebih dahulu. Dalam pengimplementasian struktur Queue dalam C dapat menggunakan tipe data array dan linked list. Dalam praktikum ini hanya akan dibahas pengimplementasian Queue dalam bentuk linked list. Implementasi Queue dalam linked list sebenarnya tidak jauh berbeda dengan operasi list biasa, malahan lebih sederhana. Karena sesuai dengan sifat FIFO dimana

proses delete hanya dilakukan pada bagian Head (depan list) dan proses insert selalu dilakukan pada bagian Tail (belakang list) atau sebaliknya, tergantung dari persepsi masing-masing. Dalam penerapannya Queue dapat diterapkan dalam single linked list dan double linked list.



Gambar 8-1 *Queue*

Contoh pendeklarasian struktur data *queue*:

```
1   #ifndef queue_H
2   #define queue_H
3   #define Nil NULL
4   #define info(P) (P)->info
5   #define next(P) (P)->next
6   #define head(Q) ((Q).head)
7   #define tail(Q) ((Q).tail)
8
9   typedef int infotype; /*tipe data dalam queue */
10  typedef struct elmQueue *address; /* tipe data pointer untuk elemen queue */
11  struct elmQueue{
12      infotype info;
13      address next;
14  }; /*tipe data elemen queue */
15  /* pendeklarasian tipe data queue */
16  struct queue {
17      address head, tail;
18  };
19  #endif
```

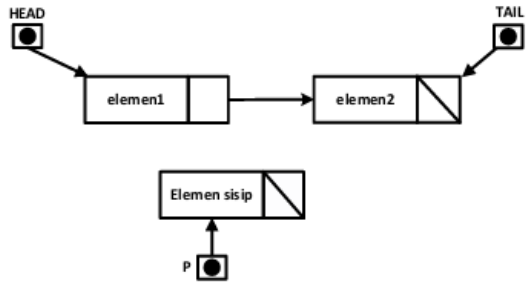
Operasi-Operasi dalam Queue

Dalam queue ada dua operasi utama, yaitu operasi penyisipan (Insert/Enqueue) dan operasi pengambilan (Delete/Dequeue).

Insert (Enqueue)

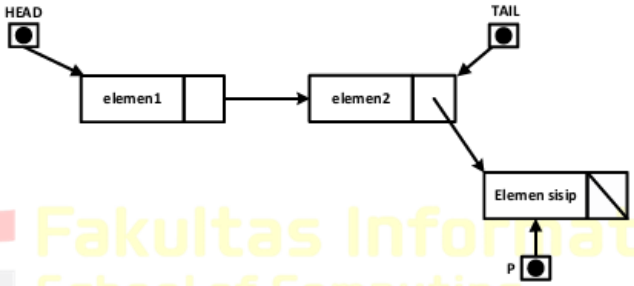
Operasi penyisipan selalu dilakukan pada akhir (*tail*).

Langkah – langkah dalam proses Enqueue:




```
P = alokasi(X);
next(P) = Nil;
```

Gambar 8-2 *Queue Insert* 1




```
next(tail(Q)) = P;
```

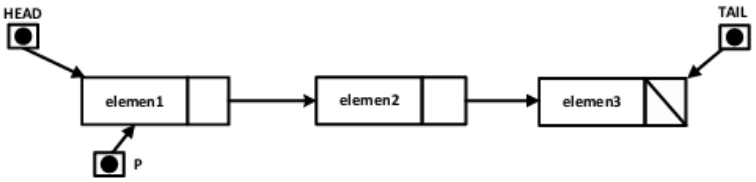
Gambar 8-3 *Queue Insert* 2


```
tail(Q) = P;
```

```
1   /* buat dahulu elemen yang akan disisipkan */
2   address createElm(int x){
3       address p = alokasi(x);
4       next(p) = null;
5       return p;
6   }
7
8   /* contoh sintak queue insert */
9   void queue(address p){
10      next(tail(Q)) = p;
11      tail(Q) = p;
12  }
```

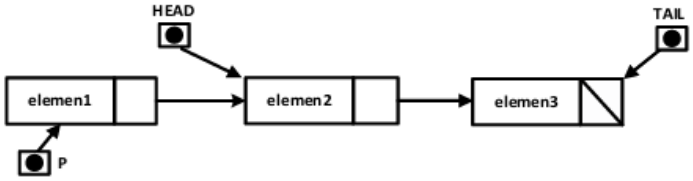
Delete (Dequeue)

Operasi delete dilakukan pada awal (head)



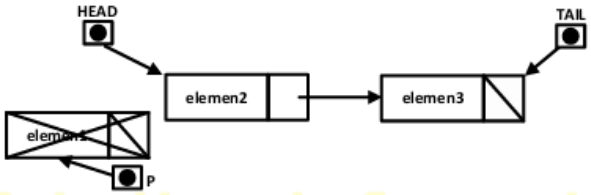

```
P = head(Q);
```

Gambar 8-5 *Queue Delete* 1




```
head(Q) = next(P);
```

Gambar 8-6 *Queue Delete* 2




```
next(P) = Nil;
return P;
```



```
1  /*contoh sintak dequeue */
2  address dequeue(address p) {
3      p = head(Q);
4      head(Q) = next(p);
5      next(p) = null;
6      return p;
7  }
```

8.3 *Primitif-Primitif dalam Queue*

Primitif-primitif pada *queue* tersimpan pada ADT *queue*, seperti pada materi sebelumnya, primitif-primitifnya tersimpan pada *file* *.h dan *.c.

File *.h untuk ADT *queue*:

```
1  /*file : queue .h
2   contoh ADT queue  dengan representasi fisik pointer
3   representasi address dengan pointer
4   info tipe adalah integer */
5
6  #ifndef QUEUE_H_INCLUDE
7  #define QUEUE_H_INCLUDE
8  #include <stdio.h>
9
10 #define Nil NULL
11 #define info(P) (P)->info
12 #define next(P) (P)->next
13 #define head(S) ((S).head)
14 #define tail(S) ((S).tail)
15 typedef int infotype;
16 typedef struct elmQ *address;
17
```

```
18  struct elmQ{
19      infotype info;
20      address next;
21  };
22  /* deklarasi tipe data queue, terdiri dari head dan tail, queue kosong jika
23      head = Nil */
24  struct queue {
25      address head, tail;
26  };
27
28  /*prototype*/
29  /********* pengecekan apakah queue kosong **********/
30  boolean isEmpty(queue Q);
31  /*mengembalikan nilai true jika queue kosong*/
32
33  /********* pembuatan queue kosong *********/
34  void CreateQueue(queue &Q);
35  /* I.S. sembarang
36     F.S. terbentuk queue kosong*/
37
38
39  /********* manajemen memori *********/
40  void alokasi(infotype X);
41  /* mengirimkan address dari alokasi elemen
42     jika alokasi berhasil, maka nilai address tidak Nil dan jika gagal, nilai
43  address Nil */
44
45  void dealokasi(address P);
46  /* I.S. P terdefinisi
47     F.S. memori yang digunakan P dikembalikan ke sistem */
48
49  /********* pencarian sebuah elemen list *********/
50  address findElm(queue Q, infotype X);
51  /* mencari apakah ada elemen queue dengan info(P) = X
52     jika ada, mengembalikan address elemen tab tsb, dan Nil jika sebaliknya */
53
54  boolean fFindElm(queue Q address P);
55  /* mencari apakah ada elemen queue dengan alamat P
56     mengembalikan true jika ada dan false jika tidak ada */
57
58  /********* penambahan elemen **********/
59  void insert(queue &Q, address P);
60  /* I.S. sembarang, P sudah dialokasikan
61     F.S. menempatkan elemen beralamat P pada akhir queue*/
62
63  /********* penghapusan sebuah elemen **********/
64  void delete(queue &Q, address &P);
65  /* I.S. queue tidak kosong
66     F.S. menunjuk elemen pertama queue, head dari queue menunjuk pada next
67      elemen head yang lama. Queue mungkin menjadi kosong */
68
```

```
68
69  /********* proses semua elemen queue**********/
70  void printInfo(queue Q);
71  /* I.S. queue mungkin tidak kosong
72     F.S. jika queue tidak kosong, maka menampilkan semua info yang ada pada
73  queue */
74
75  void nbList(queue Q);
76  /* mengembalikan jumlah elemen pada queue */
77
78  void delAll(queue &Q);
79  /* menghapus semua elemen pada queue dan semua elemen di-dealokasi */
80
81  #endif
```

Queue (Representasi Tabel)

Pengertian Pada dasarnya representasi queue menggunakan tabel sama halnya dengan menggunakan pointer. Perbedaan yang mendasar adalah pada management memori serta keterbatasan jumlah antriannya. Untuk lebih jelasnya perhatikan perbedaan representasi tabel dan pointer pada queue dibawah ini.

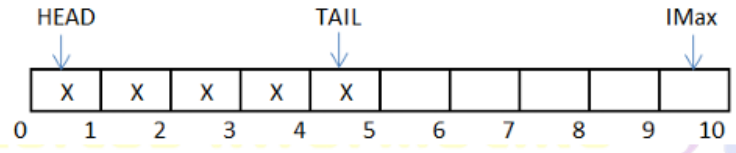
Tabel 8-1 Perbedaan *Queue* Representasi Table dan *Pointer*

No	Representasi Table	Representasi *Pointer*
1	Jumlah *Queue* terbatas	Jumlah *Queue* tak-terbatas
2	Tidak ada management memori	ada management memori

A. Alternatif 1

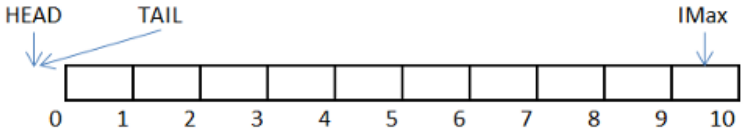
Tabel dengan hanya representasi *TAIL* adalah indeks elemen terakhir, *HEAD* selalu di-*set* sama dengan 1 jika *Queue* tidak kosong. Jika *Queue* kosong, maka *HEAD*=0.

Ilustrasi *Queue* tidak kosong, dengan 5 elemen :



Gambar 8-8 *Queue* 1 Tidak Kosong Representasi Tabel

Ilustrasi *Queue* kosong:

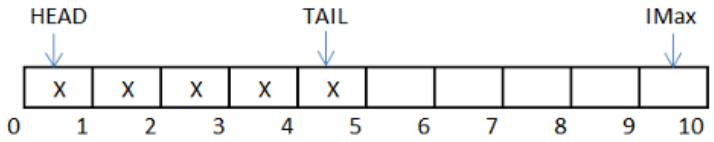


Gambar 8-9 *Queue* 1 Kosong Representasi Tabel

Algoritma paling sederhana untuk penambahan elemen jika masih ada tempat adalah dengan “memajukan” TAIL. Kasus khusus untuk Queue kosong karena HEAD harus diset nilainya menjadi 1. Algoritma paling sederhana dan “naif” untuk penghapusan elemen jika Queue tidak kosong: ambil nilai elemen HEAD, geser semua elemen mulai dari HEAD+1 s/d TAIL (jika ada), kemudian TAIL “mundur”. Kasus khusus untuk Queue dengan keadaan awal berelemen 1, yaitu menyesuaikan HEAD dan TAIL dengan DEFINISI. Algoritma ini mencerminkan pergeseran orang yang sedang mengantri di dunia nyata, tapi tidak efisien.

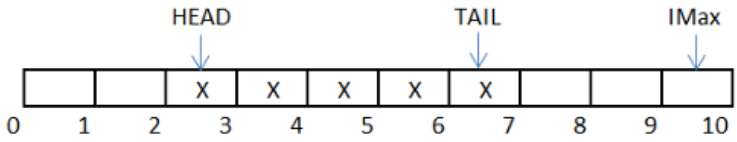
B. Alternatif 2

Tabel dengan representasi HEAD dan TAIL, HEAD “bergerak” ketika sebuah elemen dihapus jika Queue tidak kosong. Jika Queue kosong, maka HEAD=0. Ilustrasi Queue tidak kosong, dengan 5 elemen, kemungkinan pertama HEAD“sedang berada di posisi awal:



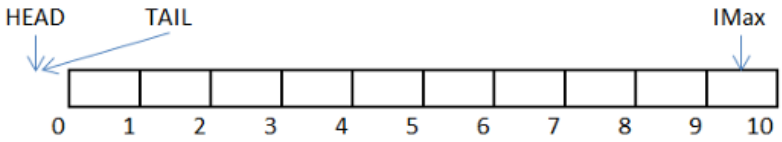
Gambar 8-10 *Queue* 2 Tidak Kosong 1 Representasi Table

Ilustrasi *Queue* tidak kosong, dengan 5 elemen, kemungkinan pertama *HEAD* tidak berada di posisi awal. Hal ini terjadi akibat algoritma penghapusan yang dilakukan (lihat keterangan).



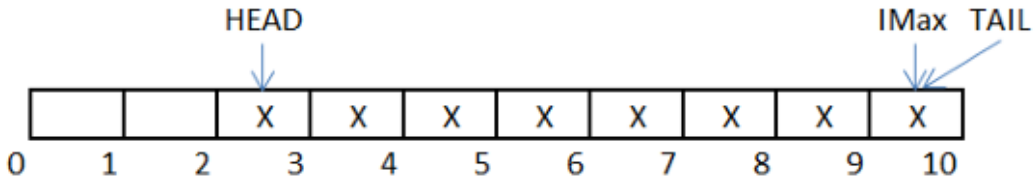
Gambar 8-11 *Queue* 2 Tidak Kosong 2 Representasi Tabel

Ilustrasi *Queue* kosong:



Gambar 8-12 *Queue* 2 Kosong Representasi Tabel

Algoritma untuk penambahan elemen sama dengan alternatif I: jika masih ada tempat adalah dengan "memajukan" TAIL. Kasus khusus untuk Queue kosong karena HEAD harus diset nilainya menjadi 1. Algoritmanya sama dengan alternatif I. Algoritma untuk penghapusan elemen jika Queue tidak kosong: ambil nilai elemen HEAD, kemudian HEAD "maju". Kasus khusus untuk Queue dengan keadaan awal berelemen 1, yaitu menyesuaikan HEAD dan TAIL dengan DEFINISI. Algoritma ini TIDAK mencerminkan pergeseran orang yang sedang mengantri di dunia nyata, tapi efisien. Namun suatu saat terjadi keadaan Queue penuh tetapi "semu" sebagai berikut :

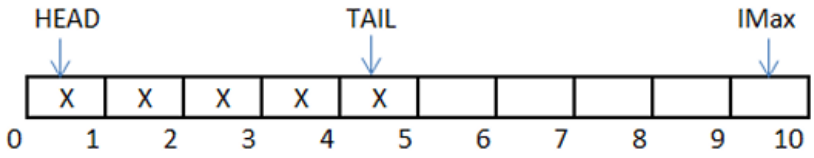


Gambar 8-13 *Queue* 2 Penuh "semu"

Jika keadaan ini terjadi, haruslah dilakukan aksi menggeser elemen untuk menciptakan ruangan kosong. Pergeseran hanya dilakukan jika dan hanya jika TAIL sudah mencapai IndexMax.

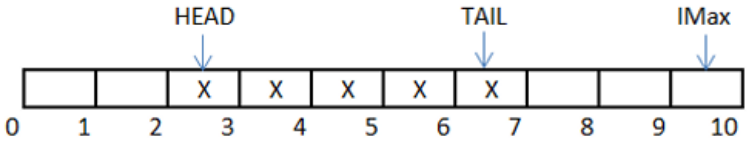
C. Alternatif 3

Tabel dengan representasi HEAD dan TAIL yang “berputar” mengelilingi indeks tabel dari awal sampai akhir, kemudian kembali ke awal. Jika Queue kosong, maka HEAD=0. Representasi ini memungkinkan tidak perlu lagi ada pergeseran yang harus dilakukan seperti pada alternatif II pada saat penambahan elemen. Ilustrasi Queue tidak kosong, dengan 5 elemen, dengan HEAD “sedang” berada di posisi awal:



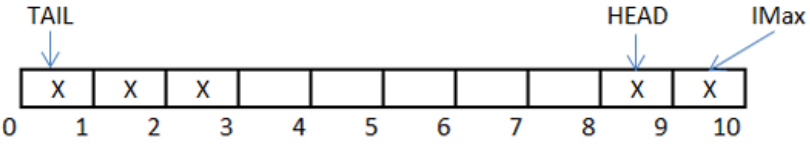
Gambar 8-14 *Queue* 3 Tidak Kosong 1 Representasi Table

Ilustrasi *Queue* tidak kosong, dengan 5 elemen, dengan *HEAD* tidak berada diposisi awal, tetapi masih “lebih kecil” atau “sebelum” *TAIL*. Hal ini terjadi akibat algoritma penghapusan/Penambahan yang dilakukan (lihat keterangan).



Gambar 8-15 *Queue* 3 Tidak Kosong 2 Representasi Table

Ilustrasi *Queue* tidak kosong, dengan 5 elemen, *HEAD* tidak berada di posisiawal, tetapi “lebih besar” atau “sesudah” *TAIL*. Hal ini terjadi akibat algoritma penghapusan/Penambahan yang dilakukan (lihat keterangan)



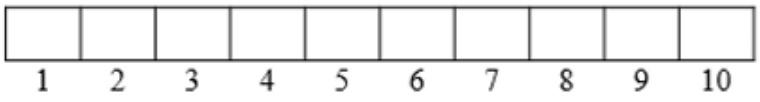
Gambar 8-16 *Queue* 3 Tidak Kosong 3 Representasi Table

Algoritma untuk penambahan elemen: jika masih ada tempat adalah dengan“memajukan” TAIL. Tapi jika TAIL sudah mencapai IdxMax, maka successor dari IdxMax adalah 1 sehingga TAIL yang baru adalah 1. Jika TAIL belum mencapai IdxMax, maka algoritma penambahan elemen sama dengan alternatif II. Kasus khusus untuk Queue kosong karena HEAD harus diset nilainya menjadi 1. Algoritma untuk penghapusan elemen jika Queue tidak kosong: ambil nilai elemen HEAD, kemudian HEAD “maju”. Penentuan suatu successor dari indeks yang diubah/”maju” dibuat Seperti pada algoritma penambahan elemen: jika HEAD mencapai IdxMAx, maka successor dari HEAD adalah 1. Kasus khusus untuk Queue dengan keadaan awal berelemen 1, yaitu menyesuaikan HEAD dan TAIL dengan DEFINISI. Algoritma ini efisien karena tidak perlu pergeseran, dan seringkali strategi pemakaian tabel semacam ini disebut sebagai “circular buffer”, dimana tabel penyimpan elemen dianggap sebagai “buffer”. Salah satu variasi dari representasi pada alternatif III adalah : menggantikan representasi TAIL dengan COUNT, yaitu

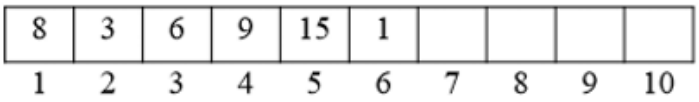
banyaknya elemen Queue. Dengan representasi ini, banyaknya elemen diketahui secara eksplisit, tetapi untuk melakukan penambahan elemen harus dilakukan kalkulasi TAIL. Buatlah sebagai latihan.

Contoh Kasus Alternatif 2

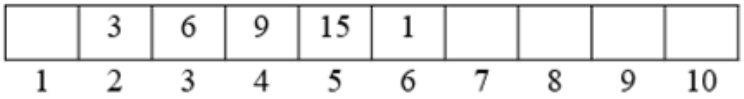
Antrian kosong (*Head*=0, *Tail*=0)



Setelah penambahan elemen 8,3,6,9,15,1 (*Head*=1, *Tail*=6)



Setelah penghapusan sebuah elemen (*Head*=2, *Tail*=6)



Algoritma	C++

Kamus Umum
constant NMax: integer = 100
constant Nil: integer = 0
type infotype = integer
type Queue = <TabQ: array[1..NMax] of infotype;
Head,Tail: integer>
Function EmptyQ(Q:Queue) → boolean
{True jika Q antrian kosong}
Procedure CreateEmpty(output Q:Queue)
{I.Q. -
F.Q. Terdefinisi antrian kosong Q}
	const int NMax=100;
	const int Nil=0;
	typedef int infotype queue {
	infotype TabQ[NMax];
	int head, tail;
	}
	
	bool Empty(Queue Q){
	/* True jika Q merupakan antrian kosong */
	}
	
	void CreateEmpty(){
	/* membuat antrian kosong */
	}

function IsFullQ(Q:Queue) → boolean
{True Q penuh}
kamus:
algoritma
→ (Q.Head = 1) and (Q.Tail=NMax)
	bool isFull(Queue Q){
	/* True jika Q penuh */
	if(head(Q)=1 && tail(Q)=Nmax){
	return true;
	}
	}

Procedure AddQ(input/output Q:Queue; input X:infotype)
{I.Q. Q antrian, terdefinisi, mungkin kosong}
{F.Q. X elemen terakhir Q jika antrian tidak penuh}
Kamus
i:integer
Algoritma
if IsFull(Q) then
output('Antrian penuh')
else
if Empty(Q) then
Q.Head ← 1
Q.Tail ← 1
else
if Q.Tail<NMax then
Q.Tail ← Q.Tail + 1
else
for i←Q.Head to Q.Tail do
Q.TabQ[i-Q.Head+1] ← Q.TabQ[i]
Q.Tail ← Q.Tail-Q.Head+2
Q.Head ← 1
Q.TabQ[Q.Tail] ← X *(menyisipkan X)*
	void addQ(Queue Q, infotype x){
	int i;
	if (isFull(Q)){
	cout >> "Antrian Penuh";
	}else{
	if(Empty(Q)){
	head(Q) = 1;
	tail(Q) = 1;
	}else{
	if(tail(Q)>NMax){
	tail(Q)++;
	}else{
	for(i=head(Q);i<=tail(Q);i++){
	TabQ[i+head(Q)+1](Q)=
	TabQ[i](Q);
	}
	tail(Q) = tail(Q)-head(Q)+2;
	head(Q) = 1;
	TabQ[tail(Q)](Q) = x;
	/*menyisipkan x */
	}
	}
	}
	}

Procedure DelQ(input/output Q:Queue; output X:infotype)
{I.Q. Q antrian, terdefinisi, tidak kosong}
{F.Q. X elemen antrian yang dihapus}
Kamus
i:integer
Algoritma
X ← Q.TabQ[Q.Head]
if Q.Head=Q.Tail then
CreateEmpty(Q)
else
Q.Head ← Q.Head+1
	void DelQ(Queue Q, infotype x){
	int i;
	x= TabQ[head(Q)](Q);
	if(head(Q)==tail(Q)){
	CreateEmpty(Q);
	}else{
	head(Q);
	}
	}

Primitif-Primitif dalam Queue

Berikut ini contoh program queue.h dalam ADT queue menggunakan representasi table.

```
1  /*file : queue .h
2   contoh ADT queue dengan representasi tabel*/
3
4  #ifndef QUEUE_H_INCLUDE
5  #define QUEUE_H_INCLUDE
6  #include <stdio.h>
7  #include <conio.h>
8
9  struct infotype {
10     char id[20];
11     char nama[20];
12 };
13
14 struct Queue {
15     infotype info[3];
16     int head;
17     int tail;
18 };
19
20 /* prototype */
21 /******** pengecekan apakah Queue penuh *********/
22 int isFull(queue Q):
23 /* mengembalikan nilai 0 jika queue penuh */
24
25 /******** pengecekan apakah Queue kosong ********/
26 int isEmpty(queue Q);
27 /* mengembalikan nilai 0 jika queue kosong */
28
29 /********* pembuatan queue *********/
30 void CreateQueue(queue &Q);
31 /* I.S. sembarang
32    F.S. terbentuk queue dengan head = -1 dan tail = -1 */
33
34 /********* penambahan elemen pada queue *********/
35 void enQueue(queue &Q, infotype X);
36 /* I.S. queue mungkin kosong
37    F.S. menambahkan elemen pada stack dengan nilai X */
38
39 /********* penghapusan elemen pada queue *********/
40 void deQueue(queue &Q);
41 /* I.S. queue tidak kosong
42    F.S. head = head + 1 */
43
44 /********* proses semua elemen pada queue *********/
45 void viewQueue(queue Q);
46 /* I.S. queue mungkin kosong
47    F.S. jika queue tidak kosong menampilkan semua info yang ada pada queue */
48
49 #endif
```


LATIHAN GUIDED

Kode Program

1

main.cpp

```cpp
#include <iostream>
#define MAX 100

using namespace std;

class Queue {
private:
    int front, rear;
    int arr[MAX];
public:

    Queue() {
        front = -1;
        rear = -1;
    }


    bool isFull() {
        return rear == MAX - 1;
    }


    bool isEmpty() {
        return front == -1 || front > rear;
    }


    void enqueue(int x) {
        if (isFull()) {
            cout << "Queue Overflow\n";
            return;
        }
        if (front == -1) front = 0;
        arr[++rear] = x;
    }


    void dequeue() {
        if (isEmpty()) {
            cout << "Queue Underflow\n";
            return;
        }
        front++;
    }
```

```
    }

    int peek() {
        if (!isEmpty()) {
            return arr[front];
        }
        cout << "Queue is empty\n";
        return -1;
    }


    void display() {
        if (isEmpty()) {
            cout << "Queue is empty\n";
            return;
        }
        for (int i = front; i <= rear; i++) {
            cout << arr[i] << " ";
        }
        cout << "\n";
    }
};


int main() {
    Queue q;

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);

    cout << "Queue elements: ";
    q.display();

    cout << "Front element: " << q.peek() << "\n";

    cout << "After dequeuing, queue elements: ";
    q.display();

    return 0;
}
```

Maka akan menghasilkan output

```
"D:\TUGAS SEMESTER 3\GUID
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 10 20 30

Process returned 0 (0x0)   execution time : 0.328 s
Press any key to continue.
```

2

```cpp
#include <iostream>

using namespace std;

// Node untuk setiap elemen Queue
class Node {
public:
    int data;       // Data elemen
    Node* next;     // Pointer ke node berikutnya

    // Konstruktor untuk Node
    Node(int value) {
        data = value;
        next = nullptr;
    }
};

// Kelas Queue menggunakan linked list
class Queue {
private:
    Node* front; // Pointer ke elemen depan Queue
    Node* rear;  // Pointer ke elemen belakang Queue

public:
    // Konstruktor Queue
    Queue() {
        front = rear = nullptr;
    }

    // Mengecek apakah Queue kosong
    bool isEmpty() {
        return front == nullptr;
    }

    // Menambahkan elemen ke Queue
    void enqueue(int x) {
        Node* newNode = new Node(x);
```

main.cpp main.cpp

```
37          Node* newNode = new Node(x);
38          if (isEmpty()) {
39              front = rear = newNode; // Jika Queue kosong
40              return;
41          }
42          rear->next = newNode; // Tambahkan node baru ke belakang
43          rear = newNode;       // Perbarui rear
44      }
45
46      // Menghapus elemen dari depan Queue
47      void dequeue() {
48          if (isEmpty()) {
49             cout << "Queue Underflow\n";
50              return;
51          }
52          Node* temp = front;       // Simpan node depan untuk dihapus
53          front = front->next;      // Pindahkan front ke node berikutnya
54          delete temp;              // Hapus node lama
55          if (front == nullptr)     // Jika Queue kosong, rear juga harus null
56              rear = nullptr;
57      }
58
59      // Mengembalikan elemen depan Queue tanpa menghapusnya
60      int peek() {
61          if (!isEmpty()) {
62              return front->data;
63          }
64         cout << "Queue is empty\n";
65          return -1; // Nilai sentinel
66      }
67
68      // Menampilkan semua elemen di Queue
69      void display() {
70          if (isEmpty()) {
71             cout << "Queue is empty\n";
72              return;
73          }
```

main.cpp main.cpp

```
70          if (isEmpty()) {
71              cout << "Queue is empty\n";
72              return;
73          }
74          Node* current = front; // Mulai dari depan
75          while (current) {      // Iterasi sampai akhir
76              cout << current->data << " ";
77              current = current->next;
78          }
79          cout << "\n";
80      }
81  };
82
83  // Fungsi utama untuk menguji Queue
84  int main() {
85      Queue q;
86
87      // Menambahkan elemen ke Queue
88      q.enqueue(10);
89      q.enqueue(20);
90      q.enqueue(30);
91
92      // Menampilkan elemen di Queue
93      cout << "Queue elements: ";
94      q.display();
95
96      // Menampilkan elemen depan
97      cout << "Front element: " << q.peek() << "\n";
98
99      // Menghapus elemen dari depan Queue
100     q.dequeue();
101     cout << "After dequeuing, queue elements: ";
102     q.display();
103
104     return 0;
105 }
106
```

Maka outputnya

```
"D:\TUGAS SEMESTER 3\GUID
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 20 30

Process returned 0 (0x0)   execution time : 0.102 s
Press any key to continue.
```

main.cpp main.cpp main.cpp

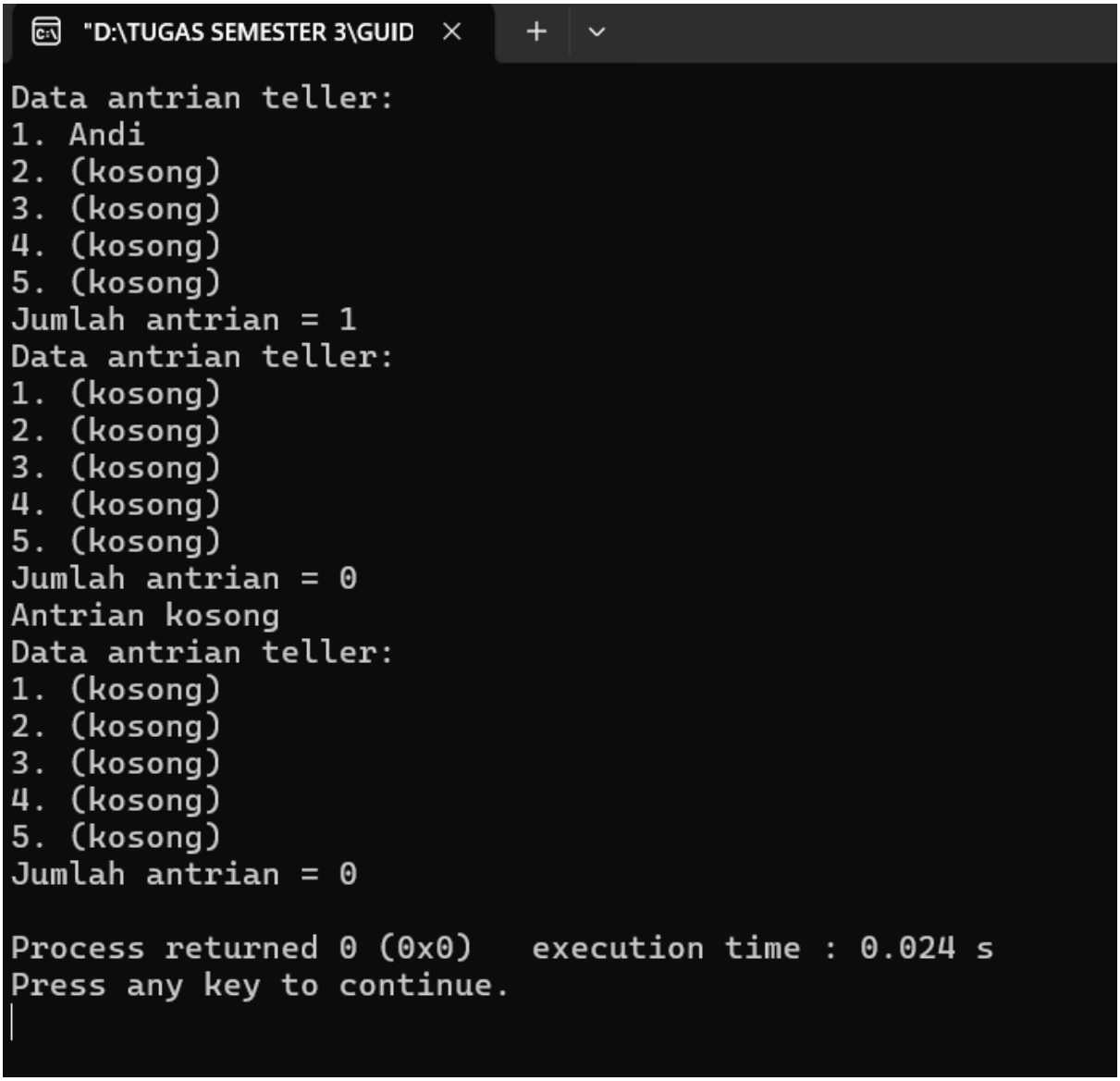
```cpp
#include<iostream>

using namespace std;

const int maksimalQueue = 5; // Maksimal antrian
int front = 0; // Penanda antrian
int back = 0; // Penanda
string queueTeller[5]; // Fungsi pengecekan

bool isFull() { // Pengecekan antrian penuh atau tidak
if (back == maksimalQueue) { return true; // =1
} else {
return false;
}
}

bool isEmpty() { // Antriannya kosong atau tidak
if (back == 0) { return true;
} else {
return false;
}
}

void enqueueAntrian(string data) { // Fungsi menambahkan antrian
if (isFull()) {
cout << "Antrian penuh" << endl;
} else {
if (isEmpty()) { // Kondisi ketika queue kosong
queueTeller[0] = data; front++;
back++;
} else { // Antrianya ada isi queueTeller[back] = data; back++;
}
}
}

void dequeueAntrian() { // Fungsi mengurangi antrian
if (isEmpty()) {
```

main.cpp main.cpp main.cpp

```
37  if (isEmpty()) {
38  cout << "Antrian kosong" << endl;
39  } else {
40  for (int i = 0; i < back; i++) { queueTeller[i] = queueTeller[i + 1];
41  }
42  back--;
43  }
44  }
45
46  int countQueue() { // Fungsi menghitung banyak antrian
47  return back;
48  }
49
50  void clearQueue() { // Fungsi menghapus semua antrian
51  if (isEmpty()) {
52  cout << "Antrian kosong" << endl;
53  } else {
54  for (int i = 0; i < back; i++) { queueTeller[i] = "";
55  }
56  back = 0;
57  front = 0;
58  }
59  }
60
61  void viewQueue() { // Fungsi melihat antrian
62  cout << "Data antrian teller:" << endl; for (int i = 0; i < maksimalQueue; i++) {
63  if (queueTeller[i] != "") {
64  cout << i + 1 << ". " << queueTeller[i] <<
65
66  endl;
67
68
69  } else {
70  cout << i + 1 << ". (kosong)" << endl;
71
72  }
73  }
```

```
72  }
73  }
74  }
75
76  int main() {
77  enqueueAntrian("Andi");
78
79  enqueueAntrian("Maya");
80
81  viewQueue();
82  cout << "Jumlah antrian = " << countQueue() << endl;
83
84  dequeueAntrian();
85  viewQueue();
86  cout << "Jumlah antrian = " << countQueue() << endl;
87
88  clearQueue();
89  viewQueue();
90  cout << "Jumlah antrian = " << countQueue() << endl;
91
92  return 0;
93  }
94
```

Maka akan menghasilkan output



4. Unguided

a. Ubahlah penerapan konsep queue pada bagian guided dari array menjadilinked list

JAWAB

Kode Program:

*main.cpp

```cpp
#include <iostream>
#include <string>
using namespace std;

struct Node {
    string data;
    Node* next;
};

class Queue {
private:
    Node* front;
    Node* back;

public:

    Queue() {
        front = nullptr;
        back = nullptr;
    }

    bool isEmpty() {
        return front == nullptr;
    }

    void enqueue(string data) {
        Node* newNode = new Node;
        newNode->data = data;
        newNode->next = nullptr;

        if (isEmpty()) {
            front = newNode;
            back = newNode;
        } else {
            back->next = newNode;
            back = newNode;
        }
```

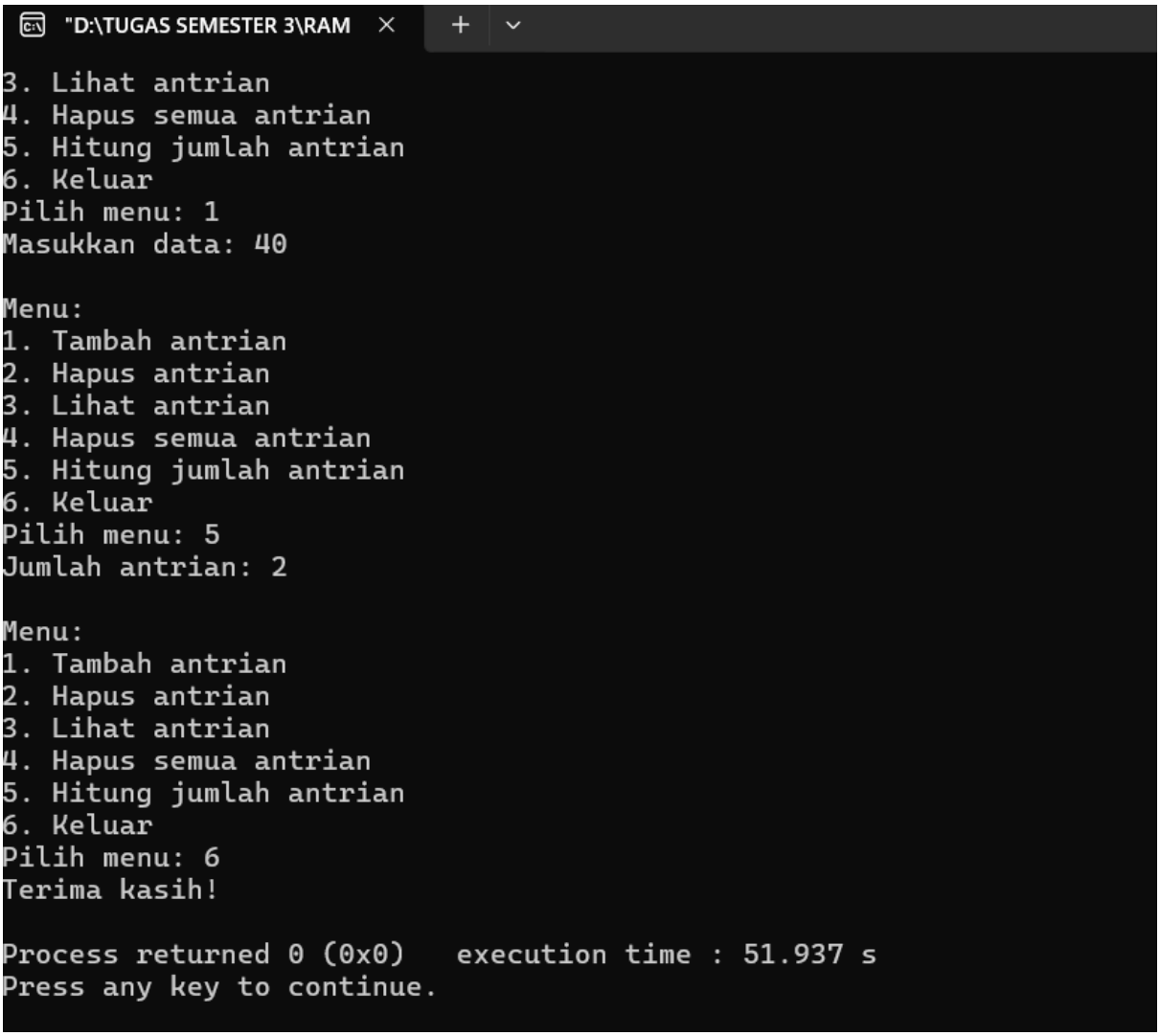
*main.cpp

```
37          }
38      }
39
40      void dequeue() {
41          if (isEmpty()) {
42              cout << "Antrian kosong" << endl;
43          } else {
44              Node* temp = front;
45              front = front->next;
46              delete temp;
47              if (front == nullptr) {
48                  back = nullptr;
49              }
50          }
51      }
52
53      void viewQueue() {
54          if (isEmpty()) {
55              cout << "Antrian kosong" << endl;
56          } else {
57              Node* current = front;
58              cout << "Data antrian:" << endl;
59              int position = 1;
60              while (current != nullptr) {
61                  cout << position << ". " << current->data << endl;
62                  current = current->next;
63                  position++;
64              }
65          }
66      }
67
68      int countQueue() {
69          int count = 0;
70          Node* current = front;
71          while (current != nullptr) {
72              count++;
73              current = current->next;
```

*main.cpp

```
            current = current->next;
        }
        return count;
    }

    void clearQueue() {
        while (!isEmpty()) {
            dequeue();
        }
        cout << "Antrian telah dikosongkan." << endl;
    }
};

int main() {
    Queue q;
    int pilihan;
    string data;

    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Tambah antrian\n";
        cout << "2. Hapus antrian\n";
        cout << "3. Lihat antrian\n";
        cout << "4. Hapus semua antrian\n";
        cout << "5. Hitung jumlah antrian\n";
        cout << "6. Keluar\n";
        cout << "Pilih menu: ";
        cin >> pilihan;
        cin.ignore();

        switch (pilihan) {
            case 1:
                cout << "Masukkan data: ";
                getline(cin, data);
                q.enqueue(data);
                break;
            case 2:
                q.dequeue();
                break;
            case 3:
                q.viewQueue();
                break;
            case 4:
                q.clearQueue();
                break;
            case 5:
                cout << "Jumlah antrian: " << q.countQueue() << endl;
                break;
            case 6:
                cout << "Terima kasih!" << endl;
                return 0;
            default:
                cout << "Pilihan tidak valid! Silakan coba lagi." << endl;
        }
    }
}
```

Maka Akan menghasilkan output



2

Dari nomor 1 buatlah konsep antri dengan atribut Nama mahasiswa dan NIM Mahasiswa

Kode Program:

main.cpp *main.cpp

```cpp
#include <iostream>
#include <string>
using namespace std;

struct Mahasiswa {
    string nama;
    string nim;
    Mahasiswa* next;
};

class Queue {
private:
    Mahasiswa* front;
    Mahasiswa* back;

public:

    Queue() {
        front = nullptr;
        back = nullptr;
    }

    bool isEmpty() {
        return front == nullptr;
    }

    void enqueue(string nama, string nim) {
        Mahasiswa* newNode = new Mahasiswa;
        newNode->nama = nama;
        newNode->nim = nim;
        newNode->next = nullptr;

        if (isEmpty()) {
            front = newNode;
            back = newNode;
        } else {
            back->next = newNode;
```

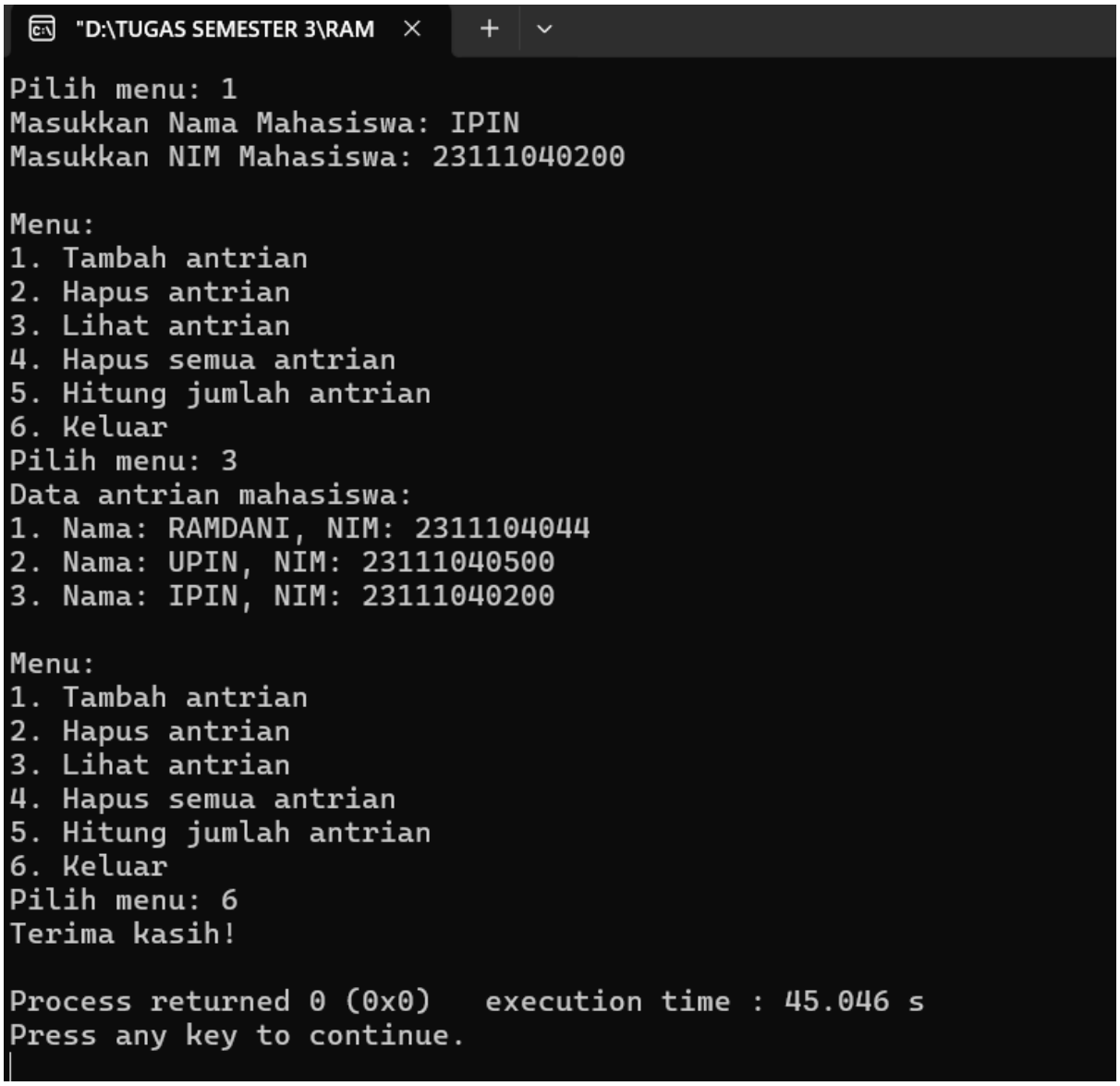
main.cpp | *main.cpp

```
            back->next = newNode;
            back = newNode;
        }
    }

    void dequeue() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
        } else {
            Mahasiswa* temp = front;
            front = front->next;
            delete temp;
            if (front == nullptr) {
                back = nullptr;
            }
        }
    }

    void viewQueue() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
        } else {
            Mahasiswa* current = front;
            cout << "Data antrian mahasiswa:" << endl;
            int position = 1;
            while (current != nullptr) {
                cout << position << ". Nama: " << current->nama << ", NIM: " << current->nim << endl;
                current = current->next;
                position++;
            }
        }
    }

    int countQueue() {
        int count = 0;
        Mahasiswa* current = front;
        while (current != nullptr) {
```

main.cpp | *main.cpp

```
        while (current != nullptr) {
            count++;
            current = current->next;
        }
        return count;
    }

    void clearQueue() {
        while (!isEmpty()) {
            dequeue();
        }
        cout << "Antrian telah dikosongkan." << endl;
    }
};

int main() {
    Queue q;
    int pilihan;
    string nama, nim;

    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Tambah antrian\n";
        cout << "2. Hapus antrian\n";
        cout << "3. Lihat antrian\n";
        cout << "4. Hapus semua antrian\n";
        cout << "5. Hitung jumlah antrian\n";
        cout << "6. Keluar\n";
        cout << "Pilih menu: ";
        cin >> pilihan;
        cin.ignore();

        switch (pilihan) {
            case 1:
                cout << "Masukkan Nama Mahasiswa: ";
                getline(cin, nama);
                cout << "Masukkan NIM Mahasiswa: ";
                cout << "Masukkan NIM Mahasiswa: ";
                getline(cin, nim);
                q.enqueue(nama, nim);
                break;
            case 2:
                q.dequeue();
                break;
            case 3:
                q.viewQueue();
                break;
            case 4:
                q.clearQueue();
                break;
            case 5:
                cout << "Jumlah antrian: " << q.countQueue() << endl;
                break;
            case 6:
                cout << "Terima kasih!" << endl;
                return 0;
            default:
                cout << "Pilihan tidak valid! Silakan coba lagi." << endl;
        }
    }
}
```

Maka Akan menghasilkan output



3.

Modifikasi program pada soal 1 sehingga mahasiswa dapat diprioritaskan berdasarkan NIM (NIM yang lebih kecil didahulukan pada saat output).

JAWAB

Kode Program:

main.cpp | main.cpp | *main.cpp

```cpp
#include <iostream>
#include <string>
using namespace std;

struct Mahasiswa {
    string nama;
    string nim;
    Mahasiswa* next;
};

class Queue {
private:
    Mahasiswa* front;
    Mahasiswa* back;

public:

    Queue() {
        front = nullptr;
        back = nullptr;
    }

    bool isEmpty() {
        return front == nullptr;
    }

    void enqueue(string nama, string nim) {
        Mahasiswa* newNode = new Mahasiswa;
        newNode->nama = nama;
        newNode->nim = nim;
        newNode->next = nullptr;

        if (isEmpty()) {

            front = newNode;
            back = newNode;
        } else {
```

main.cpp | main.cpp | *main.cpp

```
        } else {

            if (stoi(nim) < stoi(front->nim)) {

                newNode->next = front;
                front = newNode;
            } else {

                Mahasiswa* current = front;
                while (current->next != nullptr && stoi(nim) > stoi(current->next->nim)) {
                    current = current->next;
                }

                newNode->next = current->next;
                current->next = newNode;

                if (newNode->next == nullptr) {
                    back = newNode;
                }
            }
        }
    }

    void dequeue() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
        } else {
            Mahasiswa* temp = front;
            front = front->next;
            delete temp;
            if (front == nullptr) {
                back = nullptr;
            }
        }
    }

    void viewQueue() {
```

main.cpp | main.cpp | *main.cpp

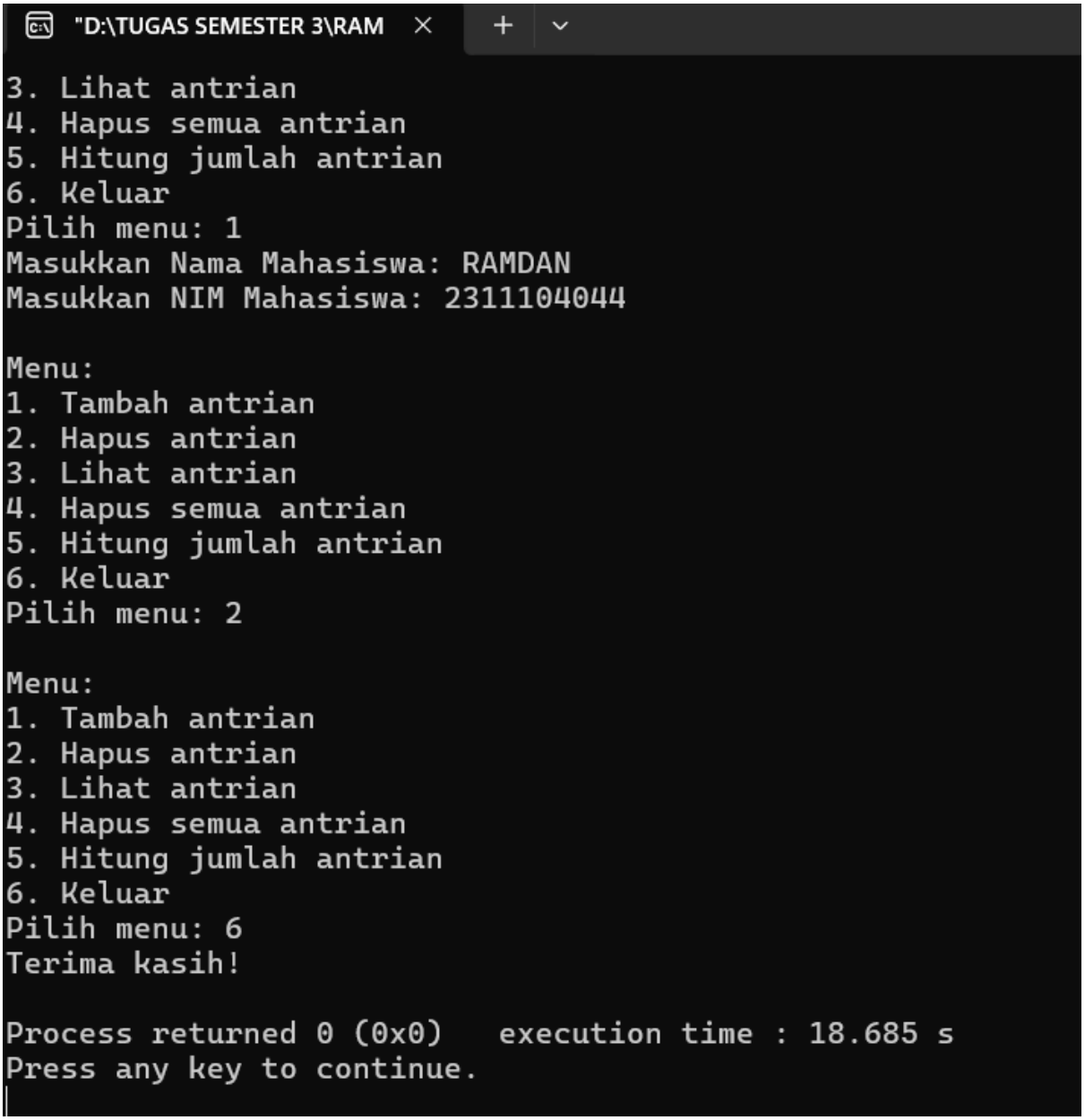
```
    void viewQueue() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
        } else {
            Mahasiswa* current = front;
            cout << "Data antrian mahasiswa:" << endl;
            int position = 1;
            while (current != nullptr) {
                cout << position << ". Nama: " << current->nama << ", NIM: " << current->nim << endl;
                current = current->next;
                position++;
            }
        }
    }

    int countQueue() {
        int count = 0;
        Mahasiswa* current = front;
        while (current != nullptr) {
            count++;
            current = current->next;
        }
        return count;
    }

    void clearQueue() {
        while (!isEmpty()) {
            dequeue();
        }
        cout << "Antrian telah dikosongkan." << endl;
    }
};

int main() {
    Queue q;
    int pilihan;
    string nama, nim;
```

main.cpp | main.cpp | *main.cpp

```
109     string nama, nim;
110
111     while (true) {
112         cout << "\nMenu:\n";
113         cout << "1. Tambah antrian\n";
114         cout << "2. Hapus antrian\n";
115         cout << "3. Lihat antrian\n";
116         cout << "4. Hapus semua antrian\n";
117         cout << "5. Hitung jumlah antrian\n";
118         cout << "6. Keluar\n";
119         cout << "Pilih menu: ";
120         cin >> pilihan;
121         cin.ignore();
122
123         switch (pilihan) {
124             case 1:
125                 cout << "Masukkan Nama Mahasiswa: ";
126                 getline(cin, nama);
127                 cout << "Masukkan NIM Mahasiswa: ";
128                 getline(cin, nim);
129                 q.enqueue(nama, nim);
130                 break;
131             case 2:
132                 q.dequeue();
133                 break;
134             case 3:
135                 q.viewQueue();
136                 break;
137             case 4:
138                 q.clearQueue();
139                 break;
140             case 5:
141                 cout << "Jumlah antrian: " << q.countQueue() << endl;
142                 break;
143             case 6:
144                 cout << "Terima kasih!" << endl;
145                 return 0;
```

```
145                 return 0;
146             default:
147                 cout << "Pilihan tidak valid! Silakan coba lagi." << endl;
148         }
149     }
150 }
151
```

Maka akan menghasilkan output



5. Kesimpulan