# Antrian(Queue)

BAB 4

# Pokok Bahasan

4.1 Karakteristik Queue

4.2 Representasi Antrian

4.2.1 Implementasi Antrian dengan Array

4.2.2 Implementasi Antrian dengan Linked list

4.3 Antrian Berprioritas

4.4 Kesimpulan

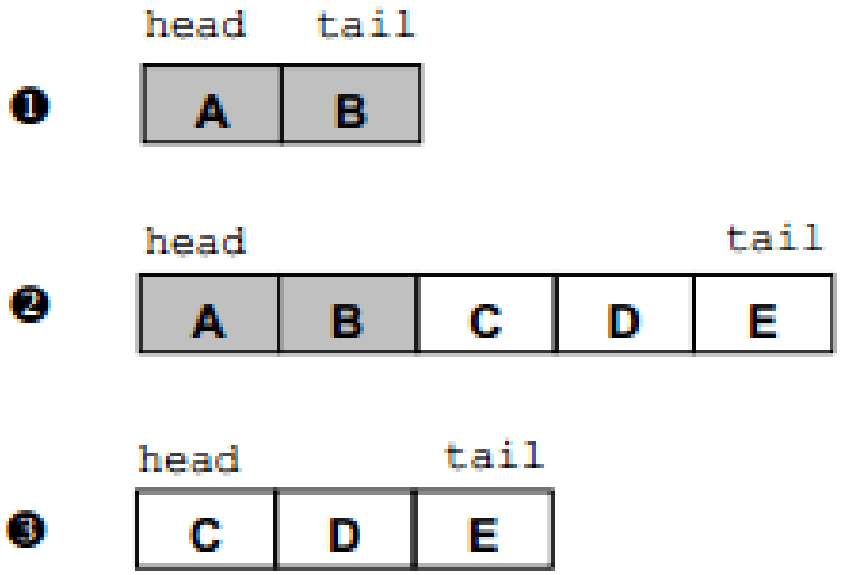
# Pengertian Queue

Karakteristik yang membedakan queue (antrian) dari stack adalah cara menyimpan dan mengambil data dengan struktur first in first out (FIFO).

Hal ini berarti elemen pertama yang ditempatkan pada queue adalah yang pertama dipindahkan.

Contoh yang paling populer untuk membayangkan sebuah queue adalah antrian pada kasir sebuah bank. Ketika seorang pelanggan datang, akan menuju ke belakang dari antrian. Setelah pelanggan dilayani, antrian yang berada di depan akan maju.

Pada saat menempatkan elemen pada ujung (*tail*) dari queue disebut dengan *enqueue*, pada saat memindahkan elemen dari kepala (*head*) sebuah queue disebut dengan *dequeue*.




Gambar 5.1 (1) Queue dengan 2 elemen; (2) Queue setelah proses enqueue C, D dan E;
(3) Setelah proses dequeue A dan B

# 4.1 Karakteristik Queue

Karakteristik penting dari antrian adalah :

1. Elemen antrian yaitu item-item data yang terdapat di elemen antrian
2. Front (elemen terdepan dari antrian)
3. Rear (elemen terakhir dari antrian)
4. Jumlah elemen pada antrian (Count)
5. Status antrian

# 4.1 Karakteristik Queue (lanjutan)

1. Penuh
Bila elemen pada antrian mencapai kapasitas maksimum antrian. Pada kondisi ini, tidak mungkin dilakukan penambahan ke antrian. Penambahan elemen menyebabkan kondisi kesalahan overflow.

2. Kosong
Bila tidak ada elemen pada antrian. Pada kondisi ini, tidak mungkin dilakukan pengambilan elemen dari antrian. Pengambilan elemen menyebabkan kondisi kesalahan Overflow.

# 4.2 Representasi Antrian

Representasi antrian secara sekuen relatif lebih sulit dibanding stack.

Seperti dijelaskan di atas bahwa antrian juga merupakan satu kumpulan data.

Dengan demikian tipe data yang sesuai untuk menyajikan antrian adalah menggunakan array atau linked list.

# 4.2.1 Implementasi Antrian dengan Array

Seperti halnya pada tumpukan, maka dalam antrian kita juga mengenal ada dua operasi dasar, yaitu menambah elemen baru yang akan kita tempatkan di bagian belakang antrian dan menghapus elemen yang terletak di bagian depan antrian.

Disamping itu seringkali kita juga perlu melihat apakah antrian mempunyai isi atau dalam keadaan kosong.

Operasi penambahan elemen baru selalu bisa kita lakukan karena tidak ada pembatasan banyaknya elemen dari suatu antrian.

Tetapi untuk menghapus elemen, maka kita harus melihat apakah antrian dalam keadaan kosong atau tidak.

Tentu saja kita tidak mungkin menghapus elemen dari suatu antrian yang sudah kosong.

# Contoh deklarasi antrian

```
#define MAXQUEUE 100;

typedef int ItemType;

typedef struct{
        int Count;
        int Front;
        int Rear;
        ItemType Item[MAXQUEUE];
        }Queue;
```

Front, menunjukkan item yang paling depan, yaitu elemen yang akan dihapus jika dilakukan operasi penghapusan.

Setelah kita melakukan penghapusan, kita melakukan increment pada indeks Front, sehingga indeks menunjuk pada posisi berikutnya.

Jika indeks ini jatuh pada angka tertinggi, yaitu angka paling maksimum dari array (N), maka kita melakukan setting ulang ke 0.

Count menunjukkan jumlah item dalam antrian. Rear menunjukkan posisi dimana setelahnya dapat dimasukkan item berikutnya.

# Implementasi Antrian dengan Array

```
#include <stdio.h>
#include <stdlib.h>
#define MAXQUEUE 100;
typedef int ItemType;
typedef struct{
int Count, Front, Rear;

ItemType Item[MAXQUEUE];
}Queue;
void InitializeQueue(Queue *Q)
{ Q->Count = 0;
Q->Front = 0;
Q->Rear = 0;
} int Empty(Queue *Q)
{ return(Q->Count == 0); }
int Full(Queue *Q) { return(Q->Count ==
MAXQUEUE); }
```

```
void Insert(ItemType ins, Queue *Q)
{ if (Q->Count == MAXQUEUE)
printf("Tidak dapat memasukkan data! Queue
Penuh!"); else {
Q->Item[Q->Rear] = ins;
Q->Rear = (Q->Rear + 1) % MAXQUEUE;
++(Q->Count); } }
void Remove(Queue *Q, ItemType *rm)
{ if (Q->Count == 0)
printf("Tidak dapat mengambil data! Queue
Kosong!");
else {
*rm = Q->Item[Q->Front];
Q->Front = (Q->Front + 1) % MAXQUEUE;
--(Q->Count); } }
```

Jika kita meletakkan beberapa item yang baru dari antrian dalam sebuah array, maka kita menambahkannya pada Rear dan memindah item dari Front.

Dengan penambahan dan pengurangan item ini, permasalahan akan terjadi jika ukuran dari array habis. Kita bisa keluar dari permasalahan ini jika kita merepresentasikan antrian secara circular.

Cara mensimulasikan antrian secara circular dalam array linear menggunakan arithmetic modular. Arithmetic modular menggunakan ekspresi rumus (X % N) untuk menjaga besarnya nilai X pada range 0:N-1.

Jika indeks telah sampai pada N dengan penambahan atau pengurangan tersebut, maka indeks akan diset pada angka 0. Hal yang sama juga dilakukan pada Front jika dilakukan pengambilan item dari antrian.

Setelah mengambil item dari antrian, kita melakukan increment terhadap Front untuk penunjukan pada posisi sesudahnya. Apabila indeks telah berada pada N, maka indeks diset juga pada angka 0. Perintah di bawah ini merupakan perintah yang menunjukkan proses Arithmetic Modular yang diterapkan pada antrian.

```
Front = (Front + 1) % N;
Rear = (Rear +1) % N;
```

# 4.2.2 Implementasi Antrian dengan Linked list

Antrian yang direpresentasikan dengan linked list mempunyai beberapa variasi. Pada kesempatan kali ini hanya direpresentasikan satu macam saja.

Linked list yang digunakan di sini menggunakan struktur yang berisi pointer yang menunjuk pada simpul Front dan Rear dari linked list.

Masing-masing simpul berisi data dari antrian dan juga link yang menunjuk pada simpul selanjutnya dari linked list, yang dinamakan Link.

# Implementasi Antrian dengan Linked List

```
#include <stdio.h>
#include <stdlib.h>
typedef int ItemType;
typedef struct QueueNodeTag {
ItemType Item;
struct QueueNodeTag *Link;
}QueueNode;
typedef struct {
QueueNode *Front;
QueueNode *Rear;
}Queue;
void InitializeQueue(Queue *Q)
{ Q->Front = NULL;
Q->Rear = NULL; }

int Empty(Queue *Q)
{ return(Q->Front == NULL); }
```

```
int Full(Queue *Q)
{ return 0; }
void Insert(ItemType R, Queue *Q)
{ QueueNode *Temp;
Temp = (QueueNode *)
malloc(sizeof(QueueNode));
if (Temp == NULL) {
printf("Queue tidak dapat tercipta");
}else{ Temp->Item = R;
Temp->Link = NULL;
if (Q->Rear == NULL){
Q->Front = Temp;
Q->Rear = Temp; }else{
Q->Rear->Link=Temp;
Q->Rear = Temp; } } }
```

```
void Remove(Queue *Q, ItemType
*F) {  QueueNode *Temp;

if (Q->Front == NULL){

printf("Queue masing kosong!");

}else{

*F = Q->Front->Item;

Temp = Q->Front;

Q->Front = Temp -> Link;

free(Temp);

if(Q->Front == NULL) Q->Rear =
NULL; } }
```

# 4.3 Antrian Berprioritas

Dalam antrian yang telah dibahas sebelumnya, semua elemen yang masuk dalam antrian dianggap mempunyai prioritas yang sama, sehingga elemen yang masuk lebih dahulu akan diproses lebih dahulu.

Dalam praktek, elemen-elemen yang akan masuk dalam suatu antrian ada yang dikatakan mempunyai prioritas yang lebih tinggi disbanding yang lain.

Antrian yang demikian ini disebut dengan antrian berprioritas (priority queue).

# 4.3 Antrian Berprioritas (Lanjutan)

Dalam antrian berprioritas, setiap elemenn yang akan msuk dalam antrian sudah ditentukan lebih dahulu prioritasnya. Dalam hal ini berlaku dua ketentuan, yaitu:

1. Elemen-elemen yang mempunyai prioritas lebih tinggi akan diproses lebih dahulu.
2. Dua elemen yang mempunyai prioritas sama akan dikerjakan sesuai dengan urutan pada saat kedua elemen ini masuk dalam antrian.

Dengan memperhatikan kedua ketentuan di atas, akan berlaku ketentuan bahwa elemen yang mempunyai prioritas lebih tinggi akan dikerjakan lebih dahulu disbanding elemen yang mempunyai prioritas lebih rendah, meskipun elemen yang berprioritas tinggi masuknya sesudah elemen yang berprioritas rendah.

Salah satu contoh antrian berprioritas ini adalah pada sistem berbagi waktu (time-sharing system) dimana program yang mempunyai prioritas tinggi akan dikerjakan lebih dahulu dan program-program yang berprioritas sama akan membentuk antrian biasa.

# 4.3 Antrian Berprioritas (Lanjutan)

Ada beberapa cara untuk mengimplementasikan antrian berprioritas. Salah satu caranya adalah dengan menggunakan linked list. Jika kita menggunakan linked list, khususnya single linked list atau double linked list, maka ada ketentuan lain yang perlu diperhatikan, yaitu:

1. Setiap node dari linked list terdiri tiga bagian, yaitu bagian informasi, angka prioritas dan bagian-bagian penyambung ke simpul lain.

2. Simpul X mendahului (terletak di sebelah kiri) simpul Y, jika prioritas X lebih tinggi dibanding prioritas Y atau jika prioritas X dan Y sama, maka simpul X datang lebih dahulu dibanding dengan Y.

Biasanya dibuat suatu perjanjian bahwa angka prioritas yang lebih kecil menunjukkan derajad prioritas yang lebih tinggi. Sebagai contoh, jika angka prioritas pada simpul X adalah 1 dan pada simpul Y adalah 2, maka dikatakan bahwa simpul X berprioritas lebih tinggi dibanding dengan simpul Y.

# 4.4 Kesimpulan

1. Antrian (qeue) adalah sebuah bentuk struktur yang berdasarkan pada proses FIFO (First In First Out)
2. Antrian dapat diimplementasikan dengan menggunakan array atau linked list
3. Antrian dapat diberikan prioritas untuk menunjukkan data mana yang akan diolah terlebih dahulu

# Terima Kasih

Pertanyaan?