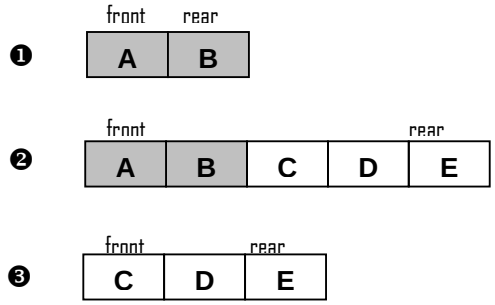
BAB V
Antrian (Queue)

Tujuan

1. Memahami berbagai cara untuk merepresentasikan queue secara sekuensial maupun dengan menggunakan linked list
2. Memahami implementasi queue dalam menyelesaikan sebuah permasalahan

---

Karakteristik yang membedakan queue (antrian) dari stack adalah bahwa cara penyimpanan dan pengambilan data pada queue yang menggunakan 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 *(rear)* dari queue disebut dengan *enqueue*, pada saat memindahkan elemen dari kepala (*front*) sebuah queue disebut dengan *dequeue*. Pada Gambar 5.1 diperlihatkan sebuah queue serta proses *enqueue* dan *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

5.1 Karakteristik Antrian

Karakteristik penting dari antrian adalah :

1. Elemen antrian yaitu data yang terdapat di elemen antrian
2. Elemen terdepan dari antrian → front
3. Elemen terakhir dari antrian → rear
4. Jumlah elemen pada antrian → count
5. Status/kondisi antrian

Kondisi antrian yang menjadi perhatian adalah ;

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.*

5.2 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.

5.2.1. 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, perlu dicek apakah antrian dalam keadaan penuh atau tidak. Sedangkan 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.

Untuk menyajikan antrian menggunakan array, maka kita membutuhkan deklarasi antrian, misalnya sebagai berikut:

```
#define MAXQUEUE 100;

typedef char itemType;
typedef struct {
 int count;
 int front;
 int rear;
 itemType data[MAXQUEUE];
} queue;
```

- `count` menunjukkan jumlah data dalam antrian.
- `front`, menunjukkan data 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.
- `rear` menunjukkan posisi di mana setelahnya dapat dimasukkan data berikutnya.
- Array `data[0:n-1]` berisi n data yang merupakan isi dari antrian yang berada pada posisi 0:n-1, di mana pada posisi ini dapat diindikasikan dua pengenal, yaitu `front` dan `rear`.

Jika kita meletakkan beberapa data yang baru dari antrian dalam sebuah array, maka kita menambahkannya pada `rear`, kemudian memindah `rear` ke posisi selanjutnya, sedangkan penghapusan data dengan cara mengambil data pada `front` dan memindah `front` ke posisi berikutnya. Dengan penambahan dan pengurangan data 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 data dari antrian. Setelah mengambil data 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 untuk *Arithmetic Modular* yang diterapkan pada antrian.

```
front = (front + 1) % N;
rear = (rear +1) % N;
```

5.2.2. Operasi Pada Antrian yang Menggunakan Array

Beberapa operasi yang dilakukan terhadap antrian yang diimplementasikan menggunakan array adalah sebagai berikut :

5.2.2.1 Inisialiasasi antrian

Operasi ini bertujuan untuk menciptakan queue sebagai antrian kosong

```
void inisialisasiQueue(queue *q) {
 q->count = 0;
 q->front = 0;
 q->rear = 0;
}
```

5.2.2.2 Pengecekan antrian dalam keadaan kosong atau tidak

Fungsi ini bertujuan untuk melakukan pengecekan apakah antrian dalam kondisi kosong

```
int empty(queue *q){
    if(q->count == 0)        //return(q->count == 0);
       return 1;
    else
       return 0;
}
```

5.2.2.3 Pengecekan antrian dalam kondisi penuh atau tidak

Fungsi ini bertujuan untuk melakukan pengecekan apakah antrian dalam kondisi penuh

```
int full(queue *q){
    if(q->count == MAXQUEUE)  //return(q->count == MAXQUEUE);
       return 1;
    else
       return 0;
}
```

5.2.2.4 Operasi Enqueue

Untuk memasukkan data `x` yang baru ke dalam antrian, maka hal-hal yang harus dilakukan adalah :

a. menambahkan data tsb pada posisi `rear` :

```
q->data[q->rear] = x;
```

b. meng-update posisi `rear` menggunakan arithmetic modular → agar menjadi array circular; ketika posisi `rear` telah mencapai `MAXQUEUE` akan diset kembali ke 0:

```
q->rear = (q->rear + 1) % MAXQUEUE;
```

c. meng-update isi count dengan menambahnya dengan 1 :

```
++(q->count);
```

d. Pemanggilan fungsinya :

```
enqueue(input, &antrian);
```

→ `input` adalah data yang akan dimasukkan ke dalam antrian

Implementasi selengkapnya adalah sebagai berikut :

```
void enqueue(itemType x, queue *q) {
    if (full(q)) {
       puts("STOP!!!");
       puts("Queue Penuh!");
    }
```

```
        else {
           q->data[q->rear] = x;
              /* using arithmetic modular -> spy mjd array circular,
              ketika posisi rear telah mencapai MAXQUEUE akan diset
              kembali ke 0 */
           q->rear = (q->rear + 1) % MAXQUEUE;
           ++(q->count);
        }
     }
```

5.2.2.5 Operasi Dequeue

Untuk mengambil/menghapus data dari sebuah antrian, yaitu elemen puncak dari antrian. Maka hal-hal yang harus dilakukan adalah :

a. mengambil data pada posisi `front` dan meng-assign-nya ke sebuah variabel penampung (misalnya `tampung`) :

```
tampung = q->data[q->front];
```

b. meng-update posisi `front` menggunakan arithmetic modular -> agar menjadi array circular; ketika posisi `front` telah mencapai `MAXQUEUE` akan diset kembali ke 0:

```
q->front = (q->front + 1) % MAXQUEUE;
```

c. meng-update isi count dengan menguranginya dengan 1 :

```
--(q->count);
```

d. Fungsi ini memiliki return value bertipe `itemType` → yaitu hasil pengambilan yang ditampung pada variable penampung `tampung`; sehingga di akhir definisi fungsi ini terdapat statemen :

```
return tampung;
```

e. Pemanggilan fungsinya :

```
dequeue(&antrian);
```

Implementasi selengkapnya adalah sebagai berikut :

```
itemType dequeue(queue *q) {
 itemType tampung;

 if(empty(q)){
  puts("Tidak dapat mengambil data!");
  puts("Queue Kosong!");
  tampung = ' ';
 }
 else {
  tampung = q->data[q->front];
    /*using arithmetic modular -> spy mjd array circular, ketika
    posisi front telah mencapai MAXQUEUE akan diset kembali ke 0 */
  q->front = (q->front + 1) % MAXQUEUE;
  --(q->count);
 }
 return tampung;        //data yg diambil dari posisi front
}
```

5.2.3 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 (`data`) dari antrian dan juga link yang menunjuk pada simpul selanjutnya dari linked list, yang dinamakan `next`.

Untuk menyajikan antrian menggunakan linked list, maka kita membutuhkan definisi, misalnya sebagai berikut:

```
typedef char itemType;
typedef struct simpul queueNode;
struct simpul {
 itemType data;
 queueNode *next;
};

typedef struct {
 queueNode *front;
 queueNode *rear;
} queue;
```

5.2.4. Operasi Pada Antrian yang Menggunakan Linked LIst

Beberapa operasi yang dilakukan terhadap antrian yang diimplementasikan menggunakan array adalah sebagai berikut :

5.2.4.1 Inisialiasasi antrian

Operasi ini bertujuan untuk menciptakan queue sebagai antrian kosong

```
void inisialisasiQueue(queue *q){
   q->front = NULL;
   q->rear = NULL;
  }
```

5.2.4.2 Pengecekan antrian dalam keadaan kosong atau tidak

Fungsi ini bertujuan untuk melakukan pengecekan apakah antrian dalam kondisi kosong

```
int empty(queue *q) {
    if(q->front == NULL)         //return(q->front == NULL);
       return 1;
    else
       return 0;
}
```

5.2.4.3 Operasi Enqueue

Untuk memasukkan data yang baru ke dalam antrian, maka hal-hal yang harus dilakukan adalah :

a. mengalokasikan node baru yang akan diinsert, masukkan data datanya dan set pointer next-nya = NULL

```
     p = (queueNode *) malloc(sizeof(queueNode));
     if (p == NULL) {
       puts("STOP!!!");
       puts("Memory tidak bisa dialokasikan");
     }
     else {
       p->data = x;                //x adl data yg akan disimpan
       p->next = NULL;             //set NULL = insert akhir
     }
```

b. Sisipkan node baru tsb dengan terlebih dahulu mengecek kondisi antrian. Jika antrian masih kosong, node baru tsb akan ditunjuk oleh pointer `front` sekaligus `rear`. Jika antrian sudah ada isinya, maka node baru tsb akan disisipkan pada posisi `rear`

```
q->rear->next=p;
```

c. meng-update posisi `rear`

```
q->rear = p;
```

d. Pemanggilan fungsinya :

```
enqueue(input, &antrian);
```

→ `input` adalah data yang akan dimasukkan ke dalam antrian

5.2.4.4 Operasi Dequeue

Untuk mengambil/menghapus data dari sebuah antrian, yaitu elemen yang ditunjuk oleh pointer `front` diperlukan sebuah pointer bantuan, misalnya `hapus`. Prosedurnya adalah sebagai berikut:

a. Cek kondisi antrian apakah dalam keadaan kosong. Jika antrian dalam keadaan kosong, tampilkan pesannya dan berikan return value berupa tanda ‘ ‘ (spasi) sebagai penanda antrian kosong.

```
if (empty(q)) {
  puts("Tidak dapat mengambil data!");
  puts("Queue Kosong!");
  return ' ';      //spasi sbg penanda queue kosong
```

b. Jika antrian tidak kosong, baca `data` dari node yang ditunjuk oleh pointer `front`, tampung dalam sebuah variabel, misalnya `tampung`.

```
tampung = q->front->data;
```

c. Arahkan pointer hapus pada posisi `front`

```
hapus = q->front;
```

d. Update posisi pointer `front` ke posisi node sesudah node yang ditunjuk oleh pointer `hapus`

```
q->front = hapus -> next;
```

e. Bebaskan memory yang ditunjuk oleh pointer `hapus` dan NULL-kan pointer `hapus`

```
free(hapus);
hapus = NULL;
```

f. Cek apakah pointer front menunjuk ke NULL. Jika ya, arahkan juga pointer rear ke NULL, yang berarti antrian kembali kosong

```
if(q->front == NULL)
  q->rear = NULL;
```

g. Fungsi ini memiliki return value bertipe `itemType` → yaitu hasil pengambilan yang ditampung pada variable penampung `x`; sehingga di akhir definisi fungsi ini terdapat statemen :

```
return tampung;
```

h. Pemanggilan fungsinya :

```
dequeue(&antrian);
```

5.3 Antrian Berprioritas

Dalam antrian yang telah dibahas di atas, 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 dibanding yang lain. Antrian yang demikian ini disebut dengan antrian berprioritas (*priority queue*).

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 dibanding 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)* di mana program yang mempunyai prioritas tinggi akan dikerjakan lebih dahulu dan program-program yang berprioritas sama akan membentuk antrian biasa.

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.

5.4 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