PERTEMUAN 13
QUEUE

A. TUJUAN PEMBELAJARAN

Setelah menyelesaikan pertemuan ini, mahasiswa mampu mempraktekkan:

1. Pengenalan Queue
2. Operasi Queue
3. Implementasi Queue sebagai array
4. Inisialisasi Queue
5. Empty Dan Full Queue
6. Operasi addQueue, front, back, dan deleteQueue

B. URAIAN MATERI

1. Pengenalan Queue

Pertemuan ini membahas tentang struktur data penting lainnya, yang disebut antrian. Pengertian antrian dalam ilmu komputer sama dengan pengertian antrian yang biasa Anda lakukan dalam kehidupan sehari-hari. Ada antrian pelanggan di bank atau di toko bahan makanan dan antrian mobil menunggu untuk melewati gerbang tol. Demikian pula, karena komputer dapat mengirim permintaan cetak lebih cepat daripada yang dapat dicetak printer, antrean dokumen sering kali menunggu untuk dicetak di printer. Aturan umum untuk memproses elemen dalam antrian adalah bahwa pelanggan di depan antrian dilayani berikutnya dan ketika pelanggan baru tiba, dia berdiri di akhir antrian. Artinya, antrian adalah struktur data First In First Out.

2. Operasi Queue

Dari definisi antrian, kita melihat bahwa dua operasi kunci adalah tambah dan hapus. Kami menyebutnya add operasi addQueue dan operasi delete deleteQueue. Karena elemen tidak dapat dihapus dari antrian kosong atau ditambahkan ke antrian penuh, kita memerlukan dua operasi lagi untuk berhasil

mengimplementasikan operasi addQueue dan deleteQueue: isEmptyQueue (memeriksa apakah antrian kosong) dan isFullQueue (memeriksa apakah antrian sudah penuh) .

Kami juga membutuhkan operasi, inisialisasiQueue, untuk menginisialisasi antrian ke keadaan kosong. Selain itu, untuk mengambil elemen pertama dan terakhir dari antrian, kami menyertakan operasi depan dan belakang seperti yang dijelaskan dalam daftar berikut. Beberapa operasi antrian adalah sebagai berikut:

initializeQueue — Menginisialisasi antrian ke keadaan kosong.

isEmptyQueue — Menentukan apakah antrian kosong. Jika antri

kosong, ini mengembalikan nilai true; jika tidak, itu mengembalikan nilai salah.

isFullQueue — Menentukan apakah antrian penuh. Jika antriannya

full, ini mengembalikan nilai true; jika tidak, itu mengembalikan nilai salah.

front — Mengembalikan bagian depan, yaitu elemen pertama antrean.

back — Mengembalikan elemen terakhir dari antrian. Sebelum operasi ini, antrian harus ada dan tidak boleh kosong.

addQueue — Menambahkan elemen baru ke belakang antrian. Sebelum operasi ini, antrian harus ada dan tidak boleh penuh.

deleteQueue — Menghapus elemen depan dari antrian. Sebelum

operasi ini, antrian harus ada dan tidak boleh kosong.

Seperti dalam kasus tumpukan, antrian dapat disimpan dalam larik atau dalam struktur terkait. Kami akan mempertimbangkan kedua penerapan tersebut. Karena elemen ditambahkan di satu ujung dan dihapus dari ujung lainnya, kita memerlukan dua penunjuk untuk melacak bagian depan dan belakang antrian, yang disebut queueFront dan queueRear.

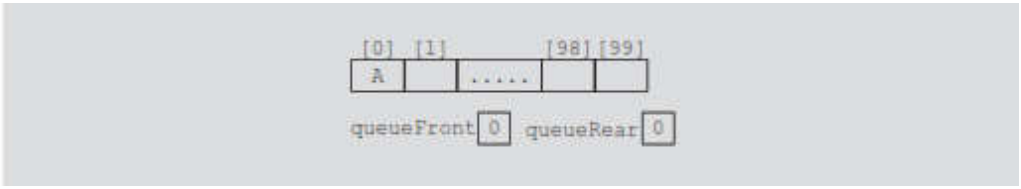
3. Implementasi Queue sebagai Array

Sebelum memberikan definisi kelas untuk mengimplementasikan antrian sebagai ADT, kita perlu memutuskan berapa banyak variabel anggota yang dibutuhkan untuk mengimplementasikan antrian. Tentu saja, kita memerlukan array untuk menyimpan elemen antrian, variabel queueFront dan queueRear untuk melacak elemen pertama dan terakhir dari antrian, dan variabel maxQueueSize untuk menentukan ukuran maksimum antrian. Jadi, kita membutuhkan setidaknya empat variabel anggota.

Sebelum menulis algoritma untuk mengimplementasikan operasi antrian, kita perlu memutuskan bagaimana menggunakan queueFront dan queueRear untuk mengakses elemen antrian. Bagaimana queueFront dan queueRear menunjukkan bahwa antrian kosong atau penuh? Misalkan queueFront memberikan indeks elemen pertama antrian, dan queueRear memberikan indeks elemen terakhir antrian. Untuk menambahkan elemen ke antrian, pertama kita maju queueRear ke posisi array berikutnya dan kemudian menambahkan elemen ke posisi yang ditunjuk queueRear. Untuk menghapus elemen dari antrian, pertama-tama kita mengambil elemen yang diarahkan oleh queueFront dan kemudian memajukan queueFront ke elemen antrian berikutnya. Jadi, queueFront berubah setelah setiap operasi deleteQueue dan queueRear berubah setelah setiap operasi addQueue.

Mari kita lihat apa yang terjadi ketika queueFront berubah setelah operasi deleteQueue dan queueRear berubah setelah operasi addQueue. Asumsikan bahwa array untuk menampung file

elemen antrian berukuran 100. Awalnya, antrian kosong. Setelah operasi: addQueue (Queue, 'A');

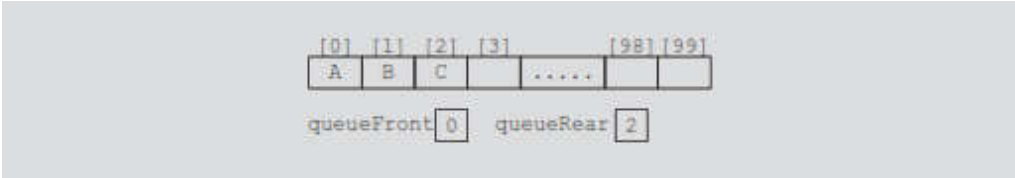


Gambar 13.31Queue setelah operasi addQueue pertama

Setelah dua operasi addQueue lagi:

addQueue (Queue, 'B');

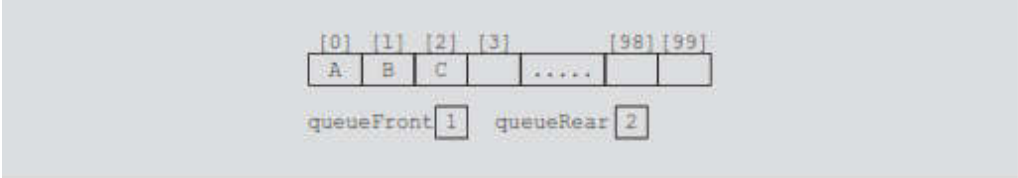
addQueue (Queue, 'C');



Gambar 13.32Queue setelah dua operasi addQueue

Sekarang pertimbangkan operasi deleteQueue:

deleteQueue (); Setelah operasi ini, array yang berisi antrian adalah

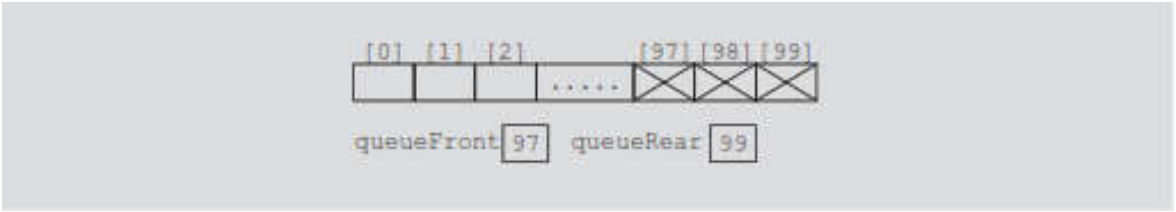


Gambar 13.33Queue setelah operasi deleteQueue

Akankah desain antrian ini berfungsi? Misalkan A berarti menambahkan (yaitu, addQueue) elemen ke antrian, dan D berarti menghapus (yaitu, deleteQueue) elemen dari antrian. Pertimbangkan urutan operasi berikut:

AAADADADADADADADA...

Urutan operasi ini pada akhirnya akan menyetel indeks queueRear untuk menunjuk ke posisi larik terakhir, memberikan kesan bahwa antrian sudah penuh. Namun, antrian hanya memiliki dua atau tiga elemen dan bagian depan array kosong.

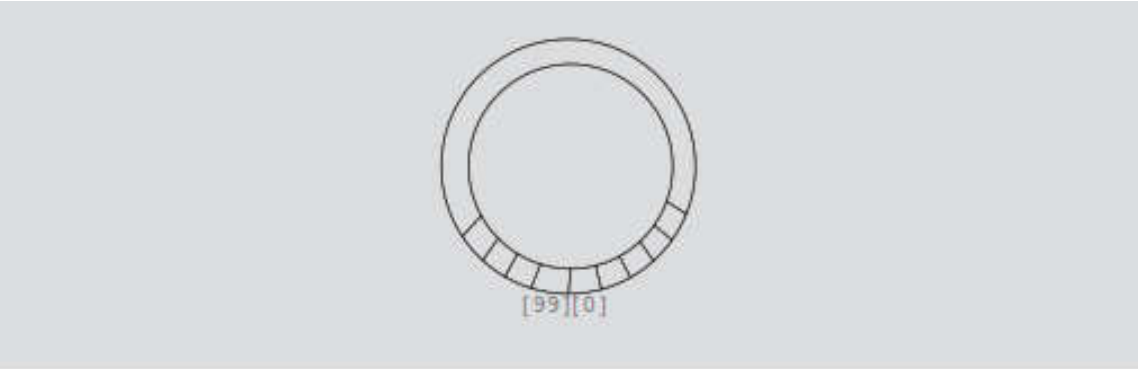


Gambar 13.34Queue setelah urutan operasi AAADADADADADA ...

Salah satu solusi untuk masalah ini adalah ketika antrian meluap ke belakang (yaitu, queueRear menunjuk ke posisi array terakhir), kita dapat memeriksa nilai indeks queueFront. Jika nilai queueFront menunjukkan bahwa ada ruang di depan larik, maka saat queueRear sampai ke posisi larik terakhir,

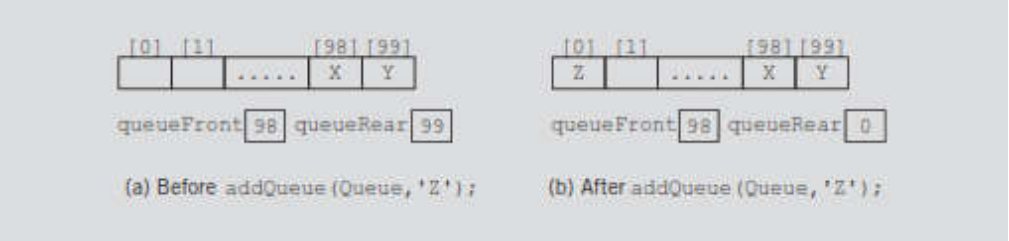
kita bisa menggeser semua elemen antrian ke posisi larik pertama. Solusi ini bagus jika ukuran antrian sangat kecil; jika tidak, program mungkin akan berjalan lebih lambat.

Solusi lain untuk masalah ini adalah dengan menganggap bahwa array itu melingkar — yaitu, posisi array pertama segera mengikuti posisi array terakhir.



Gambar 13.35Circular queue

Kita akan menganggap array yang berisi antrian berbentuk lingkaran, meskipun kita akan menggambar figur dari array yang menahan elemen antrian seperti sebelumnya.



Gambar 13.36Queue sebelum dan sesudah operasi penambahan

Setelah operasi addQueue (Queue, 'Z');

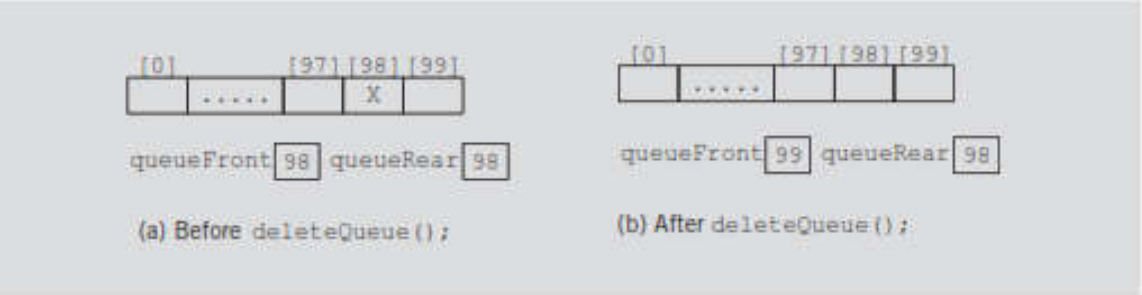
Karena array yang berisi antrian berbentuk lingkaran, kita dapat menggunakan pernyataan berikut untuk memajukan queueRear (queueFront) ke posisi array berikutnya:

queueRear = (queueRear + 1) % maxQueueSize;

Jika queueRear <maxQueueSize - 1, maka queueRear + 1 <= maxQueueSize - 1, jadi (queueRear + 1)% maxQueueSize = queueRear + 1.

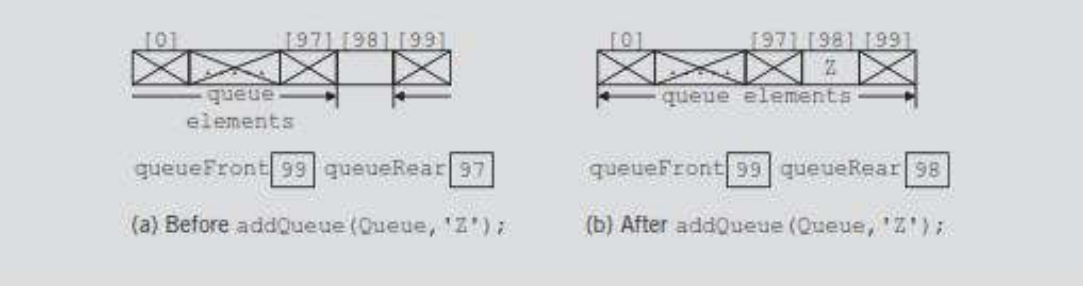
Jika queueRear == maxQueueSize - 1 (yaitu, queueRear menunjuk ke posisi array terakhir), queueRear + 1 == maxQueueSize, jadi (queueRear + 1)% maxQueueSize = 0. Dalam kasus ini, queueRear akan disetel ke 0, yang merupakan posisi array pertama.

Desain antrian ini sepertinya berfungsi dengan baik. Sebelum kita menulis algoritma untuk mengimplementasikan operasi antrian, pertimbangkan dua kasus berikut :



Gambar 13.37Queue sebelum dan sesudah operasi hapus

Setelah operasi deleteQueue ();



Gambar 13.38Queue sebelum dan sesudah operasi hapus

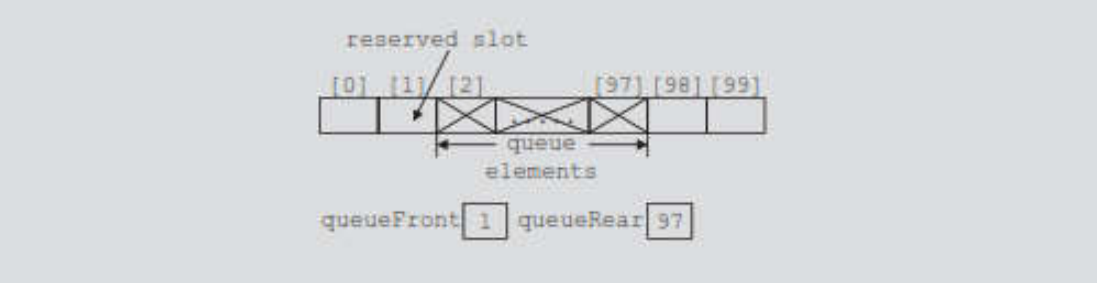
Setelah operasi addQueue (Queue, 'Z') ;,

Masalah ini memiliki beberapa solusi. Salah satu solusinya adalah menghitung. Selain variabel anggota queueFront dan queueRear, kita membutuhkan variabel lain, count, untuk mengimplementasikan antrian. Nilai hitungan bertambah setiap kali elemen baru ditambahkan ke antrian, dan dikurangi setiap kali elemen dihapus dari antrian. Dalam kasus ini, fungsi initializeQueue menginisialisasi hitungan ke 0. Solusi ini sangat berguna jika pengguna antrean sering kali perlu mengetahui jumlah elemen dalam antrean.

Solusi lain adalah membiarkan queueFront menunjukkan indeks posisi larik sebelum elemen pertama antrian, daripada indeks elemen pertama

(aktual) itu sendiri. Dalam kasus ini, dengan asumsi queueRear masih menunjukkan indeks elemen terakhir dalam antrian, antrian kosong jika queueFront == queueRear. Dalam solusi ini, slot yang ditunjukkan oleh indeks queueFront (yaitu, slot sebelum elemen benar yang pertama) dicadangkan. Antrian akan penuh jika ruang yang tersedia berikutnya adalah slot yang dipesan khusus yang ditunjukkan oleh queueFront. Akhirnya, karena posisi array yang ditunjukkan oleh queueFront harus tetap kosong, jika file

ukuran array, katakanlah, 100, maka 99 elemen dapat disimpan dalam antrian.



Gambar 13.39Queue sebelum dan sesudah operasi hapus

4. Inisialisasi Queue

Operasi ini menginisialisasi antrian ke keadaan kosong. Elemen pertama ditambahkan pada posisi array pertama. Oleh karena itu, kami menginisialisasi queueFront ke 0, queueRear ke maxQueueSize - 1, dan menghitung ke 0.

Gambar 13.40Emptyqueue

Definisi dari function initializeQueue adalah sebagai berikut:

```
Template<Class Type>
```

```
voidqueueType::initializeQueue()
{
queueFront = 0;
queueRear = maxQueueSize - 1;
count = 0;
//endinitializeQueue
```

a. Front

Operasi ini mengembalikan elemen pertama dari antrian. Jika antrian tidak kosong, elemen antrian yang ditunjukkan oleh indeks queueFront dikembalikan; jika tidak, program akan berhenti.

```
template <class Type>
Type queueType<Type>::front() const
{
assert(!isEmptyQueue());
return list[queueFront];
} //end front
```

b. Back

Operasi ini mengembalikan elemen terakhir dari antrian. Jika antrian tidak kosong, elemen antrian yang ditunjukkan oleh indeks queueRear dikembalikan; jika tidak, program akan berhenti.

```
template <class Type>
Type queueType<Type>::back() const
{
assert(!isEmptyQueue());
```

```
return list[queueRear];
} //end back
```

c. Add Queue

Selanjutnya, kami menerapkan operasi addQueue. Karena queueRear menunjuk ke elemen terakhir dari antrian, untuk menambahkan elemen baru ke antrian, pertama-tama kita memajukan queueRear ke posisi array berikutnya, dan kemudian menambahkan elemen baru ke posisi array yang ditunjukkan oleh queueRear. Kami juga menambah hitungan sebesar 1. Jadi fungsi addQueue adalah sebagai berikut:

```
template <class Type>
{
if (!isFullQueue())
{
queueRear = (queueRear + 1) % maxQueueSize; // gunakan operator mod
//untuk memajukan queueRear karena lariknya melingkar
count++;
list[queueRear] = newElement;
}
else
cout << "Cannot add to a full queue." << endl;
} //end addQueue
```

d. Delete Queue

Untuk mengimplementasikan operasi deleteQueue, kami mengakses indeks queueFront. Karena queueFront menunjuk ke posisi array yang berisi elemen antrian pertama, untuk menghapus elemen antrian pertama, kita

mengurangi jumlah 1 dan memajukan queueFront ke elemen antrian berikutnya. Jadi fungsi deleteQueue adalah sebagai berikut:

```
template <class Type>
void queueType<Type>::deleteQueue()
if (!isEmptyQueue())
{
count--;
queueFront = (queueFront + 1) % maxQueueSize; //// gunakan operator //mod untuk memajukan queueFront karena lariknya melingkar
}
else
cout << "Cannot remove from an empty queue" << endl;
} //end deleteQueue
```

5. Empty dan Full Queue

Queue kosong jika queueFront adalah NULL. Memori untuk menyimpan elemen antrian dialokasikan secara dinamis. Oleh karena itu, antrian tidak pernah penuh sehingga fungsi untuk mengimplementasikan operasi isFullQueue mengembalikan nilai false. (Queue penuh hanya jika program kehabisan memori.)

```
template <class Type>
bool linkedQueueType<Type>::isEmptyQueue() const
{
return(queueFront == NULL);
} //end
```

```
template <class Type>
```

```
bool linkedQueueType<Type>::isFullQueue() const
{
return false;
} //end isFullQueue
```

Perhatikan bahwa pada kenyataannya, dalam implementasi antrian yang terhubung, fungsi isFullQueue tidak berlaku karena secara logis antrian tidak pernah penuh. Namun, Anda harus memberikan definisinya karena ini disertakan sebagai fungsi abstrak dalam queueADT kelas induk.

6. Operasi add Queue, front, back, dan deleteQueue

Operasi addQueue menambahkan elemen baru di akhir antrian. Untuk mengimplementasikan operasi ini, kita mengakses pointer queueRear.

Jika antrian tidak kosong, operasi front mengembalikan elemen pertama dari antrian dan elemen antrian yang ditunjukkan oleh penunjuk queueFront dikembalikan. Jika antrian kosong, fungsi front menghentikan program.

Jika Queue tidak kosong, operasi back mengembalikan elemen terakhir dari antrian dan elemen antrian yang ditunjukkan oleh penunjuk queueRear dikembalikan. Jika antrian kosong, fungsi back menghentikan program. Demikian pula, jika antrian tidak kosong, operasi deleteQueue menghapus elemen pertama dari antrian, dan kita mengakses pointer queueFront.

Definisi fungsi untuk mengimplementasikan operasi ini adalah sebagai berikut:

```
template <class Type>
void linkedQueueType<Type>::addQueue(const Type& newElement)
{
odeType<Type> *newNode;
newNode = new nodeType<Type>;
newNode->info = newElement;
newNode->link = NULL;
if (queueFront == NULL)
{
queueFront = newNode;
queueRear = newNode;
}
else //add newNode at the end
{
queueRear->link = newNode;
queueRear = queueRear->link;
}
}//end addQueue
```

Ketika objek Queue keluar dari ruang lingkup, destruktor menghancurkan antrian; yaitu, membatalkan alokasi memori yang ditempati oleh elemen antrean. Definisi fungsi untuk mengimplementasikan destruktor mirip dengan definisi fungsi initializeQueue. Selain itu, fungsi untuk mengimplementasikan konstruktor salinan dan membebani operator penugasan serupa dengan fungsi terkait untuk Stack.

C. SOAL LATIHAN/TUGAS

Latihan	Petunjuk Pengerjaan Tugas
Latihan Dan Tugas 13	1. Apa yang dimaksud dengan Queue? 2. Jelaskan apa Operasi addQueue, front, back, dan deleteQueue? 3. Jelaskan apa itu Empty Dan Full Queue? 4. Buatlah program sederhana menggunakan queue? 5. Buatlah program sederhana queue dengan menggunakan singly linked list?

D. REFERENSI

Malik, 2010. *Data Structureusing C++ Juni 2010. Diambil dari :*

https://bu.edu.eg (5 desember 2020)