MAKALAH ALGORITMA
TENTANG
QUEUE (ANTREAN)

Angga Kresnabayu – 140810160001

Dzakia Rayhana – 140810160015

Syafira Fitra Annisa – 140810160047

Shofiyyah Nadhiroh – 140810160057

Patricia Joanne – 140810160065

Dosen Pembimbing:

Erick Paulus, M.Kom.

PROGRAM STUDI S-1 TEKNIK INFORMATIKA

DEPARTEMEN ILMU KOMPUTER

UNIVERSITAS PADJADJARAN

2017

KATA PENGANTAR

Puji dan syukur tim penyusun panjatkan kepada Tuhan Yang Maha Esa yang telah memberikan kekuatan tim penyusun untuk menyusun makalah algoritma tentang queue atau antrean.

Makalah ini disusun sebagai salah satu tugas mata kuliah Struktur Data berdasarkan informasi yang didapat dan hasil diskusi sederhana dan pembelajaran di kelas yang telah dilakukan oleh tim penyusun.

Tim penyusun mengucapkan terima kasih kepada dosen yang telah membimbing tim penyusun, baik itu secara langsung maupun tidak langsung serta Universitas Padjadjaran yang telah menyediakan fasilitas WiFi untuk menunjang tim penyusun menuntaskan makalah ini.

Tim penyusun menyadari bahwa makalah ini masih banyak kekurangan dan kelemahannya, baik dalam isi maupun sistematikanya. Oleh sebab itu, tim penyusun sangat mengharapkan kritik dan saran yang membangun dalam upaya menyempurnakan makalah ini. Tim penyusun juga mengharapkan agar makalah ini dapat memberikan manfaat bagi pembacanya.

Jatinangor, 8 Mei 2017

Tim Penyusun

DAFTAR ISI

BAB I
PENDAHULUAN

1.1. Latar Belakang

Salah satu konsep yang sangat berguna di dalam ilmu komputer adalah satu bentuk struktur data yang disebut dengan antrean. Dalam hal ini kami mencoba mengenali mengapa queue (antrean) sangat berguna dan memainkan peranan penting dalam pemrograman dan bahasa pemrograman.

1.2. Tujuan

Adapun tujuan dari penyusunan makalah ini adalah untuk mengetahui sejauh mana pemahaman kami mengenai queue.

1.3. Rumusan Masalah

1. Apa yang dimaksud dengan Queue (Antrean)?
2. Apa definisi dari Queue (Antrean)?
3. Bagaimana cara mendeklarasikan Queue (Antrean)?
4. Bagaimana cara menjalankan operasi pada Queue (Antrean)? (Insert, Delete)

BAB II
ISI

2.1. Teori Umum

1. Pengertian Queue

Queue pada Struktur Data atau antrean adalah sekumpulan data yang mana penambahan elemen hanya bisa dilakukan pada suatu ujung disebut dengan sisi belakang (rear), dan penghapusan (pengambilan elemen) dilakukan lewat ujung lain (disebut dengan sisi depan atau front).

Pada stack atau tumpukan menggunakan prinsip "Masuk terakhir keluar pertama" atau LIFO (Last In First Out), sedangkan pada queue atau antrean prinsip yang digunakan adalah "Masuk Pertama Keluar Pertama" atau FIFO (First In First Out).

Queue atau antrean pada struktur data banyak kita jumpai dalam kehidupan sehari-hari, misalnya pemakaian sistem komputer berbagi waktu (time-sharing computer system) dimana ada sejumlah pemakai yang akan menggunakan sistem tersebut secara serempak.

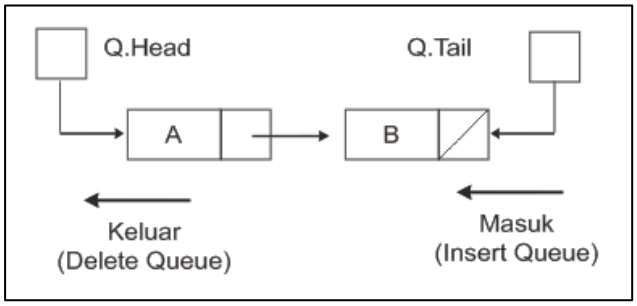
Pada queue terdapat satu buah pintu masuk di suatu ujung dan satu buah pintu keluar di ujung satunya dimana membutuhkan variabel Head dan Tail (depan/front dan belakang/rear).

2. Definisi Queue

Queue memiliki karakteristik yaitu elemen antrian, head (elemen terdepan), tail (elemen terakhir), jumlah elemen pada antrian, dan status antrian.

Cara mendefinisikan queue ada dua cara, yaitu menggunakan array dan list berkait. Yang membedakan keduanya adalah pada isi fungsi operasinya.

Gambaran queue secara umum:



3. Deklarasi Queue

- Menggunakan Array

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

const int maxElemen = 255;
```

```
struct Queue{
    char isi[maxElemen];
    int head;
    int tail;
};

void createQueue(Queue& Q){
Q.head = 0;
Q.tail = -1;
}
```

- Menggunakan List Berkait

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

struct elqueue{
    char info;
    elqueue* next;
};

typedef elqueue* pointer;

struct Queue{
    pointer Head;
    pointer Tail;
};
Queue Q;
```

4. Inisialisasi Queue

Pada mulanya isi head dengan -1, karena dimulai dari 0, yang berarti queue kosong. Head adalah suatu variabel penanda dalam queue yang menunjukkan elemen terdepan queue.

5. Operasi Queue

Operasi-operasi yang biasanya tredapat pada **Queue** yaitu:

- Insert: menambah item pada belakang antrean
- Delete: menghapus elemen depan dari antrean

2.2. Operasi Queue dengan Array

1. Jenis-jenis operasi

- Insert

  Ada 2 kasus yang perlu ditangani yaitu:

  1) Queue sudah penuh, maka tidak ada penambahan elemen.
  2) Queue belum penuh, maka index dari Tail bertambah 1 dan kemudian di posisi index yang baru, elemen yang baru disisipkan.

```
void insertQArray(arrayQ& aq, char baru){
    if(aq.tail == maxElem-1){
        cout<<"Antrian penuh"<<endl;
    }
    else{
        aq.tail = aq.tail+1;
        aq.isi[aq.tail] = baru;
    }
}
```

- Delete

  Ada 2 kasus yang harus ditangani yaitu:

  1) Queue sudah kosong, dengan ciri index head > tail atau tail = -1, maka tidak ada penghapusan.
  2) Queue masih ada isi, setelah data yang dihapus diamankan maka selanjutnya elemen yang lain digeser ke depan satu persatu.

```
void deleteQArray(arrayQ& aq, char& hapus){
    if(aq.head > aq.tail){
        cout<<"Antrian Kosong"<<endl;
    }
    else{
        hapus = aq.isi[aq.head];
        for(int i=0; i<aq.tail; i++){
            aq.isi[aq.tail] = aq.isi[aq.tail+1];
        }
        aq.tail = aq.tail-1;
    }
}
```

2. Contoh pengimplementasian

- Daftar nama dan npm dengan menu operasi queue

```
/*Nama Program: Daftar mahasiswa dengan queue
Nama: Shofiyyah Nadhiroh
NPM: 140810160057
Tanggal Buat: Mei 2017
Deskripsi: makalah
*****************************************************************

#include <iostream>
#include <string.h>
#include <iomanip>
#include <stdlib.h>
using namespace std;

const int maxElem = 255;

struct mhs{
    char nama[10];
    char npm[8];
};

struct queue{
    mhs isi[maxElem];
    int head;
    int tail;
};
queue q;
```

```
void isi(mhs& baru){
    cout<<"Masukkan nama : "; cin>>baru.nama;
    cout<<"Masukkan npm : "; cin>>baru.npm;
}

void insertQueue(queue& q, mhs baru){
    if(q.tail == maxElem-1){
        cout<<"Antrian penuh"<<endl;
    }
    else{
        q.tail = q.tail+1;
        q.isi[q.tail] = baru;
    }
}
```

```
void deleteQueue(queue& q, mhs& hapus){
    if(q.head > q.tail){
        cout<<"Antrian Kosong"<<endl;
    }
    else{
        hapus = q.isi[q.head];
        for(int i=0; i<q.tail; i++){
            q.isi[q.tail] = q.isi[q.tail+1];
        }
        q.tail = q.tail-1;
    }
}
```

```
void cetakQueue(queue q){
    int i=0;
    while(i != q.tail+1){
        cout<<"nama : "<<q.isi[i].nama<<endl;
        cout<<"npm : "<<q.isi[i].npm<<endl;
        i++;
    }
}
```

```
int main(){
    queue q;
    mhs baru, hapus;
    createQueue(q);
    int pilihan;
    do{
        system("cls");
        cout<<"1. Insert Queue"<<endl;
        cout<<"2. Delete Queue"<<endl;
        cout<<"3. Cetak Data"<<endl;
        cout<<"0. Exit"<<endl;
        cout<<"Masukan Pilihan : "; cin>>pilihan;
        if(pilihan == 1){
            isi(baru);
            insertQueue(q, baru);
            cetakQueue(q);
            system("pause");
        }else if(pilihan == 2){
            deleteQueue(q, hapus);
            cetakQueue(q);
            system("pause");
        }else if(pilihan == 3){
            cetakQueue(q);
            system("pause");
        }
    }while(pilihan!=0);
}
```

```
1. Insert Queue
2. Delete Queue
3. Cetak Data
0. Exit
Masukan Pilihan : 1
Masukkan nama : shofi
Masukkan npm : 57
nama : shofi
npm : 57
Press any key to continue . . .
```

```
1. Insert Queue
2. Delete Queue
3. Cetak Data
0. Exit
Masukan Pilihan : 1
Masukkan nama : sarah
Masukkan npm : 68
nama : shofi
npm : 57
nama : sarah
npm : 68
Press any key to continue . . .
```

```
1. Insert Queue
2. Delete Queue
3. Cetak Data
0. Exit
Masukan Pilihan : 2
nama : shofi
npm : 57
Press any key to continue . . .
```

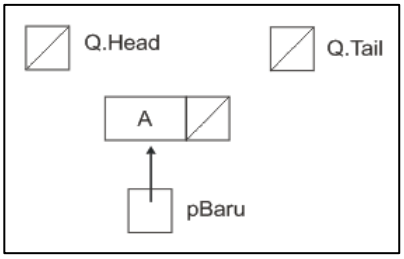
2.3. Operasi Stack dengan List Berkait

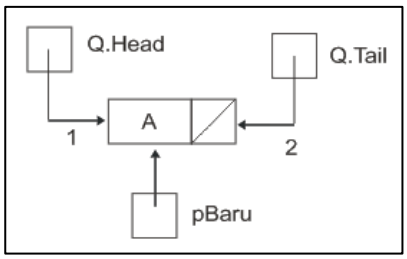
1. Jenis-jenis operasi

- Insert

  Ada 2 kasus yang harus ditangani yaitu :

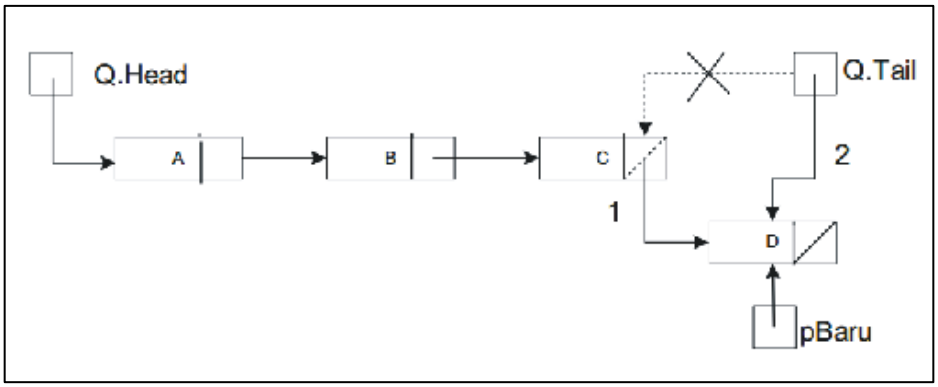
  1) Kasus Kosong





  2) Kasus ada isi

     Dalam hal ini tidak diperlukan lagi pencarian elemen terakhir karena elemen terakhir selalu dicatat oleh Q.Tail



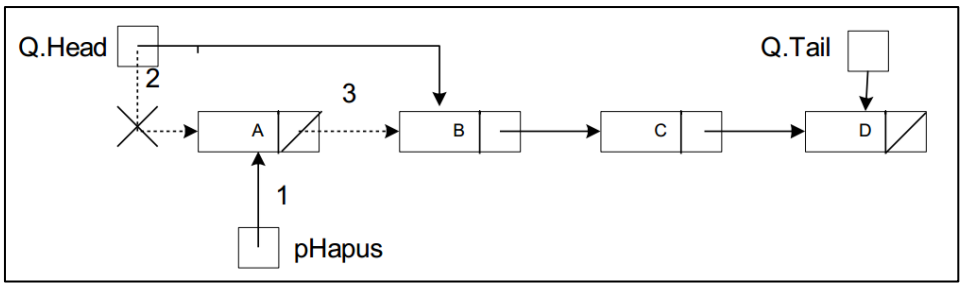
Codingan InsertQueue:

```
void insertQueue(Queue& Q, pointer pBaru){
    if (Q.Head==NULL && Q.Tail==NULL) {
        Q.Head = pBaru;
        Q.Tail = pBaru;
    }
    else {
        Q.Tail->next = pBaru;
        Q.Tail = pBaru;
    }
}
```

- Delete

  Ada 3 kasus yang harus ditangani yaitu:

  1) Queue kosong: Ciri dari Queue kosong adalah Q.Head==NULL dan Q.Tail==NULL. Tidak ada elemen yang dihapus.
  2) Queue dengan isi 1 elemen: list akan menjadi kosong dengan keadaan Head dan Tail bernilai NULL
  3) Queue dengan isi lebih dari 1 elemen



Codingan DeleteQueue:

```
void deleteQueue(Queue& Q, pointer& pHapus){
    cout<<"Delete Queue"<<endl;
    if (Q.Head==NULL && Q.Tail==NULL) {
        pHapus=NULL;
        cout<<"List Queue kosong "<<endl;
    }

    else if (Q.Head->next==NULL) {
        pHapus=Q.Head;
        Q.Head=NULL;
        Q.Tail=NULL;
    }

    else {
        pHapus=Q.Head;
        Q.Head=Q.Head->next;
        pHapus->next=NULL;
    }
}
```

2. Contoh pengimplementasian

- Daftar Mahasiswa

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

struct Mahasiswa {
    char nama[20];
    char npm[20];
    Mahasiswa* next;
};

typedef Mahasiswa* pointer;
typedef pointer List;

struct Queue {
    List Head;
    List Tail;
};
Queue Q;

void many(int& n){
    cout<<"Input Jumlah Mahasiswa: ";
    cin>>n;
}
```

```
void CreateQueue(Queue& Q) {
    Q.Head = NULL;
    Q.Tail = NULL;
}

void CreateElmt(pointer& pBaru) {
    pBaru = new Mahasiswa;
    cout << "Nama : "; cin >> pBaru->nama;
    cout << "NPM  : "; cin >> pBaru->npm;
    pBaru->next = NULL;
}

void insertQueue(Queue& Q, pointer pBaru){
    if (Q.Head==NULL && Q.Tail==NULL) {
        Q.Head = pBaru;
        Q.Tail = pBaru;
    }
    else {
        Q.Tail->next = pBaru;
        Q.Tail = pBaru;
    }
}
```

```
void deleteQueue(Queue& Q, pointer& pHapus){
    cout<<"Delete Queue"<<endl;
    if (Q.Head==NULL && Q.Tail==NULL) {
        pHapus=NULL;
        cout<<"List Queue kosong "<<endl;
    }

    else if (Q.Head->next==NULL) {
        pHapus=Q.Head;
        Q.Head=NULL;
        Q.Tail=NULL;
    }

    else {
        pHapus=Q.Head;
        Q.Head=Q.Head->next;
        pHapus->next=NULL;
    }
}
```

```
void traversal(Queue Q){
    pointer pBantu;
    if (Q.Head==NULL) {
        cout << "List kosong "<<endl;
    }
    else {
    cout<<"------------------"<<endl;
    cout<<"|  Nama  |  NPM  |"<<endl;
    cout<<"------------------"<<endl;
    pBantu=Q.Head;
        do {
        cout <<"| "<<pBantu->nama<<"  | ";
        cout << pBantu->npm <<" | "<<endl;
        pBantu = pBantu->next;
        } while(pBantu != Q.Tail->next);
    }
    cout<<"------------------"<<endl;
}
```

```
main(){
    int n;
    pointer a, h;

    many(n);
    cout<<endl;
    CreateQueue(Q);
    for(int i=0; i<n; i++){
        cout<<"Masukkan data mahasiswa ke-"<<i+1<<": "<<endl;
        CreateElmt(a);
        insertQueue(Q,a);
        cout<<endl;
    }
    traversal(Q);
    cout<<endl;

    cout<<"Menampilkan fungsi Delete Queue"<<endl;
    deleteQueue(Q, h);
    traversal(Q);
}
```

```
Input Jumlah Mahasiswa: 4

Masukkan data mahasiswa ke-1:
Nama : Angga
NPM  : 16001

Masukkan data mahasiswa ke-2:
Nama : Dwisa
NPM  : 16002

Masukkan data mahasiswa ke-3:
Nama : Reinh
NPM  : 16003

Masukkan data mahasiswa ke-4:
Nama : Drani
NPM  : 16004
```

```
------------------
|  Nama   |  NPM  |
------------------
| Angga   | 16001 |
| Dwisa   | 16002 |
| Reinh   | 16003 |
| Drani   | 16004 |
------------------

Menampilkan fungsi Delete Queue
Delete Queue
------------------
|  Nama   |  NPM  |
------------------
| Dwisa   | 16002 |
| Reinh   | 16003 |
| Drani   | 16004 |
------------------

Process returned 0 (0x0)   execution time : 41.089 s
Press any key to continue.
```

2.4. Circular Queue

Terlihat dalam operasi yang biasa ada ketidakefisienan ketika dilakukan proses delete yaitu selalu terjadi proses pergeseran elemen. Untuk itu dilakukan pengaturan penyimpanan array dengan bentuk circular (memutar) untuk menghindari ketidakefisienan ketika proses delete tersebut. Dengan bentuk memutar ini maka elemen pertama larik terletak berdekatan dengan elemen terakhir larik.

1. Deklarasi Circular Queue

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

const int maxElemen = 255;

struct Queue{
    char isi[maxElemen];
    int head;
    int tail;
};

void createQueue(Queue& Q){
Q.head = -1;
Q.tail = -1;
}
```

2. Insert Queue

   Ada 2 kasus yang harus ditangani yaitu:

1) Queue kosong
2) Queue berisi
    - Ada pengamanan pada posisi Q.tail ke posisiTemp sebelum dilakukan proses circular
    - Ketika queue penuh, Q.tail dikembalikan pada posisiTemp

```
void insertQueue(Queue& Q, char elemen){
    int posisiTemp;
    if (Q.tail==-1){
        Q.head=0;
        Q.tail=0;
        Q.isi[Q.tail] = elemen;
    }
    else{
        posisiTemp=Q.tail;
        if(Q.tail<maxElemen-1) Q.tail=Q.tail+1;
        else{
            Q.tail=0;
        }
        if (Q.tail==Q.head){
            cout<<"Antrian sudah penuh"<<endl;
            Q.tail=posisiTemp;
        }
        else {
        Q.isi[Q.tail] = elemen;
        }
    }
}
```

3. Delete Queue

Ada tiga kasus yang harus ditangani:

1) Queue kosong, tidak ada yang dihapus
2) Queue berisi 1 elemen
3) Queue berisi >1 elemen

```
void deleteQueue(Queue& Q, char& elemenHapus){
    if (Q.head==-1) cout<<"Antrian kosong"<<endl;
    else if (Q.head==Q.tail){
        elemenHapus= Q.isi[Q.head];
        Q.isi[Q.head]=' ';
        Q.head=-1;
        Q.tail=-1;
    }
    else{
        elemenHapus= Q.isi[Q.head];
        Q.isi[Q.head]=' ';
        if (Q.head<maxElemen-1) Q.head=Q.head+1;
        else Q.head=0;
    }
}
```

2.5. Priority Queue

Priority Queue digunakan untuk antrian yang beberapa elemennya memiliki prioritas yang lebih tinggi daripada urutan yang seharusnya atau bisa disebut juga dengan jalur VIP.

Prosedurnya:

- Elemen P dengan prioritas tertentu (10 - tertinggi, 1 - terendah) akan menyisip sebelum elemen dengan prioritas yang lebih rendah darinya.
- Jika prioritas sama maka P akan menyisip tepat dibelakang deretan elemen yang sama prioritasnya
- P akan menyisip dibelakang antrian jika prioritasnya paling kecil atau sama dengan prioritas Tail

Deklarasi Priority Queue:

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

struct elqueue{
    char info;
    int prior;
    elqueue* next;
};
typedef elqueue* pointer;
typedef pointer list;

struct queue{
    list Head;
    list Tail;
};
queue Q;
```

Membuat queue dengan list berkait:

```
void createList(queue& Q){
    Q.Head = NULL;
    Q.Tail = NULL;
}
```

```
void createElement(pointer& pBaru){
    pBaru = new elqueue;
    cout<<"Masukkan karakter: ";
    cin>>pBaru->info;
    do{
        cout<<"Masukkan prioritas: ";
        cin>>pBaru->prior;
        if (pBaru->prior<1||pBaru->prior>= 10){
            cout<<"ERROR Masukkan Angka 1-10"<<endl<<endl;
        }
    }
    while(pBaru->prior<=1||pBaru->prior>=10);
    pBaru->next=NULL;
}
```

Insert Queue:

```
void insertQueue(queue& Q,pointer pBaru){
    if (Q.Head==NULL && Q.Tail==NULL){
        Q.Head= pBaru;
    }
    else if (Q.Head->next == NULL){
        if (pBaru->prior>Q.Head->prior){
            pBaru->next=Q.Head;
            Q.Head=pBaru;
        }
        else{
            Q.Head->next = pBaru;
            Q.Tail = pBaru;
        }
    }
```

```
    else{
        pointer pBantu;
        pointer pBefore;
        pBantu=Q.Head;
        while(pBantu->prior>=pBaru->prior&&pBantu->next!=NULL){
            pBefore=pBantu;
            pBantu=pBantu->next;
        }
        if (pBantu==Q.Head){
            pBaru->next=pBantu;
            Q.Head=pBaru;
        }
        else if(Q.Tail->prior>=pBaru->prior){
            Q.Tail->next=pBaru;
            Q.Tail=pBaru;
        }
        else{
            pBaru->next=pBefore->next;
            pBefore->next=pBaru;
        }
    }
}
```

Delete Queue:

```
void deleteQueue(queue& Q, pointer& pHapus){
    if (Q.Head==NULL && Q.Tail==NULL){
        pHapus=NULL;
        cout<<"List Queue kosong"<<endl;
    }
    else if (Q.Head->next==NULL) {
        pHapus=Q.Head;
        Q.Head=NULL;
        Q.Tail=NULL;
    }
    else{
        pHapus=Q.Head;
        Q.Head=Q.Head->next;
        pHapus->next=NULL;
    }
}
```

Traversal:

```
void traversal(queue Q){
    if (Q.Head==NULL){
        "Data kosong tidak ada yang dihapus";
    }
    else{
        pointer pBantu;
        pBantu=Q.Head;
        cout<<"Info\tPrioritas"<<endl;
        cout<<"======================"<<endl;
        do{
            cout<<pBantu->info;
            cout<<"\t";
            cout<<pBantu->prior<<endl;
            pBantu=pBantu->next;
        }while(pBantu!=NULL);
    }
}
```

Main:

```
int main(){
    queue Q;
    pointer p;
    int n;
    char pilih;

    cout<<"P R I O R I T Y  Q U E U E"<<endl;
    cout<<"-------------------------"<<endl;

    createList(Q);
    cout<<"Masukkan banyak data: ";
    cin>>n;
    cout<<endl;
```

```
    for (int i=0;i<n;i++){
        createElement(p);
        insertQueue(Q,p);
    }
    cout<<endl;
    traversal(Q);

    cout<<endl<<endl;
    do{
        cout<<"Setelah Delete Queue :"<<endl;
        deleteQueue(Q,p);
        traversal(Q);
        cout<<endl;
        cout<<"Apakah ingin melakukan Delete Queue lagi (y/n)?";
        cin>>pilih;

    } while(pilih=='y'||pilih=='Y');
}
```

Compile:

```
P R I O R I T Y  Q U E U E
--------------------------
Masukkan banyak data: 5

Masukkan karakter: t
Masukkan prioritas: 3
Masukkan karakter: b
Masukkan prioritas: 2
Masukkan karakter: y
Masukkan prioritas: 7
Masukkan karakter: n
Masukkan prioritas: 6
Masukkan karakter: p
Masukkan prioritas: 4

Info    Prioritas
==================
y       7
n       6
p       4
t       3
b       2


Setelah Delete Queue :
Info    Prioritas
==================
n       6
p       4
t       3
b       2

Apakah ingin melakukan Delete Queue lagi (y/n)?n


Terminated with return code 0
Press any key to continue ...
```

BAB III
PENUTUP

3.1. Kesimpulan

Queue pada Struktur Data atau antrean adalah sekumpulan data yang mana penambahan elemen hanya bisa dilakukan pada suatu ujung disebut dengan sisi belakang (rear), dan penghapusan (pengambilan elemen) dilakukan lewat ujung lain (disebut dengan sisi depan atau front). Pada queue, prinsip yang digunakan adalah "Masuk Pertama Keluar Pertama" atau FIFO (First In First Out).

Queue memiliki karakteristik yaitu elemen antrian, head (elemen terdepan), tail (elemen terakhir), jumlah elemen pada antrian, dan status antrian. Cara mendefinisikan queue ada dua cara, yaitu menggunakan array dan list berkait. Yang membedakan keduanya adalah pada isi fungsi operasinya dimana jenis operasi dari queue sendiri terdiri dari insert dan delete.

Queue memiliki dua jenis khusus, yang pertama adalah circular queue dimana antrean tersebut membantu mengatasi masalah ketidakefisienan ketika dilakukan proses delete yaitu selalu terjadi proses pergeseran elemen. Dengan bentuk memutar ini maka elemen pertama larik terletak berdekatan dengan elemen terakhir larik. Kedua adalah priority queue dimana beberapa elemen antrean memiliki prioritas yang lebih tinggi sehingga pengurutan dilakukan berdasarkan prioritasnya baru urutan penambahan elemen.

3.2. Saran

Pembelajaran tentang queue ini masih harus lebih diperdalam lagi agar dapat diterapkan untuk contoh-contoh lain.

DAFTAR PUSTAKA

http://blog-arul.blogspot.co.id/2012/01/queue-pada-struktur-data.html

Contoh Makalah.docx

10. Queue.pdf