Informasi dan Struktur Data

Modul 5 – Queue (Antrian)

Tujuan:

1. Praktikan dapat memahami konsep Queue (Antrian) secara Sirkular maupun yang tidak Sirkular
2. Praktikan dapat mengimplementasikan konsep Queue (Antrian) ke dalam Bahasa Pemrograman C

Pembahasan:

Queue (Antrian) adalah sebuah struktur data linier tersusun serta terbuka pada kedua ujugnya yang membentuk seperti antrian. Struktur data queue ini memiliki konsep **FIFO** (First In First Out) artinya data yang masuk pertama maka akan keluar pertama juga. Di dalam struktur data queue ini memiliki 2 penanda sebagai ujungnya yaitu sebuah kepala (Head) sebagai penanda bahwa data yang pertama dan sebuah ekor (Tail) sebagai data yang terakhir.

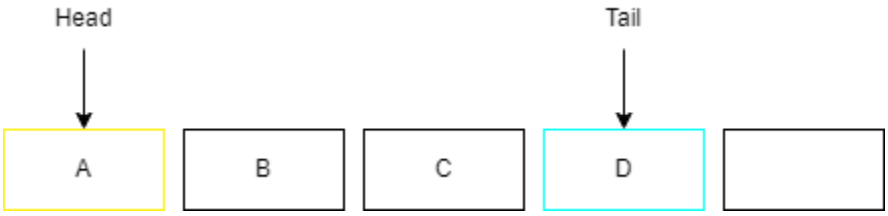
Hal-hal dasar Queue:

1. **Inisialisasi** Queue dalam kondisi awal **Kosong**
2. **Penambahan** data akan dilakukan pada posisi Tail
3. **Penghapusan** data akan dilakukan pada posisi Head

4. Pengecekan apakah antrian **sudah penuh** atau **masih kosong**
5. **Traversal** untuk melakukan **Print** dan **Pencarian Data**

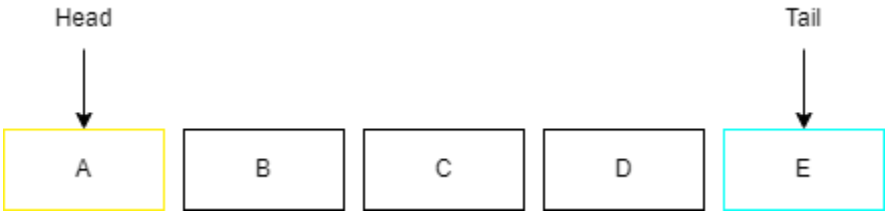
Ilustrasi & Pembahasan Singkat Queue:

1. **Kondisi awal**



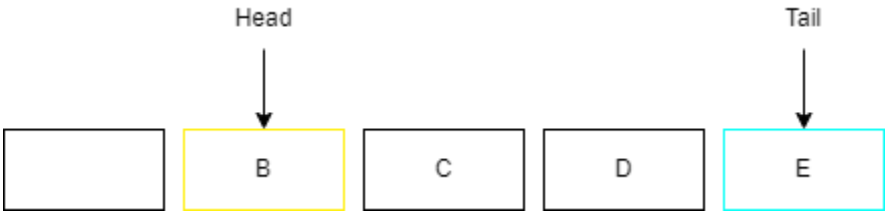
Kondisi awal pada Queue tersebut dapat di simpulkan bahwa data yang pertama adalah A dan data yang terakhir adalah D, serta Queue tersebut masih terdapat 1 slot

2. **Penambahan Data**



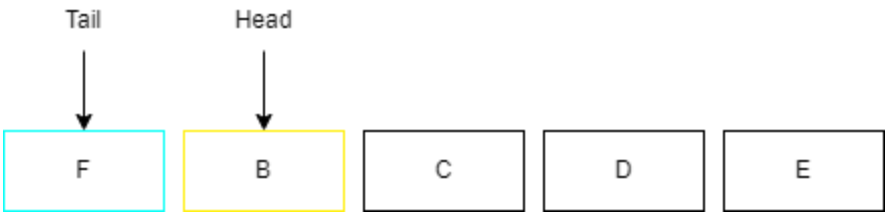
Pada kondisi di atas terjadi penambahan data dalam Queue. Data yang ditambahkan adalah **E**, saat penambahan sebuah data maka posisi **Tail** akan bergeser ke arah kanan. Penambahan data pada Queue diatas merupakan Queue normal.

3. **Penghapusan Data**



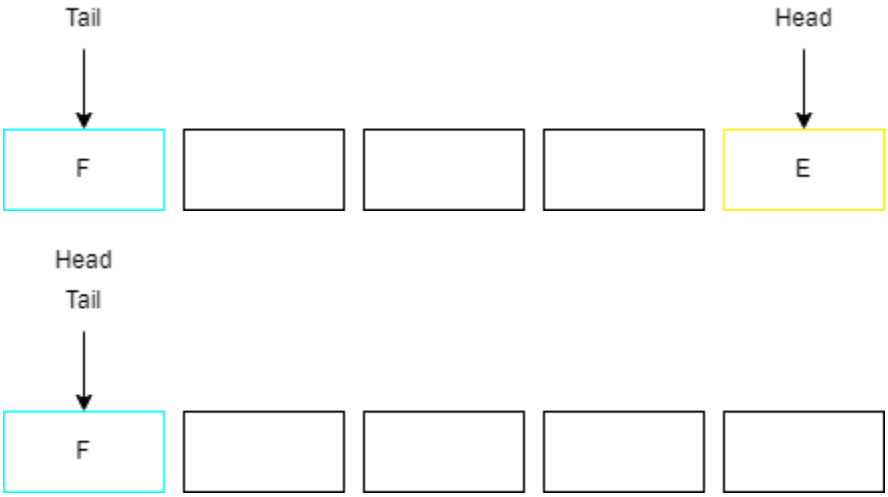
Kemudian kondisi berikutnya dapat dilihat jika terjadi penghapusan data pada Queue, data yang terhapus pada Queue tersebut adalah data yang sedang berada di posisi Head sebelumnya yaitu (A). Setelah data berhasil di hapus maka Head akan bergeser ke arah selanjutnya yaitu kanan.

4. **Penambahan Data kedua**



Pada kondisi ini terjadi penambahan data pada Queue yaitu Data F, maka Tail akan mencari slot yang kosong kemudian akan terjadi penginputan data pada posisi Tail. Pada kondisi Sirkular, posisi Tail akan kembali ke bagian awal karena sudah mencapai Array maksimalnya.

5. **Penghapusan Data jika Head sudah sampai indeks akhir**



Pada kondisi ini terjadi jika posisi **Head** sudah berada di akhir indeks tetapi masih bisa melakukan **Dequeue** karena masih adanya data dalam Queue tersebut. pada kondisi ini, maka yang hanya perlu dilakukan adalah membuat posisi **Head** ke data paling awal yaitu array indeks 0.

Guided

- Header.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define max 5

typedef char string[50];

typedef struct{
    string nama, npm;
    float ipk;
}Data;

typedef struct{
    Data D[max];
    int Head; //penanda bagian terdepan sebuah Queue
    int Tail; //penanda bagian terakhir sebuah Queue
}Queue;

Data makeData(string nama, string npm, float ipk); //membuat sebuah element atau data dengan tipe data Data menggunakan Fungsi
void createEmpty(Queue *Q); //membuat Queue menjadi kosong
int isEmpty(Queue Q); //mengecek apakah Queue kosong
int isFull(Queue Q); //mengecek apakah Queue penuh
int isOneElmt(Queue Q); //mengecek apakah Queue hanya memiliki 1 Data
void Enqueue(Queue *Q, Data D); //memasukkan sebueah data ke dalam Queue
void Dequeue(Queue *Q); //menghapus sebuah data dari dalam Queue
void print(Queue Q); //menampilkan data-data yang berada dalam sebuah Queue
int isFound(Queue Q, string target); //mencari sebuah data yang ingin dicari dalam Queue
```

- Source.c

```c
#include "header.h"

Data makeData(string nama, string npm, float ipk){ //make data untuk memenuhi syarat ADT
    Data data;

    strcpy(data.nama, nama);
    strcpy(data.npm, npm);
    data.ipk = ipk;

    return data;
}

void createEmpty(Queue *Q){ //membuat penanda Head dan Tail menunjuk pada arah -1 / kosong
    (*Q).Head = (*Q).Tail = -1;
}

int isEmpty(Queue Q){ //mengecek apakah isi Queue kosong atau tidak
    return (Q.Head == -1 && Q.Tail == -1);
}

int isFull(Queue Q){ //mengecek apakah isi Queue penuh atau tidak
    return ((Q.Head < Q.Tail && Q.Tail - Q.Head == max-1) || (Q.Tail < Q.Head && Q.Tail + 1 == Q.Head));
}

int isOneElmt(Queue Q){ //mengecek apakah isi Queue hanya 1 elemen atau tidak
    return (Q.Head == Q.Tail && !isEmpty(Q));
}

void Enqueue(Queue *Q, Data D){
    if(isEmpty(*Q)){ //jika Queue masih kosong maka Head dan Tail langsung diarahkan ke indeks 0
        (*Q).Head = (*Q).Tail = 0;
    } else{
        if((*Q).Tail == max-1)  { //kondisi jika Tail sudah mencapai akhir indeks tetapi bisa menambah data
            (*Q).Tail = 0;
        }else{ //kondisi Queue normal
            (*Q).Tail++;
        }
    }
    (*Q).D[(*Q).Tail] = D; //input Data ke dalam Queue

    printf("\n\t BERHASIL MEMASUKKAN DATA");;
}
```

```
44  void Dequeue(Queue *Q){
45      string temp;
46
47      strcpy(temp, (*Q).D[(*Q).Head].npm); //sebagai penampung sebelum data dihapus
48
49      if(isOneElmt(*Q)){ //jika hanya ada 1 data/elemen maka langsung create empty saja
50          createEmpty(&(*Q));
51      } else{
52          if((*Q).Head == max-1){ //kondisi jika Head sudah mencapai akhir indeks dan bisa melakukan penghapusan data
53              (*Q).Head = 0;
54          } else{ //kondisi normal
55              (*Q).Head++;
56          }
57      }
58
59      printf("\n\t Data dengan NPM %s Berhasil Dihapus",temp);
60  }
61
62  void print(Queue Q){
63      int i, temp=1;
64
65      if(Q.Head <= Q.Tail){ //kondisi normal
66          for(i=Q.Head; i<=Q.Tail; i++){
67              printf("%d. %s(%.2f); ", i+1, Q.D[i].nama, Q.D[i].ipk);
68          }
69      }else{ //kondisi circular akan melakukan print secara bertahap
70          for(i=Q.Head; i<=max-1; i++){ //pertama akan print dari posisi Head hingga akhir dari indeks
71              printf("%d. %s(%.2f); ", temp, Q.D[i].nama, Q.D[i].ipk);
72              temp++;
73          }
74
75          for(i=0; i<Q.Tail; i++){ //kedua akan print dari indeks 0 hingga posisi Tail
76              printf("%d. %s(%.2f); ", temp, Q.D[i].nama, Q.D[i].ipk);
77              temp++;
78          }
79      }
80  }
```

```
82  int isFound(Queue Q, string target){
83      int i;
84      if(Q.Head <= Q.Tail){ //kondisi normal
85          for(i=Q.Head; i<=Q.Tail; i++){
86              if(strcmp(Q.D[i].npm, target)==0){ //jika dapat datanya akan mereturnkan indeks
87                  return i;
88              }
89          }
90      }else{
91          for(i=Q.Head; i<=max-1; i++){
92              if(strcmp(Q.D[i].npm, target)==0){
93                  return i;
94              }
95          }
96
97          for(i=0; i<Q.Tail; i++){
98              if(strcmp(Q.D[i].npm, target)==0){
99                  return i;
100             }
101         }
102     }
103     return -1;
104 }
```

- **Main.c**

```c
#include "header.h"

int main(int argc, char *argv[]) {
    Queue Q;
    int menu, index;
    Data temp;

    createEmpty(&Q);

    do{
        system("cls");
        printf("[1]. Enqueue/Tambah Data");
        printf("\n[2]. Dequeue/Hapus Data");
        printf("\n[3]. Show");
        printf("\n[4]. Cari Data");
        printf("\n[0]. Keluar");
        printf("\n>>> "); scanf("%d", &menu);

        switch(menu){
            case 1:
                if(!isFull(Q)){
                    printf("\n\tInput Data");
                    printf("\n\nNama: "); fflush(stdin); gets(temp.nama);
                    printf("NPM : "); fflush(stdin); gets(temp.npm);
                    printf("IPK : "); scanf("%f", &temp.ipk);

                    Enqueue(&Q, makeData(temp.nama, temp.npm, temp.ipk));
                }else{
                    printf("\n\t[!] Queue sudah penuh [!]");
                }
                break;

            case 2:
                if(!isEmpty(Q)){
                    Dequeue(&Q);
                }else{
                    printf("\n\t[!] Queue masih kosong [!]");
                }
                break;
```

```
            case 3:
                if(!isEmpty(Q)){
                    print(Q);
                }else{
                    printf("\n\t[!] Queue masih kosong [!]");
                }
                break;

            case 4:
                if(!isEmpty(Q)){
                    printf("\n\tPencarian Data");
                    printf("\n\nMasukkan NPM yang ingin dicari: "); fflush(stdin); gets(temp.npm);
                    index = isFound(Q, temp.npm);
                    if(index!=-1){
                        printf("\n\t Data yang ditemukan!!!");
                        printf("\n\t\tNama : %s", Q.D[index].nama);
                        printf("\n\t\tNPM  : %s", Q.D[index].npm);
                        printf("\n\t\tIPK  : %.2f", Q.D[index].ipk);
                    }else{
                        printf("\n\t[!] Data yang dicari tidak ditemukan [!]");
                    }
                }else{
                    printf("\n\t[!] Queue masih kosong [!]");
                }
                break;

            case 0:
                printf("\n\n\tCODING ITU MENYENANGKAN :)");
                break;

            default:
                printf("\n\t[!] Menu tidak tersedia [!]");
                break;
        }
        getch();
    }while(menu!=0);
    return 0;
}
```

Penjelasan Code & Ilustrasi

- **Inisialisasi**

```
void createEmpty(Queue *Q){ //membuat penanda Head dan Tail menunjuk pada arah -1 / kosong
    (*Q).Head = (*Q).Tail = -1;
}
```

Code di atas adalah untuk menginisialisasi sebuah Queue. Inisialisasi Queue dilakukan dengan cara set posisi **Head** dan **Tail** menjadi -1 (karena index array dimulai dari 0).

- **isEmpty**

```
int isEmpty(Queue Q){ //mengecek apakah isi Queue kosong atau tidak
    return (Q.Head == -1 && Q.Tail == -1);
}
```

Code diatas adalah untuk mengecek apakah sebuah Queue kosong atau tidak. Queue dianggap kosong jika **Head** dan **Tail** berada di posisi -1, sehingga pada code tersebut akan mereturnkan 1 jika kosong dan mereturnkan 0 jika tidak kosong.

- **isFull**

```
int isFull(Queue Q){ //mengecek apakah isi Queue penuh atau tidak
    return ((Q.Head < Q.Tail && Q.Tail - Q.Head == max-1) || (Q.Tail < Q.Head && Q.Tail + 1 == Q.Head));
}
```

Code di atas terdapat **OR** yang menandakan terdapat 2 kondisi pengecekan untuk melakukan return.

Kondisi Normal (Gambar no.1 Ilustrasi & Pembahasan Singkat)

Sesuai dengan Code beserta gambar ilustrasi diatas dapat dilihat bahwa pada kondisi normal ini, indeks posisi **Head** akan lebih kecil daripada indeks posisi **Tail.** sehingga didapat code `Q.Head < Q.Tail && Q.Tail - Q.Head == max-1.`

Kondisi Sirkular (Gambar no.3 Ilustrasi & Pembahasan Singkat)

Sesuai dengan code beserta gambar ilustrasi diatas dapat dilihat bahwa pada kondisi normal ini, indeks posisi **Tail** akan lebih kecil daripada indeks posisi **Head**. Sehingga didapat code `Q.Tail < Q.Head && Q.Tail + 1 == Q.Head`

- **isOneElement (Gambar no.4 Ilustrasi & Pembahasan Singkat)**

```
int isOneElmt(Queue Q){ //mengecek apakah isi Queue hanya 1 elemen atau tidak
    return (Q.Head == Q.Tail && !isEmpty(Q));
}
```

Code di atas akan mengecek apakah di dalam Queue tersebut hanya terdapat 1 data/element atau tidak. Kondisi Queue 1 element adalah saat indeks posisi **Head** dan **Tail** sama dan kondisi Queue tidak kosong.

- **Enqueue**

```
void Enqueue(Queue *Q, Data D){
    if(isEmpty(*Q)){ //jika Queue masih kosong maka Head dan Tail langsung diarahkan ke indeks 0
        (*Q).Head = (*Q).Tail = 0;
    } else{
        if((*Q).Tail == max-1)  { //kondisi jika Tail sudah mencapai akhir indeks tetapi bisa menambah data
            (*Q).Tail = 0;
        }else{ //kondisi Queue normal
            (*Q).Tail++;
        }
    }
    (*Q).D[(*Q).Tail] = D; //input Data ke dalam Queue

    printf("\n\t BERHASIL MEMASUKKAN DATA");;
}
```

Dari code di atas, terdapat beberapa kondisi saat ingin melakukan Enqueue/penambahan data pada sebuah Queue.

Kondisi Kosong

Saat Queue masih kosong kemudian ingin melakukan Enqueue, maka yang hanya perlu dilakukan adalah set posisi **Head** dan **Tail** menjadi 0 karena indeks 0 merupakan data pertama dari sebuah array.

Kondisi Normal (Gambar no.4 Ilustrasi & Pembahasan Singkat)

Pada kondisi normal ini yang hanya perlu dilakukan hanya menggeser posisi **Tail** ke arah kanan dengan cara menambah indeks posisinya.

Kondisi Jika Tail Sudah di Indeks Akhir (Gambar no.2 Ilustrasi & Pembahasan Singkat)

Pada kondisi Tail yang sudah berada di Indeks akhir tetapi masih bisa malakukan penambahan data (Belum full) maka yang harus dilakukan adalah set indeks posisi Tail menjadi posisi awal yaitu indeks 0.

- **Dequeue**

```
void Dequeue(Queue *Q){
    string temp;

    strcpy(temp, (*Q).D[(*Q).Head].npm); //sebagai penampung sebelum data dihapus

    if(isOneElmt(*Q)){ //jika hanya ada 1 data/elemen maka langsung create empty saja
        createEmpty(&(*Q));
    } else{
        if((*Q).Head == max-1){ //kondisi jika Head sudah mencapai akhir indeks dan bisa melakukan penghapusan data
            (*Q).Head = 0;
        } else{ //kondisi normal
            (*Q).Head++;
        }
    }

    printf("\n\t Data dengan NPM %s Berhasil Dihapus",temp);
}
```

Dari code di atas, terdapat beberapa kondisi saat ingin melakukan Dequeue/penghapusan data pada sebuah Queue.

Kondisi hanya 1 element (Gambar no.4 Ilustrasi & Pembahasan Singkat)

Pada kondisi ini hanya terdapat 1 data pada sebuah array, dan jika ingin melakukan penghapusan data maka Queue akan langsung kosong. Jadi, pada kondisi ini hanya perlu memanggil fungsi `CreateEmpty` yang telah dibuat sebelumnya.

Kondisi normal (Gambar no.1 Ilustrasi & Pembahasan Singkat)

Pada kondisi ini adalah kondisi normal jika indeks posisi **Head** belum mencapai akhir batas indeks. Yang perlu dilakukan pada kondisi ini adalah menggeser posisi **Head** kearah kanan dengan menambah indeks posisi **Head.**

Kondisi Head sudah di Indeks Akhir (Gambar no.4 Ilustrasi & Pembahasan Singkat)

Pada kondisi ini adalah kondisi jika indeks posisi **Head** sudah mencapai batas akhir dari indeks array. Yang perlu dilakukan pada kondisi ini jika ingin menghapus data adalah set indeks posisi **Head** menjadi posisi awal yaitu indeks posisi 0

- **Print (Traversal)**

```
62  void print(Queue Q){
63      int i, temp=1;
64
65      if(Q.Head <= Q.Tail){ //kondisi normal
66          for(i=Q.Head; i<=Q.Tail; i++){
67              printf("%d. %s(%.2f); ", i+1, Q.D[i].nama, Q.D[i].ipk);
68          }
69      }else{ //kondisi circular akan melakukan print secara bertahap
70          for(i=Q.Head; i<=max-1; i++){ //pertama akan print dari posisi Head hingga akhir dari indeks
71              printf("%d. %s(%.2f); ", temp, Q.D[i].nama, Q.D[i].ipk);
72              temp++;
73          }
74
75          for(i=0; i<Q.Tail; i++){ //kedua akan print dari indeks 0 hingga posisi Tail
76              printf("%d. %s(%.2f); ", temp, Q.D[i].nama, Q.D[i].ipk);
77              temp++;
78          }
79      }
80  }
```

Dari code di atas terdapat 2 kondisi traversal saat ingin melakukan print data yaitu kondisi normal dan kondisi sirkular.

Kondisi Normal (Gambar no.1 Ilustrasi & Pembahasan Singkat)

Kondisi ini adalah kondisi yang normal sehingga traversal akan dilakukan secara normal yaitu dari Indeks posisi **Head** hingga sama dengan indeks posisi **Tail.**

Kondisi Sirkular (Gambar no.3 Ilustrasi & Pembahasan Singkat)

Kondisi ini adalah kondisi yang Sirkular (Tail<Head). Pada kondisi ini traversal akan dilakukan 2 kali. Traversal yang pertama akan dilakukan dari **Head** hingga batas akhir array dan kemudian traversal yang kedua akan dilakukan dari posisi awal data (indeks 0) hingga posisi sama dengan **Tail.**

- **isFound (Traversal)**

```
82  int isFound(Queue Q, string target){
83      int i;
84      if(Q.Head <= Q.Tail){ //kondisi normal
85          for(i=Q.Head; i<=Q.Tail; i++){
86              if(strcmp(Q.D[i].npm, target)==0){ //jika dapat datanya akan mereturnkan indeks
87                  return i;
88              }
89          }
90      }else{
91          for(i=Q.Head; i<=max-1; i++){
92              if(strcmp(Q.D[i].npm, target)==0){
93                  return i;
94              }
95          }
96
97          for(i=0; i<Q.Tail; i++){
98              if(strcmp(Q.D[i].npm, target)==0){
99                  return i;
100             }
101         }
102     }
103     return -1;
104 }
```

Kode Traversal diatas sama seperti Traversal Print, yang berbeda hanya pada print akan melakukan print sedangkan `isFound` akan mereturnkan indeks dimana data ditemukan

Ketentuan & Format Pengumpulan:

1. Comment pada code hanya sebagai bantuan jadi tidak dimasukkan tidak masalah
2. Pembuatan Guided harus dengan pembagian 3 File yaitu header, main, dan source
3. Diharap Pengerjaan Guided dilakukan secara sendiri, tidak melakukan ***COPAS*** dari praktikan lainnya karena Guided akan sangat membantu saat Unguided dan Tugas.
4. Semua code dimasukkan ke dalam satu folder dengan penamaan Folder : GD4_X_YYYYY
5. Folder yang berisi code dizip dengan format penamaan GD4_X_YYYYY.zip
6. KET:
   - X = Kelas
   - YYYYY = 5 digit terakhir NPM