JOBSHEET 07

Queue/ Antrian

Fakultas	Proram Studi	Kode MK	Waktu
Teknik	Teknik Informatika	INF1.62.2014	2 x 50 Menit

TUJUAN PRAKTIKUM

1. Setelah mengikuti perkuliahan ini diharapkan mahasiswa
   - Mahasiswa mengerti konsep stack dan operasi pada queue.
   - Mahasiswa dapat menggunakan queue untuk memecahkan permasalahan pemrograman.

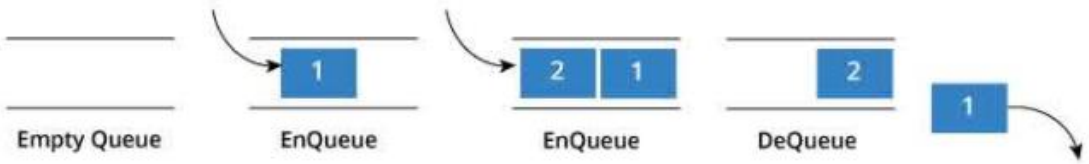
HARDWARE & SOFTWARE

1. Personal Computer
2. DevC++ IDE

TEORI SINGKAT

A. Konsep Queue

Sebuah antrian sangat berguna pada pemograman struktur data. Hal ini sama dengan membeli tiket pergi nonton ke bioskop. Dimana orang pertama yang memasuki antrian adalah yang pertama mendapat tiket masuk. Antrian menggunakan prinsip FIFO First in First Out. Item yang mendapatkan giliran pertama adalah item yang pertama keluar.



Pada gambar di atas, terdapat item dengan no 1 berada pada antrian sebelum item no 2, hal demikian menjadi yang pertama yang akan diambil dari antrian menurut prinsip FIFO. Dalam istilah pemograman,

meletakkan sebuah item ke dalam antrian disebut "enqueue" dan menghapusnya disebut "dequeue". Kita dapat mengimplementasikan antrian di bahasa pemograman apapun seperti C, C++, Java, Phyton atau C#, tapi spesifikasinya tetap.

B. Spesifikasi Queue

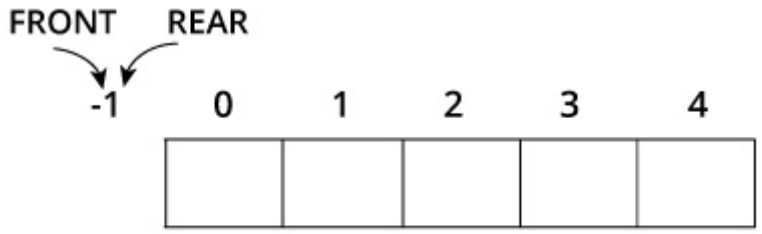
Sebuah antrian dinamakan juga dengan sebuah objek atau lebih spesifiknya ADT yang memiliki operasi sebgai berikut :

- Enqueue: Add element to end of queue
- Dequeue: Remove element from front of queue
- IsEmpty: Check if queue is empty
- IsFull: Check if queue is full
- Peek: Get the value of the front of queue without removing it

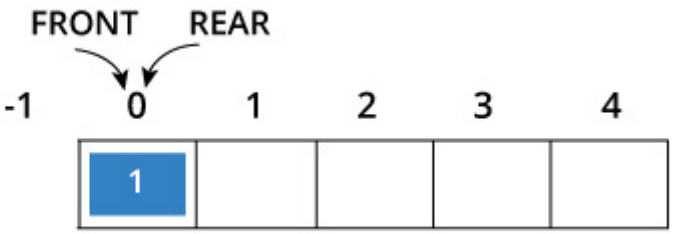
C. Cara kerja Queue

Sebuah antrian memiliki proses sebagai berikut :

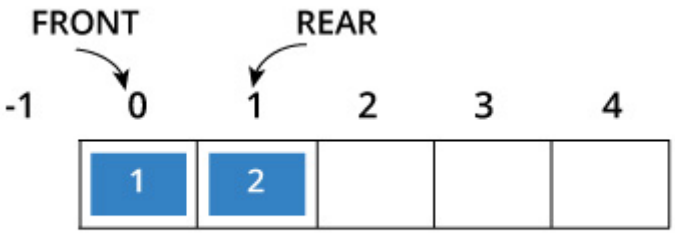
1. Dua pointer yang disebut FRONT dan REAR digunakan untuk melacak elemen pertama dan terakhir dalam antrian.
2. Saat menginisialisasi antrian, kami menetapkan nilai FRONT dan REAR ke -1.
3. Pada enqueing elemen, kita meningkatkan nilai indeks REAR dan menempatkan elemen baru di posisi yang ditunjuk oleh REAR.
4. Pada dequeueing suatu elemen, kita mengembalikan nilai yang ditunjuk oleh FRONT dan meningkatkan indeks FRONT.
5. Sebelum enqueing, kami memeriksa apakah antrian sudah penuh.
6. Sebelum dequeuing, kami memeriksa apakah antrian sudah kosong.
7. Saat membuat elemen pertama, kami menetapkan nilai FRONT ke 0.
8. Saat mendekor elemen terakhir, kita mereset nilai FRONT dan REAR ke -1.



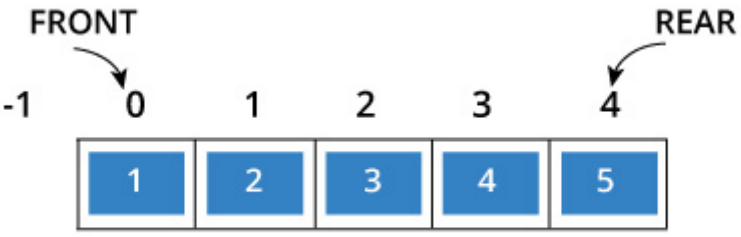
Empty Queue



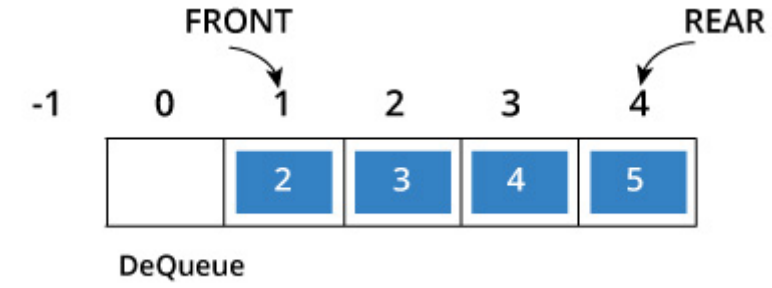
EnQueue first element



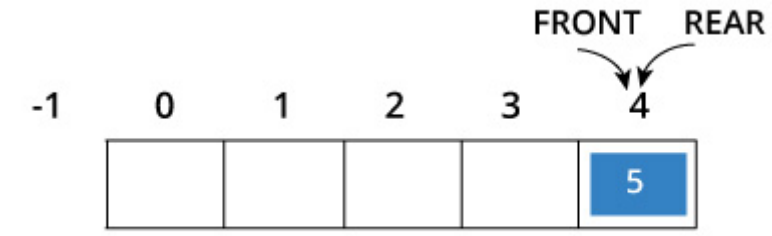
EnQueue



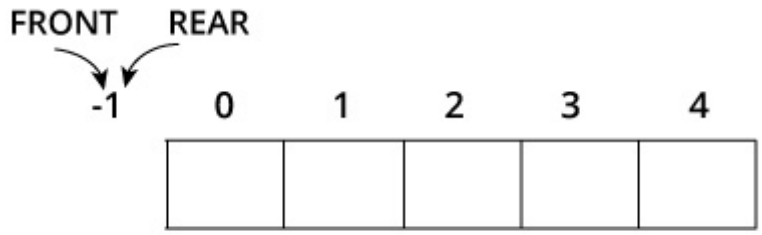
EnQueue



DeQueue



DeQueue last element



Empty Queue

Queue dengan Array

Pada implementasi antrian dengan array, digunakan sejumlah array MAX untuk menyimpan data. Untuk menunjuk bagian depan dan bagian belakang digunakan variable Front dan Rear. Bila antrian kosong, nilainya diset -1. Untuk operasi penambahan dan penghapusan diimplementasikan dua fungsi yaitu Tambah() dan Hapus().

Pada saat menambah elemen baru pada antrian, pertama kali dicek apakah penambahan tersebut dimungkinkan atau tidak. Karena indeks array dimulai dengan 0 maka maksimum data yang dapat disimpan pada antrian adalah MAX-1. Jika semua elemen menempati ruang array maka antrian dalam kondisi penuh. Apabila data masih dapat ditambahkan pada antrian maka variable Rear dinaikkan satu dan data baru disimpan pada array. Apabila data baru ditambahkan ke antrian untuk pertama kali (dimana variable Front bernilai -1) maka variable Front diset 0 yang menandakan antrian tidak lagi kosong.

Sebelum menghapus elemen dari antrian harus dipastikan apakah elemen tersedia untuk penghapusan. Jika tidak maka antrian dalam kondisi kosong. Sebaliknya bila tersedia data pada array maka dapat dilakukan penghapusan dan variable Front dinaikkan. Apabila nilai variable Front dan Rear sama (yang berarti antrian dalam keadaan kosong) maka Front dan Rear direset -1.

Misalnya akan dilakukan penambahan data pada antrian sampai semua array terisi. Pada kondisi ini nilai Rear menjadi MAX-1. Misalnya dilakukan penghapusan 5 elemen, maka antrian dikatakan dalam kondisi penuh meskipun 5 array pertama kosong. Untuk mengatasi kondisi ini maka diimplmentasikan antrian sebagai antrian sirkular (*circular queue*). Sehingga selama penambahan, jika sudah mencapai akhir array dan Jika awal array kosong (sebagai akibat dari penghapusan) maka elemen baru ditambahkan pada awal array.

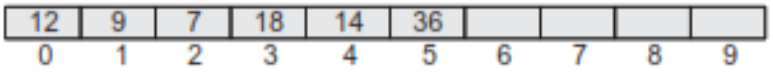


Figure 8.1 Queue

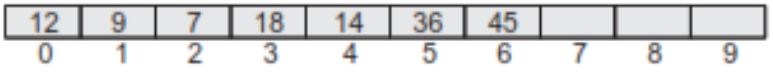


Figure 8.2 Queue after insertion of a new element

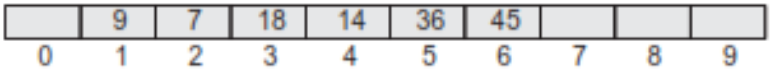


Figure 8.3 Queue after deletion of an element

Queue dengan Linked List

Array yang digunakan untuk mengimplementasikan antrian dideklarasikan mempunyai ukuran MAX. Ukuran ini ditentukan pada saat menulis program, tidak dapat diubah pada saat program berjalan. Sehingga pada saat menulis program, harus ditentukan ukuran memori maksimum yang diperlukan untuk membangun array. Hal ini berhugungan dengan deklarasi ruang memori yang tersedia. Jika jumlah elemen yang dapat disimpan dalam antrian kecil, maka banyak ruang memori yang tidak digunakan. Sebaliknya, jika jumlah elemen yang akan disimpan pada antrian terlalu banyak, maka menyebabkan overflow. Untuk menghindari hal tersebut terjadi dan keterbatasan memory maka digunakan struktur data yang disebut linked list.

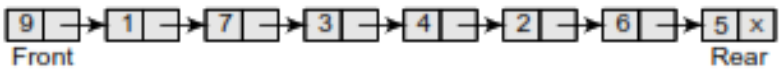


Figure 8.6 Linked queue

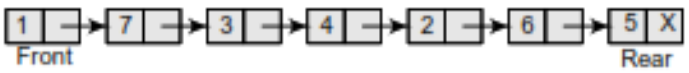
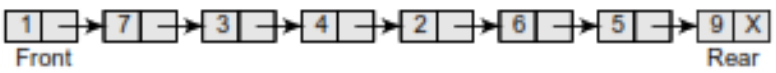


Figure 8.7 Linked queue



PERCOBAAN

1. Queue menggunakan array

```
#include<stdio.h>
#include<stdlib.h>
#define ukuran 5
void tambah();
void hapus();
void display();
int front = -1, rear = -1;
int queue[ukuran];

int main ()
{
    int pilihan;
    while(pilihan != 4)
    {
        printf("\n************************Main
Menu*****************************\n");

printf("\n=========================================================
=========\n");
        printf("\n1.Tambah sebuah elemen\n2.hapus sebuah
elemen\n3.Tampilkan antrian\n4.Keluar\n");
        printf("\nMasukan pilihanmu ?");
        scanf("%d",&pilihan);
        switch(pilihan)
        {
            case 1:
            tambah();
            break;
            case 2:
            hapus();
            break;
            case 3:
            display();
            break;
            case 4:
            exit(0);
            break;
            default:
            printf("\nMasukan pilihan yang sesuai?\n");
        }
    }
}
 void tambah()
{
    int item;
    printf("\nSilakan masukan elemen yang kamu inginkan\n");
    scanf("\n%d",&item);
    if(rear == ukuran-1)
    {
        printf("\nOVERFLOW\n");
        return;
    }
    if(front == -1 && rear == -1)
    {
```

```
        front = 0;
        rear = 0;
    }
    else
    {
        rear = rear+1;
    }
    queue[rear] = item;
    printf("\nNilai berhasil ditambahkan ");

}
void hapus()
{
    int item;
    if (front == -1 || front > rear)
    {
        printf("\nUNDERFLOW\n");
        return;

    }
    else
    {
        item = queue[front];
        if(front == rear)
        {
            front = -1;
            rear = -1 ;
        }
        else
        {
            front = front + 1;
        }
        printf("\nNilai berhasil dihapus ");
    }


}
void display()
{
    int i;
    if(rear == -1)
    {
        printf("\nAntrian kosong\n");
    }
    else
    {   printf("\nMenampilkan nilai .....\n");
        for(i=front;i<=rear;i++)
        {
            printf("\n%d\n",queue[i]);

        }
    }
}
```

2. Queue Menggunakan Linked List

```
#include<stdio.h>
#include<stdlib.h>
struct node
```

```
{
    int data;
    struct node *next;
};
struct node *front;
struct node *rear;
void tambah();
void hapus();
void display();

int main ()
{
    int menu;
    while(menu != 4)
    {
        printf("\n***************Queue Menggunakan Linked
List******************\n");

printf("\n=============================================================
==========\n");
        printf("\n1.Tambah sebuah elemen\n2.Hapus sebuah
elemen\n3.Tampilkan Antrian\n4.Keluar\n");
        printf("\nMasukan pilihanmu ?");
        scanf("%d",& menu);
        switch(menu)
        {
            case 1:
            tambah();
            break;
            case 2:
            hapus();
            break;
            case 3:
            display();
            break;
            case 4:
            exit(0);
            break;
            default:
            printf("\nMasukan pilihan menu yang sesuai?\n");
        }
    }
}
void tambah()
{
    struct node *ptr;
    int item;

    ptr = (struct node *) malloc (sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW\n");
        return;
    }
    else
    {
        printf("\nSilakan masukan elemen yang kamu inginkan\n");
        scanf("%d",&item);
```

```
            ptr -> data = item;
            if(front == NULL)
            {
                front = ptr;
                rear = ptr;
                front -> next = NULL;
                rear -> next = NULL;
            }
            else
            {
                rear -> next = ptr;
                rear = ptr;
                rear->next = NULL;
            }
        }
    }
    void hapus ()
    {
        struct node *ptr;
        if(front == NULL)
        {
            printf("\nUNDERFLOW\n");
            return;
        }
        else
        {
            ptr = front;
            front = front -> next;
            free(ptr);
        }
    }
    void display()
    {
        struct node *ptr;
        ptr = front;
        if(front == NULL)
        {
            printf("\nAntrian kosong\n");
        }
        else
        {   printf("\nMenampilkan antrian .....\n");
            while(ptr != NULL)
            {
                printf("\n%d\n",ptr -> data);
                ptr = ptr -> next;
            }
        }
    }
```

3. Circular Queue

```
#include <stdio.h>

# define ukuran 6
int queue[ukuran];  // deklarasi array
int front=-1;
int rear=-1;
// fungsi untuk menambahkan elemen pada circular queue
```

```
void enqueue(int element)
{
    if(front==-1 && rear==-1)    // kondisi untuk mengecek apakah
queue kosong
    {
        front=0;
        rear=0;
        queue[rear]=element;
    }
    else if((rear+1)%ukuran==front)    // kondisi untuk mengecek
apakah queue penuh
    {
        printf("Antrian telah penuh..");
    }
    else
    {
        rear=(rear+1)%ukuran;        // nilai pada rear bertambah
        queue[rear]=element;       // memberikan nilai pada queue
posisi rear
    }
}

// fungsi untuk menghapus elemen dari queue
void dequeue()
{
    if((front==-1) && (rear==-1))  // kondisi untuk mengecek apakah
queue kosong
    {
        printf("\nQueue belum penuh..");
    }
      else if(front==rear) {
            printf("\nElemen   yang   di-dequeue   adalah   %d   ",
queue[front]);
            front=-1;
            rear=-1;
      } else {
            printf("\nElemen   yang   di-dequeue   adalah   %d   ",
queue[front]);
            front=(front+1)%ukuran;
      }
}
// fungsi untuk menampilkan elemen queue

void display()
{
    int i=front;
    if(front==-1 && rear==-1){
        printf("\n Queue kosong!!..");
    }
    else {
        printf("\nElemen pada queue adalah : ");
        while(i<=rear) {
            printf("%d,", queue[i]);
            i=(i+1)%ukuran;
        }
    }
}
int main()
```

```
{
    int menu=1,x;   // deklarasi variabel

    while(menu<4 && menu!=0)   // looping menggunakan while
    {
      printf("\n****************Circular
Queue*******************\n");

printf("\n=================================================\n");
    printf("\n1: Tambah sebuah elemen");
    printf("\n2: Hapus sebuah elemen");
    printf("\n3: Tampilkan sebuah elemen");
    printf("\nMasukan pilihanmu! ");
    scanf("%d", &menu);

    switch(menu)
    {

        case 1:

        printf("Silakan masukan elemen yang ingin kamu tambah ");
        scanf("%d", &x);
        enqueue(x);
        break;
        case 2:
        dequeue();
        break;
        case 3:
        display();

    }
      }
    return 0;
}
```

4. Deque

```
#include <stdio.h>
#define ukuran 5
int deque[ukuran];
int f = -1, r = -1;
//  Fungsi untuk menambahkan nilai pada bagian depan
void insert_front(int x)
{
    if((f==0 && r==ukuran-1) || (f==r+1))
    {
        printf("Overflow");
    }
    else if((f==-1) && (r==-1))
    {
```

```
    f=r=0;
    deque[f]=x;
}
else if(f==0)
{
    f=ukuran-1;
    deque[f]=x;
}
else
{
    f=f-1;
    deque[f]=x;
    }
}

// Fungsi untuk menambahkan nilai pada bagian belakang
void insert_rear(int x)
{
    if((f==0 && r==ukuran-1) || (f==r+1))
    {
        printf("Overflow");
    }
    else if((f==-1) && (r==-1))
    {
        r=0;
        deque[r]=x;
    }
    else if(r==ukuran-1)
    {
        r=0;
        deque[r]=x;
    }
    else
    {
        r++;
        deque[r]=x;
    }

}

// Menampilkan semua nilai yang di deque
void display()
{
    int i=f;
    printf("\nElemen pada deque adalah: ");

    while(i!=r)
    {
       printf("%d ",deque[i]);
       i=(i+1)%ukuran;
    }
    printf("%d",deque[r]);
}

// getfront function retrieves the first value of the deque.
void getfront()
{
    if((f==-1) && (r==-1))
    {
        printf("Deque kosong");
    }
```

```
    else
    {
        printf("\nNilai dari elemen pada bagian depan adalah: %d ",
deque[f]);
    }

}

// fungsi getrear digunakan untuk mendapatkan nilai paling belakang
pada deque.
void getrear()
{
    if((f==-1) && (r==-1))
    {
        printf("Deque kosong");
    }
    else
    {
        printf("\nNilai pada elemen belakang adalah %d ",
deque[r]);
    }

}

// fungsi delete_front() untuk menghapus elemen dari depan

void delete_front()
{
    if((f==-1) && (r==-1))
    {
        printf("Deque kosong");
    }
    else if(f==r)
    {
        printf("\nElemen yang dihapus adalah %d ", deque[f]);
        f=-1;
        r=-1;

    }
     else if(f==(ukuran-1))
      {
          printf("\nElemen yang dihapus adalah %d ", deque[f]);
          f=0;
      }
      else
      {
           printf("\nElemen yang dihapus adalah %d ", deque[f]);
           f=f+1;
      }
}

//  fungsi  delete_rear()  digunakan  untuk  menghapus  elemen  dari
belakang
void delete_rear()
{
    if((f==-1) && (r==-1))
    {
        printf("Deque kosong");
    }
    else if(f==r)
```

```
    {
        printf("\nElemen yang dihapus adalah %d ", deque[r]);
        f=-1;
        r=-1;
    }
    else if(r==0)
    {
        printf("\nElemen yang dihapus adalah %d ", deque[r]);
        r=ukuran-1;
    }
    else
    {
        printf("\nElemen yang dihapus adalah %d ", deque[r]);
        r=r-1;
    }
}

int main()
{
    insert_front(20);
    insert_front(10);
    insert_rear(30);
    insert_rear(50);
    insert_rear(80);
    display();  // Memanggil fungsi display untuk mendapatkan nilai
yang di deque
    getfront();  // Mendapatkan nilai pada bagian front-end
    getrear();  // Mendapatkan nilai pada bagian rear-end
    delete_front();
    delete_rear();
    display(); // Memanggil fungsi display untuk mendapatkan nilai
setelah proses penghapusan
    return 0;
}
```

5. Priority Queue

```
//Implementasi priority queue menggunakan heap

#include <stdio.h>
int ukuran = 0;
void swap(int *a, int *b) {
  int temp = *b;
  *b = *a;
  *a = temp;
}

// Fungsi untuk mengatur ulang elemen heap pada tree
void heapify(int array[], int ukuran, int i) {
  if (ukuran == 1) {
    printf("Elemen tunggal pada heap");
  } else {
    // Menemukan yang terbesar di antara root, left child dan
right child
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < ukuran && array[l] > array[largest])
      largest = l;
    if (r < ukuran && array[r] > array[largest])
```

```
      largest = r;

    // Tukar (swap) dan lanjutkan proses untuk mengatur ulang
elemen heap jika root bukan yang terbesar
    if (largest != i) {
      swap(&array[i], &array[largest]);
      heapify(array, ukuran, largest);
    }
  }
}
// Fungsi untuk memasukan elemen pada tree
void insert(int array[], int newNum) {
  if (ukuran == 0) {
    array[0] = newNum;
    ukuran += 1;
  } else {
    array[ukuran] = newNum;
    ukuran += 1;
    for (int i = ukuran / 2 - 1; i >= 0; i--) {
      heapify(array, ukuran, i);
    }
  }
}

// Fungsi untuk menghapus elemen dari tree
void deleteRoot(int array[], int num) {
  int i;
  for (i = 0; i < ukuran; i++) {
    if (num == array[i])
      break;
  }

  swap(&array[i], &array[ukuran - 1]);
  ukuran -= 1;
  for (int i = ukuran / 2 - 1; i >= 0; i--) {
    heapify(array, ukuran, i);
  }
}

// Menampilkan array
void printArray(int array[], int ukuran) {
  for (int i = 0; i < ukuran; ++i)
    printf("%d ", array[i]);
  printf("\n");
}

// Kode driver untuk menjalankan program
int main() {
  int array[10];
  insert(array, 3);
  insert(array, 4);
  insert(array, 9);
  insert(array, 5);
  insert(array, 2);

  printf("Max-Heap array adalah : ");
  printArray(array, ukuran);
  deleteRoot(array, 4);
  printf("Setelah proses penghapusan elemen: ");
  printArray(array, ukuran);
}
```

LATIHAN

1. **Latihan 1**

```
#include<stdio.h>
#define ---- 5

void enQueue(int);
void deQueue();
void tampilkan();

int items[ukuran], ---- = -1, ---- = -1;
int main()
{
    deQueue();
    enQueue(7);
    enQueue(8);
    enQueue(3);
    enQueue(4);
    enQueue(9);
    enQueue(6);
    tampilkan();
    deQueue();
    tampilkan();
    return 0;
}

void ----(int value){
    if(belakang == ukuran-1)
        printf("\nQueue telah penuh!");
    else {
        if(depan == -1)
            depan = 0;
        belakang++;
        -----[belakang] = value;
        printf("\nNilai ditambahkan -> ----", value);
    }
}

void ----(){
    if(depan == -1)
        printf("\nQueue kosong!!");
    else{
        printf("\nNilai dihapus adalah : %d", ----[depan]);
        depan++;
        if(depan > belakang)
            depan = belakang = -1;
    }
}
void tampilkan(){
    if(belakang == -1)
        printf("\nQueue kosong!!!");
    else{
        int ----;
        printf("\nElemen pada Queue adalah:\n");
        for(i=depan; i<=belakang; i++)
            printf("%d\t",----[i]);
    }
}
```

2. **Latihan 2**

```
#include <stdio.h>
#include <stdlib.h>
#define BENAR 1
#define SALAH 0
#---- PENUH 10

struct node
{
    int data;
    struct node *next;
};
----- struct node node;

struct queue
{
    int count;
    node *front;
    node *rear;
};
typedef struct queue queue;

void inisialisasi(---- *q)
{
    q->count = 0;
    q->front = NULL;
    q->rear = NULL;
}

int isempty(---- *q)
{
    return (q->rear == NULL);
}

void enqueue(---- *q, int value)
{
    if (q->count < PENUH)
    {
        node *tmp;
        tmp = (node*)malloc(sizeof(node));
        tmp->data = value;
        tmp->next = NULL;
        if(!isempty(q))
        {
            q->rear->next = tmp;
            q->rear = tmp;
        }
        else
        {
            q->front = q->rear = tmp;
        }
        q->count++;
    }
    ----
    {
        printf("Antrian penuh\n");
    }
}
```

```
int dequeue(---- *q)
{
    node *tmp;
    int n = q->front->data;
    tmp = q->front;
    q->front = q->front->next;
    q->count--;
    free(tmp);
    return(n);
}

void display(---- *head)
{
    if(head == NULL)
    {
        printf("NULL\n");
    }
    else
    {
        printf("%d\n", head -> data);
        display(head->next);
    }
}

int main()
{
    ---- *q;
    q = (queue*)----(sizeof(queue));
    inisialisasi(q);
    enqueue(----,10);
    enqueue(----,20);
    enqueue(----,30);
    printf("Queue sebelum proses dequeue\n");
    display(q->front);
    dequeue(q);
    printf("Queue setelah proses dequeue\n");
    display(q->front);
    return 0;
}
```

TUGAS

1. Buatlah program tentang algoritma Algoritma Breadth First Search (BFS) dalam bahasa C. Lalu, jelaskan terkait algoritma tersebut dan bagaimana prinsip queue digunakan dalam algoritma tersebut!
2. Buatlah tugas anda dalam dokumen .PDF lengkap dengan source code dan penjelasan
3. Dokumen .PDF dan source code di upload ke github bersama hasil percobaan dan Latihan

DAFTAR PUSTAKA

1. Kernighan, Brian W, & Ritchie, Dennis M. 1988. The Ansi C Programming Language Second Edition, Prentice-Hall.
2. Cipta Ramadhani. 2015. Dasar Algoritma & Struktur Data. Yogyakarta: ANDI.