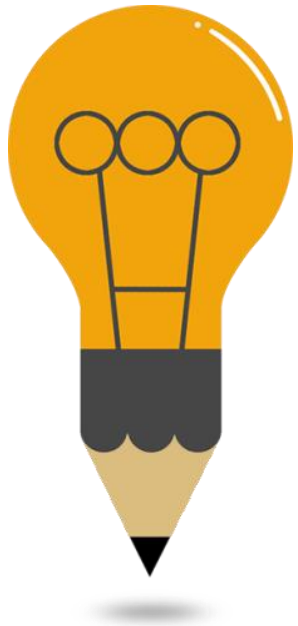
STRUKTUR DATA

Pertemuan 7

Ratih Ngestrini, Nori Wilantika

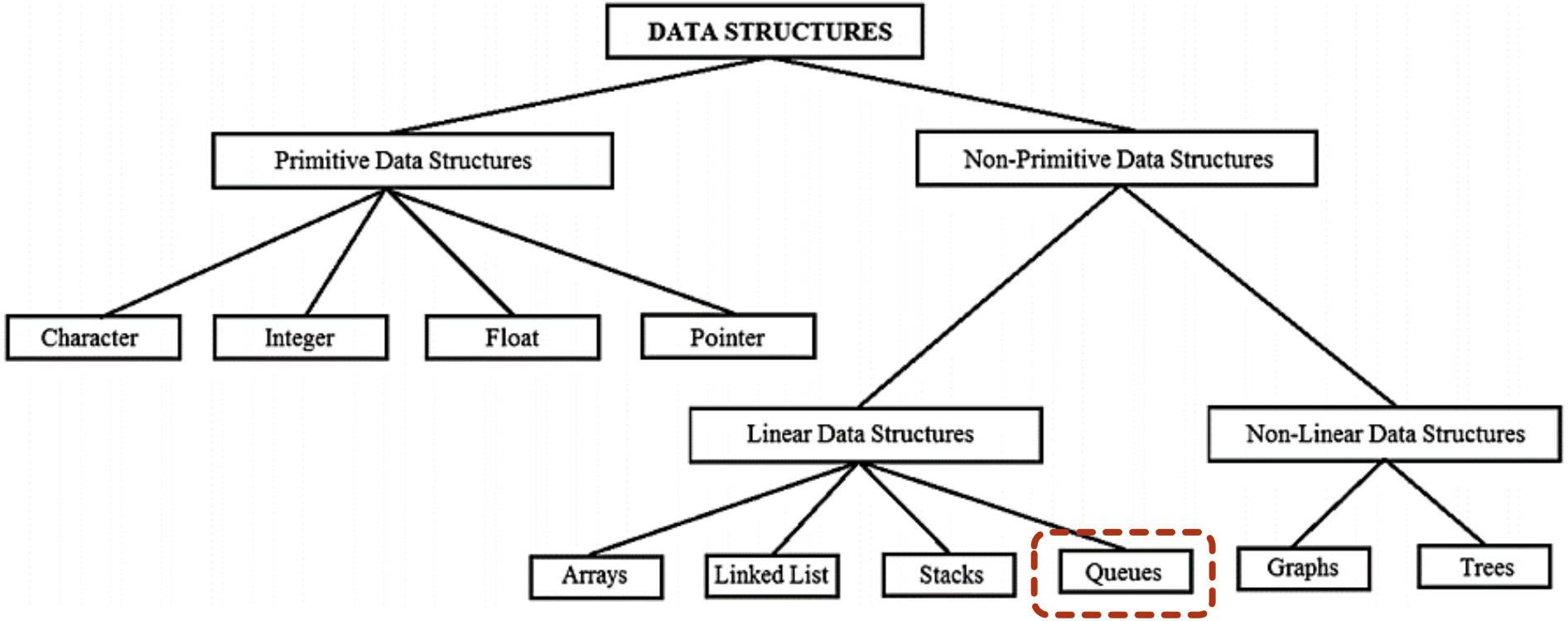
Agenda Pertemuan



1. Review Latihan Stack
2. Antrian (*queue*)
3. Antrian Berprioritas (*priority queue*)

ANTRIAN (QUEUE)

Jenis-Jenis Struktur Data

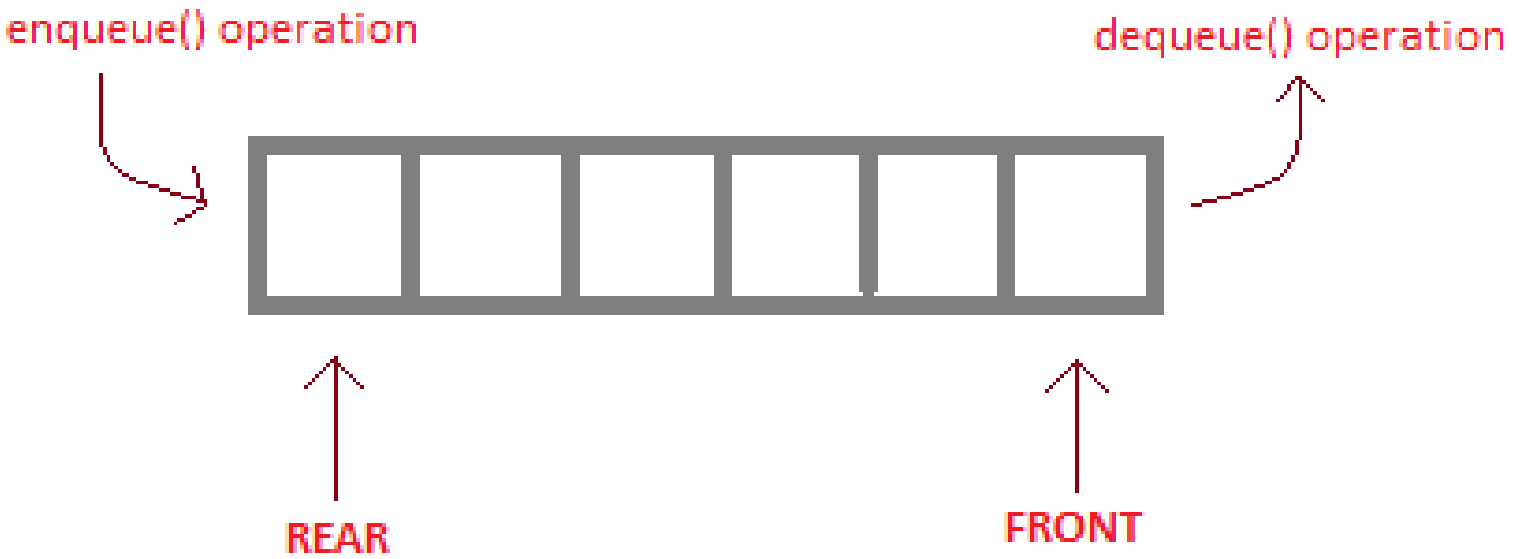


Queue (Antrian)

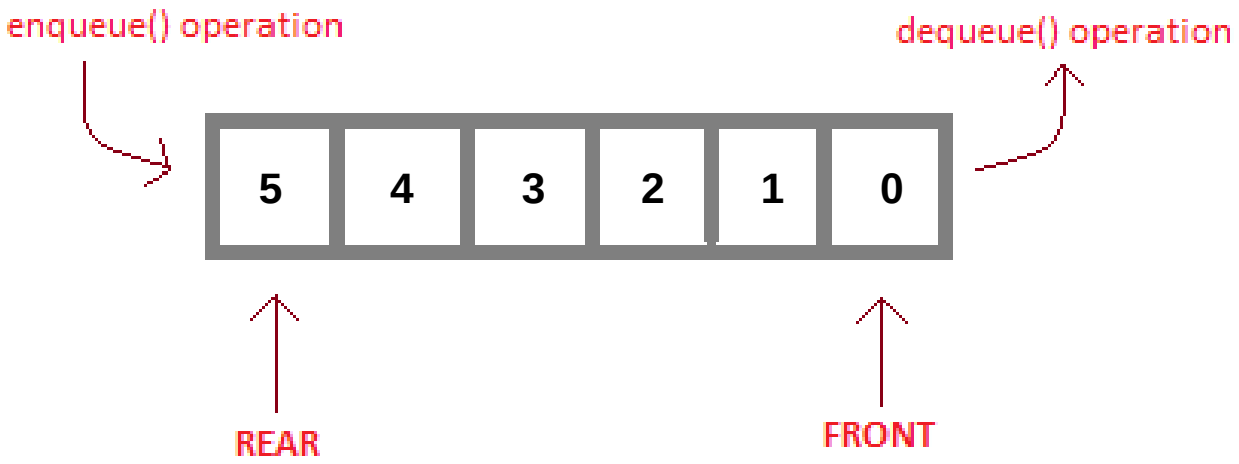
- Struktur data yang menyimpan data dengan konsep **FIFO (First In First Out)**
  - Atau bisa juga disebut **LILO (Last In Last Out)**
- Tidak seperti stack yang mempunyai satu *end*, queue mempunyai 2 *ends*/ujung yaitu **tail (rear)** dan **head (front)**
- Penambahan elemen hanya bisa dilakukan pada **tail (rear)** dan penghapusan (pengambilan elemen) dilakukan lewat **head (front)**

Operasi pada Queue

- **Enqueue** : menambah satu elemen baru ke dalam antrian (queue)
- **Dequeue** : menghapus elemen di dalam antrian (queue)



Implementasi Queue menggunakan Array



1. Deklarasikan stack **QUEUE** sebagai Array dengan ukuran **N** (kapasitas dari antrian)

```
#define N 50
int QUEUE[N], rear, front;
```

FRONT = -1 dan **REAR = -1** artinya queue kosong karena index array dimulai dari 0.

Implementasi Queue menggunakan Array

2. Buat fungsi untuk men-***display***, men-***enqueue*** **(insert)**, dan men-***dequeue*** **(remove)** elemen dalam antrian

```
void q_insert(int);
void q_remove();
void q_display();
```

Implementasi Queue menggunakan Array = ***enqueue***

Fungsi untuk menambahkan elemen baru ke antrian (dari **`rear`**)

```
void q_insert(int item)
{
    if(rear == N - 1){
        printf("Antrian penuh \n");
        return;
    }

    if(front == -1)
        front = 0;

    rear++;
    QUEUE[rear] = item;
}
```

Jika antrian kosong dan ditambahkan elemen baru, maka elemen tersebut berada pada **`front = rear = 0`**

Implementasi Queue menggunakan Array = ***dequeue***

Fungsi untuk menghapus elemen di antrian

```
void q_remove()
{
    if(rear == -1){
        printf("Antrian kosong \n");
        return;
    }

    if(front == rear)
        front = rear = -1;
    else{
        for(int i = 0; i < rear; i++) {
            QUEUE[i] = QUEUE[i + 1];
        }
        rear--;
        front = 0;
    }
}
```

Jika hanya ada satu elemen dalam antrian (**`front = rear = 0`**), maka kosongkan antrian (**`front = rear = -1`**)

Hapus elemen di front dengan memindahkan isi dari array dengan nilai setelahnya

Implementasi Queue menggunakan Array = ***display()***

Fungsi untuk menampilkan elemen di antrian

```
void q_display()
{
    if (rear == -1) {
        printf("Antrian kosong \n");
    }
    else{
        printf("Daftar antrian : \n");
        for(int i = front; i <= rear; i++) {
            printf("%d\n", QUEUE[i]);
        }
    }
}
```

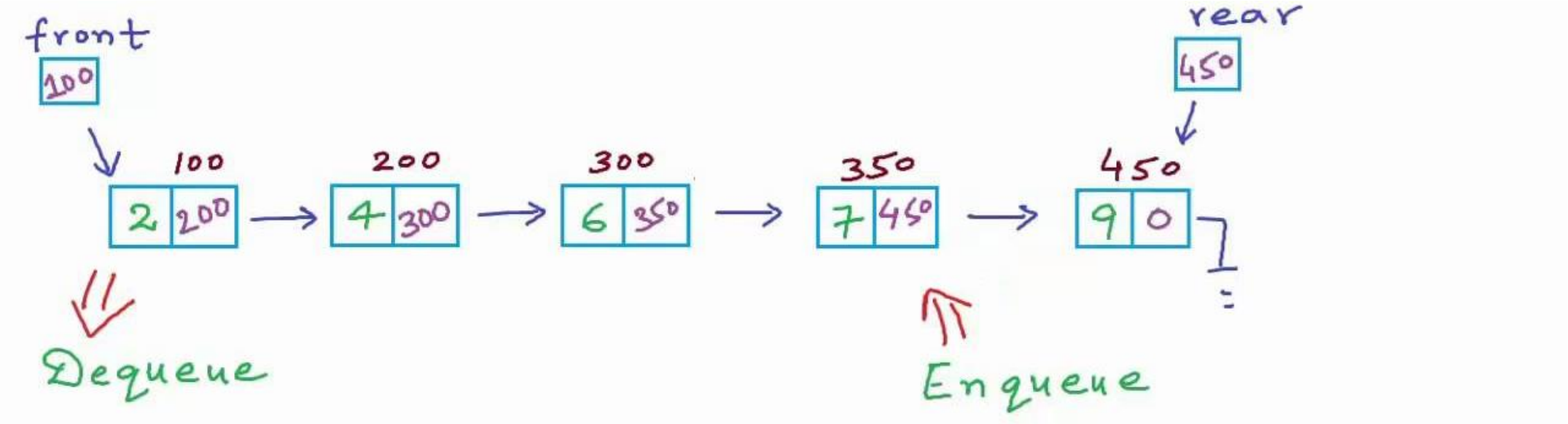
Implementasi Queue menggunakan Array

```c
#include <stdio.h>

#define N 50
int QUEUE[N], rear, front;

void q_insert(int);
void q_remove();
void q_display();
```

```c
int main()
{
    rear = - 1;
    front = - 1;

    q_insert(40);
    q_insert(50);
    q_insert(60);
    q_insert(70);
    q_insert(80);

    q_remove();
    q_remove();
    q_display();

    printf("\n\n%d %d", front, rear);

}
```

Implementasi Queue menggunakan Linked List

Pointer **`front`** menunjuk elemen pertama dalam antrian



Pointer **`rear`** menunjuk elemen terakhir dalam antrian

Implementasi Queue menggunakan Linked List

1. Deklarasikan elemen dari antrian dengan mendefinisikan **node structure** linked list

```
struct node
{
    int data;
    struct node* next;
};
typedef struct node* item;
```

2. Deklarasikan variabel untuk menyimpan jumlah node dalam antrian, **FRONT**, dan **REAR**

```
int count;
item front, rear;
```

Implementasi Queue menggunakan Linked List

3. Inisialisasi queue dengan jumlah node masih 0, front dan rear menunjuk ke NULL

```
void initialize()
{
    count = 0;
    front = NULL;
    rear = NULL;
}
```

Antrian kosong

```
bool isempty()
{
    return (rear == NULL);
}
```

Pointer **rear** atau **front** akan NULL jika antrian kosong

Implementasi Queue menggunakan Linked List = ***enqueue***

Fungsi untuk menambahkan elemen baru ke antrian – (setelah **`rear`**)

```
void q_insert(int value)
{
    item new_node;
    new_node = (item)malloc(sizeof(struct node));
    new_node->data = value;
    new_node->next = NULL;
    if(!isempty())
    {
        rear->next = new_node;
        rear = new_node;
    }
    else
    {
        front = rear = new_node;
    }
    count++;
}
```

1. Buat node/item baru **`new_node`**
2. Masukan data dari node **`new_node`**
3. Jika antrian tidak kosong, maka tunjuk pointer **`rear`** ke **`new_node`** dan buat **`new_node`** sebagai **`rear`** yang baru
4. Jika antrian kosong, maka tunjuk pointer **`front`** dan **`rear`** antrian ke node baru **`new_node`**
5. Jangan lupa update **`count`** di antrian

Implementasi Queue menggunakan Linked List = ***dequeue***

Fungsi untuk menghapus elemen di antrian – (di **`front`**)

```
void q_remove()
{
    item tmp;
    tmp = front;
    front = front->next;
    free(tmp);

    count--;
}
```

1. Buat temporary node **`tmp`** yang menunjuk pada elemen front
2. Jadikan node setelah **`front`** sebagai **`front`** yang baru
3. Hapus/bebaskan memory dari temporary node **`tmp`**
4. Jangan lupa update **`count`** di antrian

Implementasi Queue menggunakan Linked List

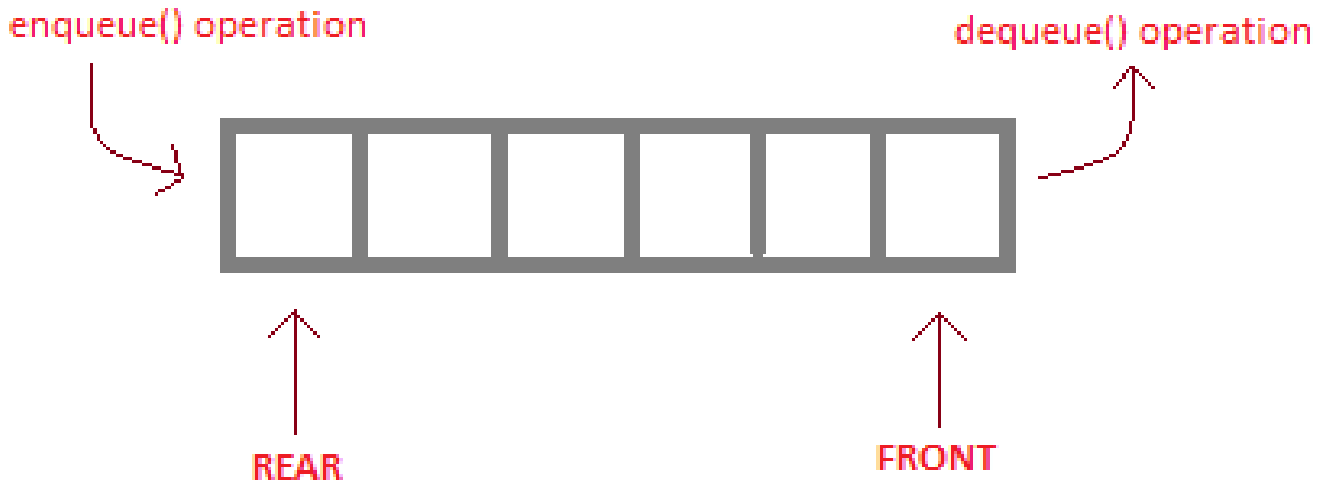
```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

struct node
{
    int data;
    struct node* next;
};
typedef struct node* item;


int count;
item front, rear;


void initialize();
bool isempty();
void q_insert(int value);
void q_remove();
void display(item head);
```

```c
int main()
{

    initialize();
    q_insert(10);
    q_insert(20);
    q_insert(30);
    q_insert(40);
    printf("Queue sebelum dequeue\n");
    display(front);
    q_remove();
    printf("Queue setelah dequeue\n");
    display(front);

    return 0;
}
```

ANTRIAN BERPRIORITAS (PRIORITY QUEUE)

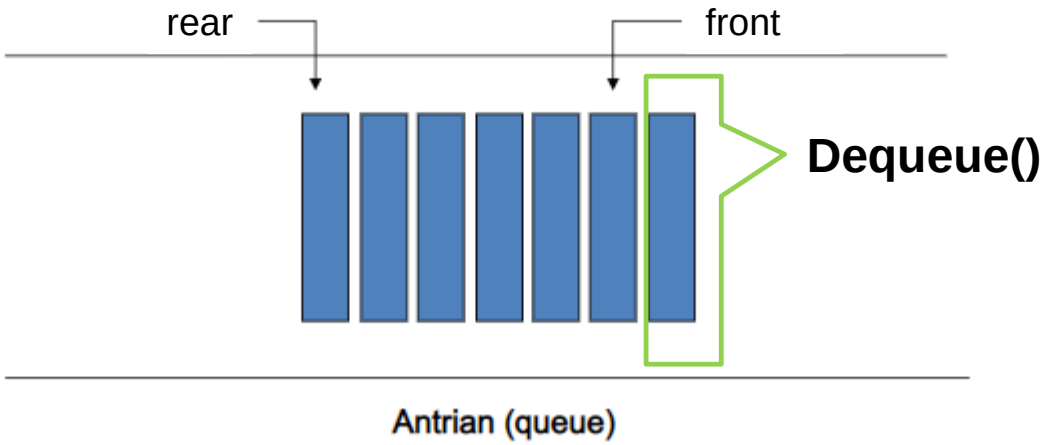
Antrian

- **Queue/antrian** adalah struktur data dimana penyisipan di satu ujung (rear), sedangkan penghapusan di ujung lain (front)
- Mengikuti konsep **FIFO (First In First Out)**: elemen yang pertama masuk akan menjadi elemen yang pertama keluar

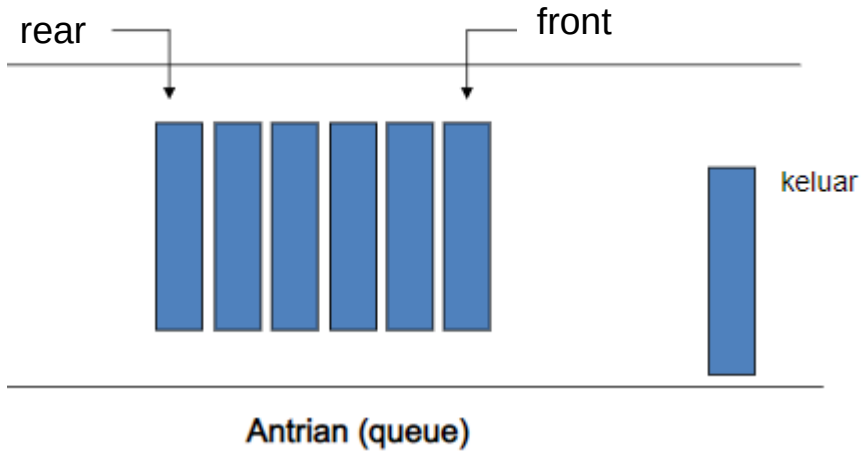


Aturan Mengeluarkan Elemen pada Queue

- Penunjuk **`front`** diubah ke elemen di belakang elemen paling depan



- Elemen paling depan dikeluarkan dari queue



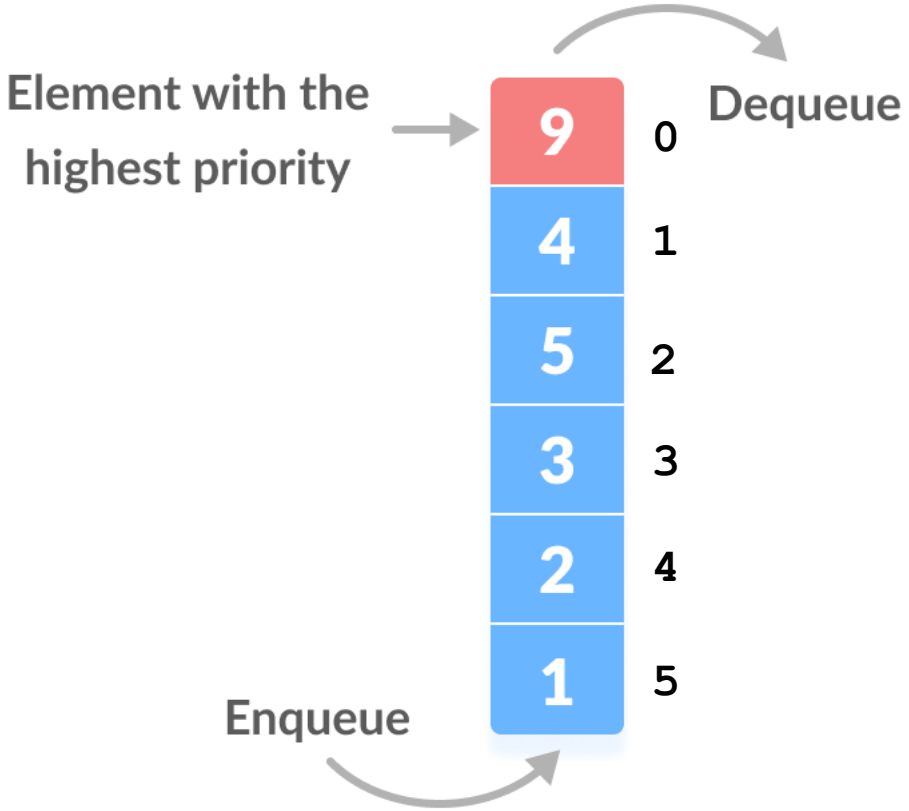
Antrian Berprioritas (*priority queue*)

- **Antrian berprioritas** adalah himpunan elemen, yang setiap elemennya telah diberikan sebuah prioritas
- Elemen pada antrian berprioritas harus menyimpan **nilai prioritas** pada setiap elemen
- Untuk memasukan elemennnya, tidak harus melalui ujung **`rear`**, tetapi disisipkan dan diurutkan berdasarkan prioritas elemen
- Mengeluarkan elemen dari queue berdasarkan prioritas dari elemen itu

Aturan

- Elemen yang prioritasnya lebih tinggi, diproses lebih dahulu dibandingkan dengan elemen yang prioritasnya lebih rendah (atau tergantung kasus/problem-nya)
- Dua elemen dengan prioritas yang sama, diproses sesuai dengan urutan mereka sewaktu dimasukkan ke dalam priority queue

Implementasi Menggunakan Array



Kita asumsikan yang datanya lebih besar mempunyai prioritas lebih tinggi (index array lebih kecil)

Implementasi Menggunakan Array

```
#define MAX 6

int intArray[MAX];
int itemCount = 0;

int peek(){
   return intArray[itemCount - 1];
}

bool isEmpty(){
   return itemCount == 0;
}

bool isFull(){
   return itemCount == MAX;
}
```

```
int size(){
   return itemCount;
}

void display(){
   for(int i = 0; i < itemCount; i++){
      printf("%d ", intArray[i]);
   }
}
```

Operasi `insert()` atau `enqueue()`

```
void insert(int data){
   int i = 0;
   if(!isFull()){
      if(itemCount == 0){
         intArray[itemCount] = data;
      }
      else{
         for(i = itemCount - 1; i >= 0; i-- ){
            if(data > intArray[i]){
               intArray[i+1] = intArray[i];
            }else{
               break;
            }
         }
         // insert the data
         intArray[i+1] = data;
      }

      itemCount++;
   }
}
```

Cari posisi untuk menambahkan elemen baru

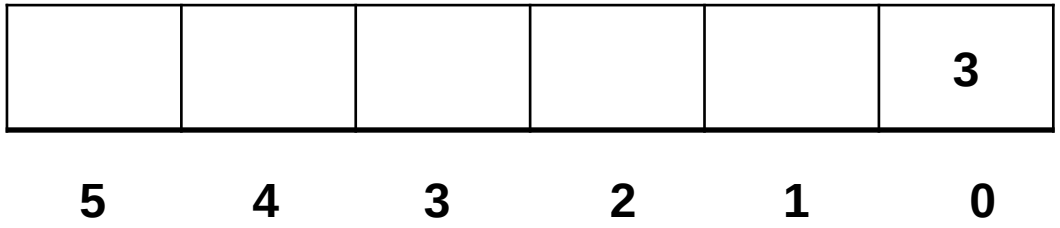
Kita asumsikan yang datanya lebih besar mempunyai prioritas lebih tinggi (index array lebih kecil)

Keluar dari `for()`

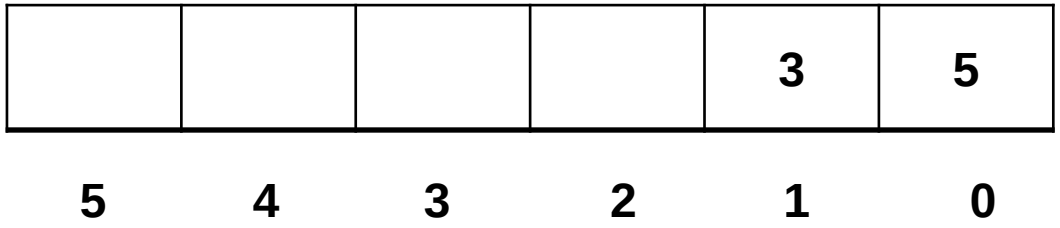
data = 5

`insert(5)`

Maka, data akan terurut dari besar ke kecil (nilai besar = prioritas tinggi untuk dequeue lebih dulu)



`itemCount = 1`



`itemCount = 2`

Operasi `remove()` atau `dequeque()`

Menghapus elemen di indeks ke-0 dengan menggeser elemen satu-satu

```
void removeData(){

    for(int i = 0; i < itemCount; i++){
        intArray[i] = intArray[i+1];
    }
    itemCount--;
}
```

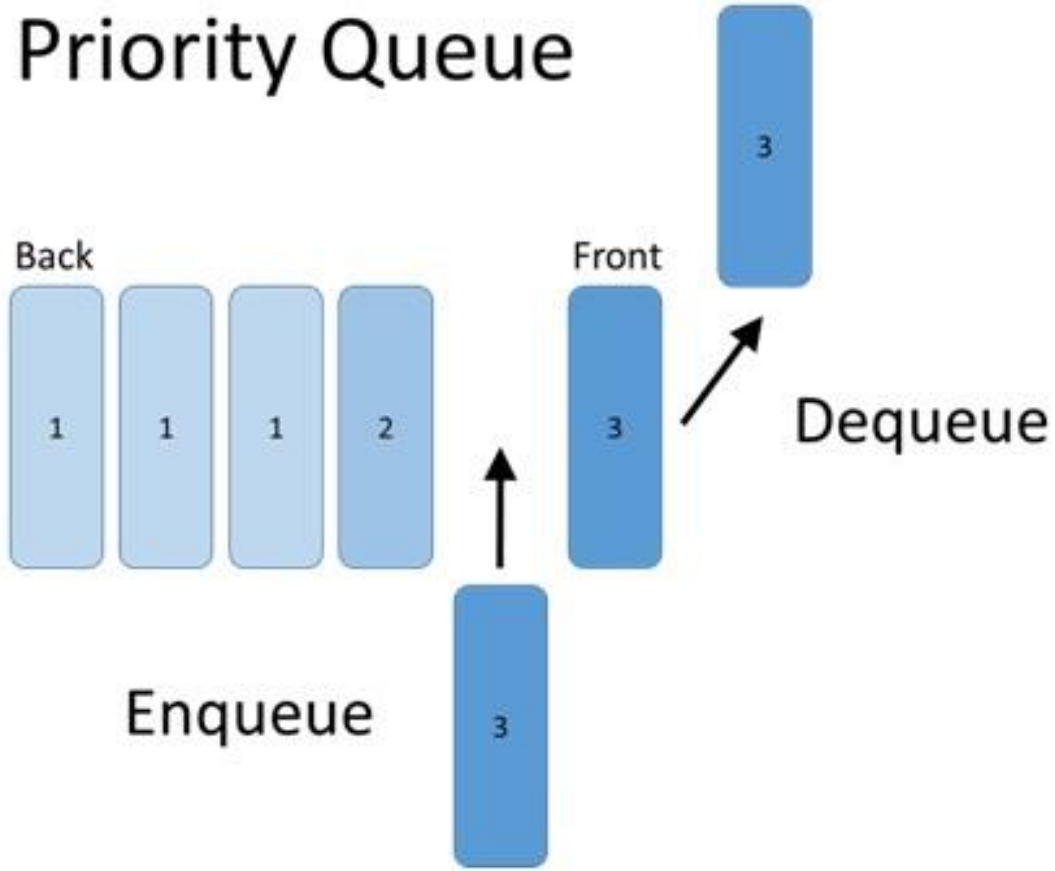
Main function()

```
int main() {
    insert(3);
    insert(5);
    insert(9);
    insert(1);
    insert(12);

    printf("Queue:  ");
    display();

    printf("\nElement at front: %d\n",peek());

    removeData();

    printf("\n---------Setelah Remove--------\n");
    printf("Queue:  ");

     display();
}
```

Output:

```
Queue:  12 9 5 3 1
Element at front: 12

---------Setelah Remove--------
Queue:  9 5 3 1
```

Implementasi Menggunakan Linked List



Implementasi Menggunakan Linked List

Nilai lebih rendah menunjukan prioritas lebih tinggi

```
struct node
{
    int data;
    int priority;
    struct node* next;
};
typedef struct node* item;
```

Fungsi untuk create node baru

```
item createNode(int d, int p)
{
    item new_node = (item)malloc(sizeof(struct node));
    new_node->data = d;
    new_node->priority = p;
    new_node->next = NULL;

    return new_node;
}
```

Operasi `insert()` atau `enqueue()`

```
void pq_insert(item* head, int d, int p)
{
    item new_node = createNode(d, p);

    if ((*head)->priority > p) {
        new_node->next = *head;
        (*head) = new_node;
    }
    else {
        item cursor = (*head);
        while (cursor->next != NULL && cursor->next->priority <= p) {
            cursor = cursor->next;
        }

        new_node->next = cursor->next;
        cursor->next = new_node;
    }
}
```

Buat node baru dengan data ***d*** dan priority ***p***

Jika node baru lebih diprioritaskan (priority-nya lebih kecil daripada head) maka jadikan sebagai head

Nilai p lebih rendah menunjukan prioritas lebih tinggi

Cari posisi untuk menambahkan elemen baru

Operasi `remove()` atau `dequeque()`

```
void pq_remove(item* head)
{
    item temp = *head;
    *head = (*head)->next;
    free(temp);
}
```

Menghapus elemen dengan prioritas paling tinggi (**hapus node head**)

Nilai p lebih rendah menunjukan prioritas lebih tinggi

Tampilkan Isi Antrian

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

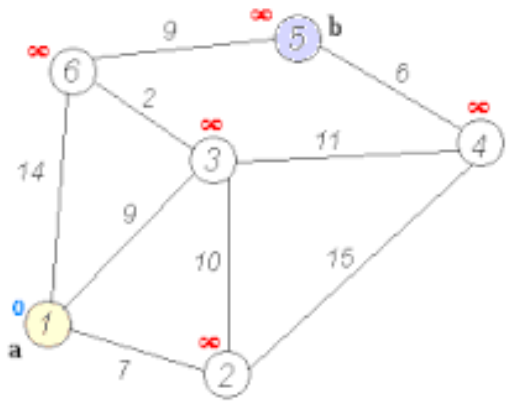
Main Function

```
int main()
{
    item pq = createNode(4, 1);
    pq_insert(&pq, 5, 2);
    pq_insert(&pq, 6, 3);
    pq_insert(&pq, 7, 0);

    display(pq);

    pq_remove(&pq);

    display(pq);

    return 0;
}
```

```
Output:

7
4
5
6
-
4
5
6
-
```

Contoh

- Time-sharing system dimana program yang mempunyai prioritas tinggi akan dikerjakan lebih dahulu dan program-program yang berprioritas sama akan membentuk antrian biasa
- Dijkstra's Shortest Path Algorithmm (https://id.wikipedia.org/wiki/Algoritme_Dijkstra)



- Prim's algorithm (https://id.wikipedia.org/wiki/Algoritme_Prim)

Aplikasi Queue dalam Dunia Nyata

- Digunakan Operating systems untuk job scheduling, CPU scheduling, Disk Scheduling
- Antrian pada ticket counter, customer service system, phone answering system, dll

TERIMA KASIH