

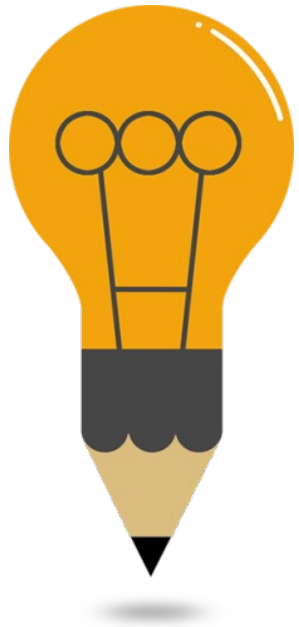
STRUKTUR DATA

Pertemuan 6

(Ratih Ngestrini)



Agenda Pertemuan



1

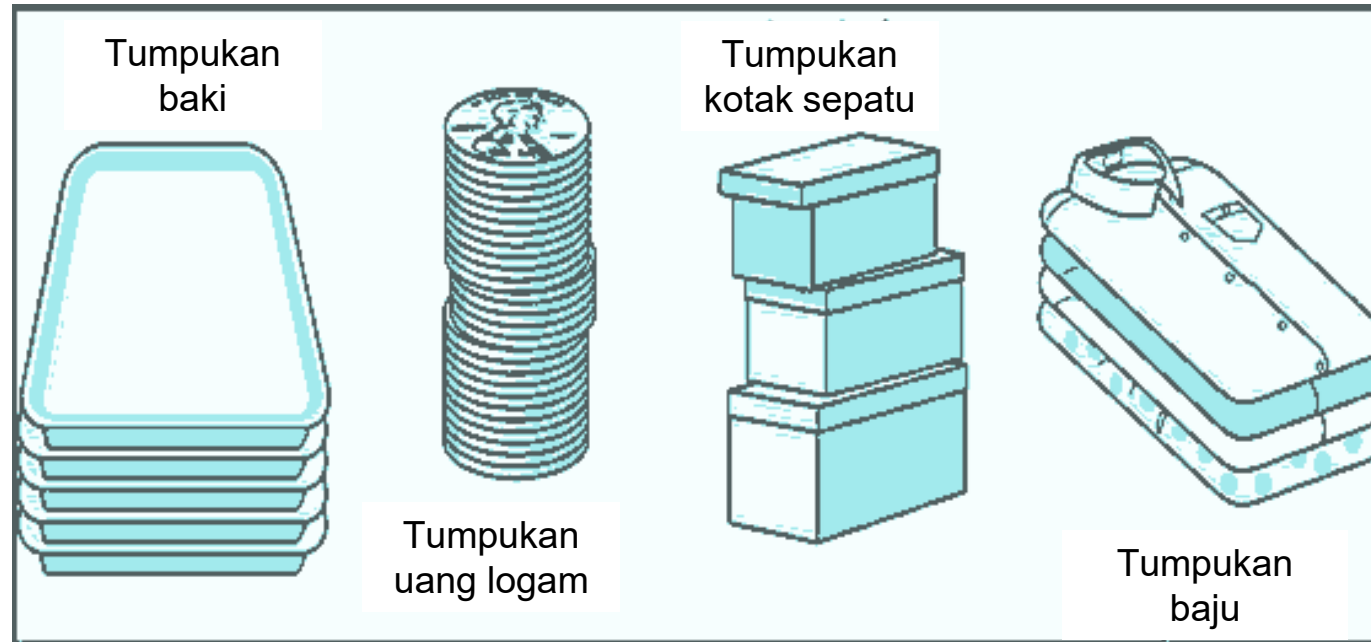
Tumpukan (*stack*)

2

Antrian (*queue*)

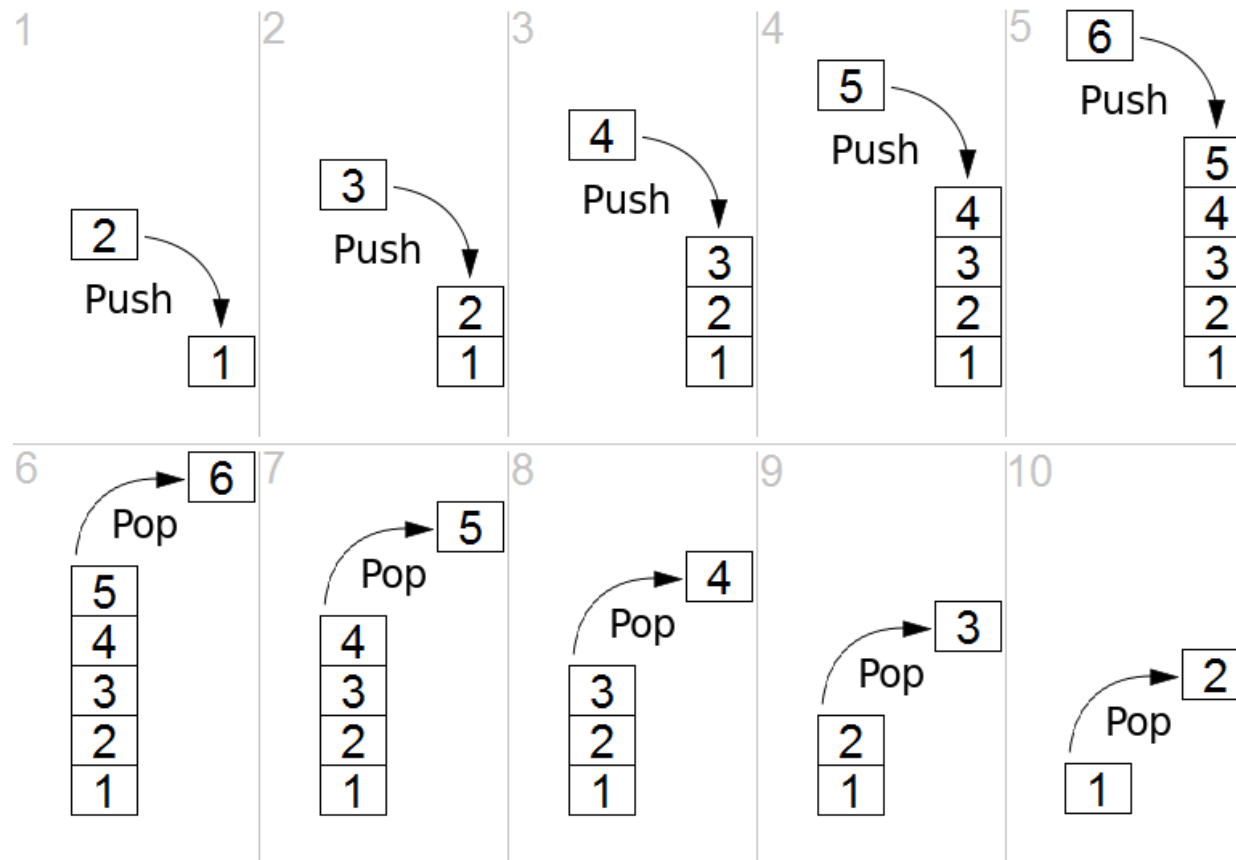
Stack (Tumpukan)

- Salah satu konsep penggunaan array atau linked list
- Struktur data untuk menyimpan data dengan *order* **LIFO (Last In First Out)**. Maksudnya, setiap data yang terakhir masuk, itu yang akan di panggil lebih dulu atau keluar lebih dulu
 - Atau bisa juga disebut **FILO (First In Last Out)**



Operasi pada Stack

- **Stack / Push:** insert elemen ke dalam stack
- **Unstack / Pop:** hapus elemen yang terakhir ditambahkan ke dalam stack



- Stack hanya mempunyai satu *end/pointer* yaitu **TOP** (elemen teratas dalam stack tersebut)
- Item dapat di-*push* atau di-*pop* menggunakan **TOP**
- **TOP = -1** stack kosong

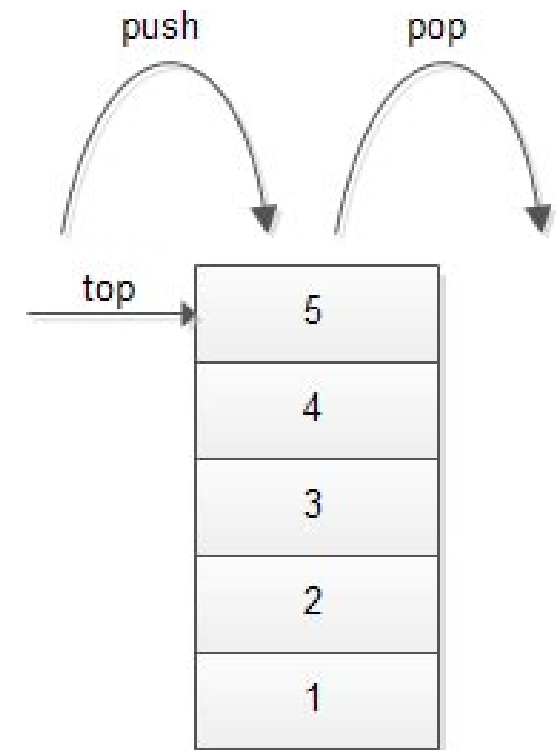
Implementasi Stack menggunakan Array

1. Deklarasikan stack **STACK** sebagai Array dengan ukuran **N** (kapasitas dari stack) dan **TOP** sebagai indeks array dari elemen paling atas stack tersebut

```
#define N 10  
int STACK[N], TOP;
```

2. Buat fungsi untuk men-**display**, mem-**push**, dan mem-**pop** elemen stack

```
void display(int []);  
void push(int [],int);  
void pop(int []);
```



Implementasi Stack menggunakan Array - `display()`

Fungsi untuk menampilkan isi dari stack

```
void display(int stack[])
{
    if(TOP >= 0){
        printf("Isi STACK : \n");
        for(int i = TOP; i >= 0; i--)
        {
            printf("\n%d", stack[i]);
        }
    }
    else{
        printf("STACK kosong .\n");
    }

    printf("\n\n");
}
```

Jika nilai **TOP** ≥ 0 artinya elemen teratas dari array stack ada di index 0, 1, 2, dst. Berarti stack tersebut ada isinya.

TOP = -1 artinya stack kosong karena index array dimulai dari 0.

Implementasi Stack menggunakan Array - **push()**

Fungsi untuk menambahkan elemen ke dalam stack

```
void push(int stack[], int item)
{
    if (TOP == N-1) {
        printf("\nSTACK penuh, tidak dapat ditambahkan item baru\n");
    }
    else {
        TOP++;
        stack[TOP] = item;
    }
}
```

Jika **TOP** (elemen teratas) berada di array index **N-1** berarti stack tersebut sudah penuh

Jika belum penuh, maka:

- **TOP = TOP + 1**
- isi array stack pada **TOP** dengan item

Implementasi Stack menggunakan Array = `pop()`

Fungsi untuk menghapus elemen dari stack : **LIFO (Last In First Out)** – yang dihapus adalah elemen di indeks **TOP**

```
void pop(int stack[])
{
    if(TOP == -1){
        printf("STACK sudah kosong.\n");
    }
    else{
        int deletedItem = stack[TOP];
        TOP--;
        printf("%d telah terhapus\n", deletedItem);
    }
}
```

Jika stack tidak kosong,
maka **TOP = TOP-1**

Apakah elemen yang dihapus yaitu `deletedItem` masih ada di array?

Jawab: masih, yang kita ubah-ubah hanya **TOP**, ketika kita **display()** tetap akan terbaca sampai **TOP**, ketika **push()** pun elemen baru akan menempa elemen yang tadinya sudah di **pop()**

Implementasi Stack menggunakan Array – Misal kita buat program yang menampilkan menu sehingga user bisa memilih operasi yang akan dilakukan

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

```
#define N 10
int STACK[N], TOP;
```

```
void display(int []);
void push(int [],int);
void pop(int []);
```

```
int main()
{
    TOP = -1;
    int choice = 0;
    do
    {
        printf("Masukan Pilihan 1: Display, 2: Tambah (PUSH), 3: Hapus (POP), 4: Exit :");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                display(STACK);
                break;
            case 2:
                printf("Masukan Item untuk Ditambahkan :");
                int ITEM = 0;
                scanf("%d",&ITEM);
                push(STACK,ITEM);
                break;
            case 3:
                pop(STACK);
                break;
            case 4:
                printf("\nKELUAR ");
                break;
            default:
                printf("\nPilihan Tidak Valid.");
        }
    }
    while(choice != 4);
    return 0;
}
```

Masukan Pilihan 1: Display, 2: Tambah (PUSH), 3: Hapus (POP), 4: Exit :1
STACK kosong .

Masukan Pilihan 1: Display, 2: Tambah (PUSH), 3: Hapus (POP), 4: Exit :2

Masukan Item untuk Ditambahkan :12

Masukan Pilihan 1: Display, 2: Tambah (PUSH), 3: Hapus (POP), 4: Exit :2

Masukan Item untuk Ditambahkan :13

Masukan Pilihan 1: Display, 2: Tambah (PUSH), 3: Hapus (POP), 4: Exit :2

Masukan Item untuk Ditambahkan :56

Masukan Pilihan 1: Display, 2: Tambah (PUSH), 3: Hapus (POP), 4: Exit :1

Isi STACK :

56

13

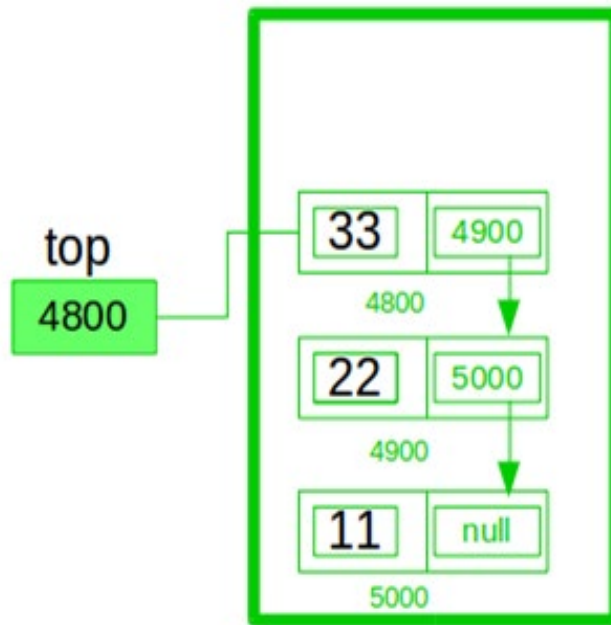
12

Masukan Pilihan 1: Display, 2: Tambah (PUSH), 3: Hapus (POP), 4: Exit :3

56 telah terhapus

Masukan Pilihan 1: Display, 2: Tambah (PUSH), 3: Hapus (POP), 4: Exit :

Implementasi Stack menggunakan Linked List



Stack menggunakan single linked list yang memiliki 3 elemen dan **TOP** (elemen teratas) mempunyai alamat **4800**

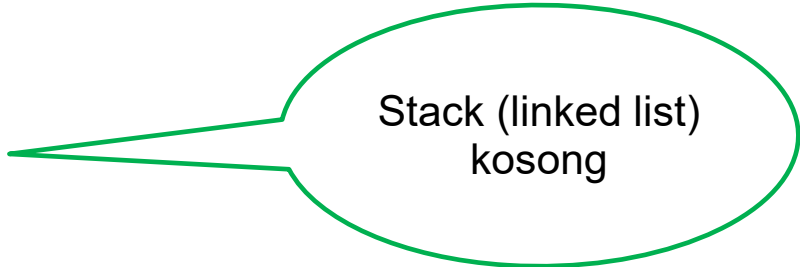
Implementasi Stack menggunakan Linked List

1. Deklarasikan elemen dari stack dengan mendefinisikan **node structure** linked list dan **TOP** dari stack

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

2. Inisialisasi stack dengan membuat pointer head dari stack tersebut menunjuk ke NULL

```
void initialize()
{
    top = NULL;
}
```



Stack (linked list)
kosong

Implementasi Stack menggunakan Linked List = `push()`

Fungsi untuk menambahkan elemen baru ke stack

```
void push(int value)
{
    item new_node;
    new_node = (item)malloc(sizeof(struct node));
    new_node->data = value;
    new_node->next = top;
    top = new_node;
}
```

1. Buat node baru `new_node`
2. Masukkan data dari node `new_node`
3. Tunjuk pointer `next` dari node `new_node` ke `TOP` sebelumnya
4. Jadikan node `new_node` sebagai `TOP` yang baru

Implementasi Stack menggunakan Linked List = `pop()`

Fungsi untuk menghapus elemen dari stack : **LIFO (Last In First Out)**

```
void pop()  
{  
    item tmp;  
    tmp = top;  
    top = top->next;  
    free(tmp);  
}
```

1. Buat temporary node `tmp` yang menunjuk ke `TOP`
2. Jadikan node setelah `TOP` sebagai `TOP` yang baru
3. Hapus/bebaskan memory dari temporary node `tmp`

Implementasi Stack menggunakan Linked List = `display()`

Fungsi untuk menampilkan isi dari stack

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

Fungsi untuk menampilkan isi dari node TOP

```
int DisplayTop()
{
    return top->data;
}
```

Implementasi Stack menggunakan Linked List

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

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

void initialize();
void push(int);
void pop();
void display(item head);
int DisplayTop();

int main()
{
    initialize();
    push(10);
    push(20);
    push(30);
    push(40);
    printf("Top dari stack adalah %d\n", DisplayTop());
    pop();
    printf("Top dari stack setelah pop adalah %d\n", DisplayTop());
    display(top);
    return 0;
}
```


Aplikasi Stack dalam Dunia Nyata

- Fitur undo-redo pada editor seperti text editor, photoshop, dll
- Fitur backward dan forward pada web browser
- Backtracking pada game

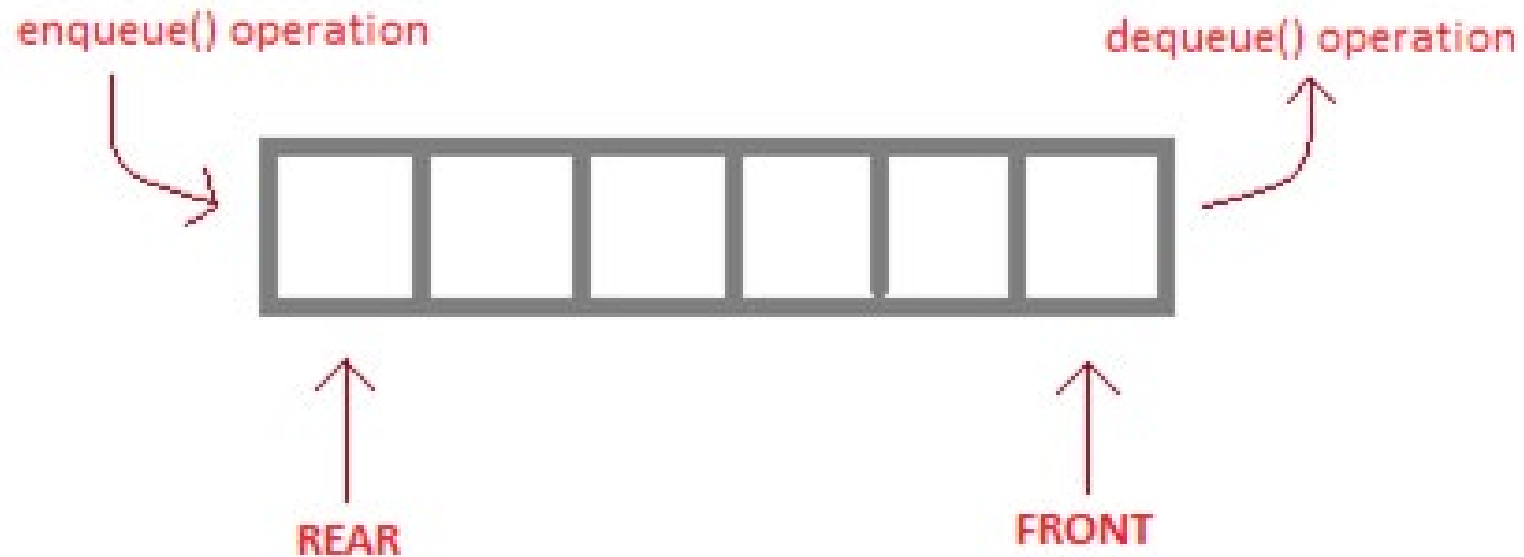
Queue (Antrian)

- Struktur data yang menyimpan data dengan konsep **FIFO (First In First Out)**
 - Atau bisa juga disebut **LIFO (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)**

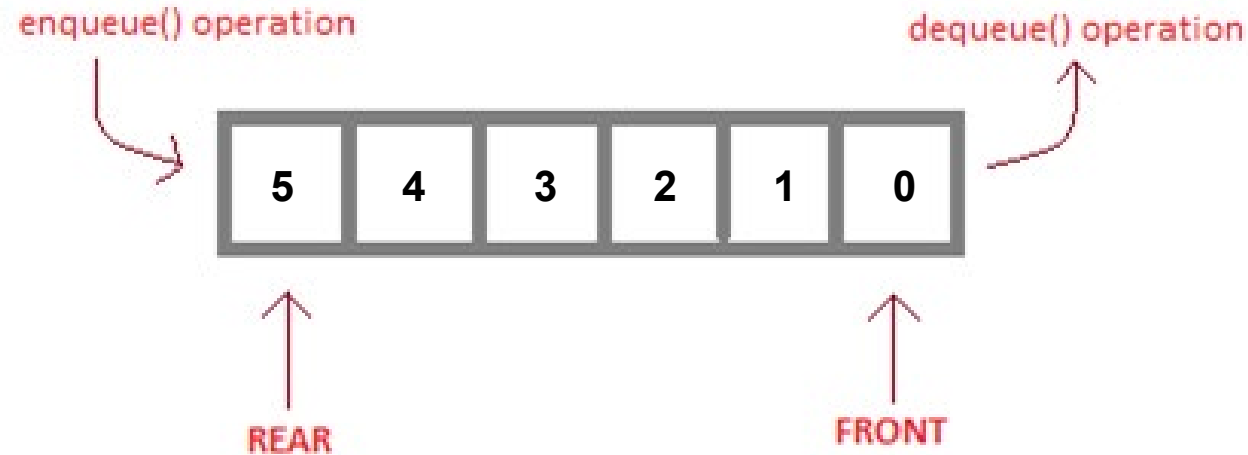


Operasi pada Queue

- **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

```
#include <stdio.h>

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

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

```
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);

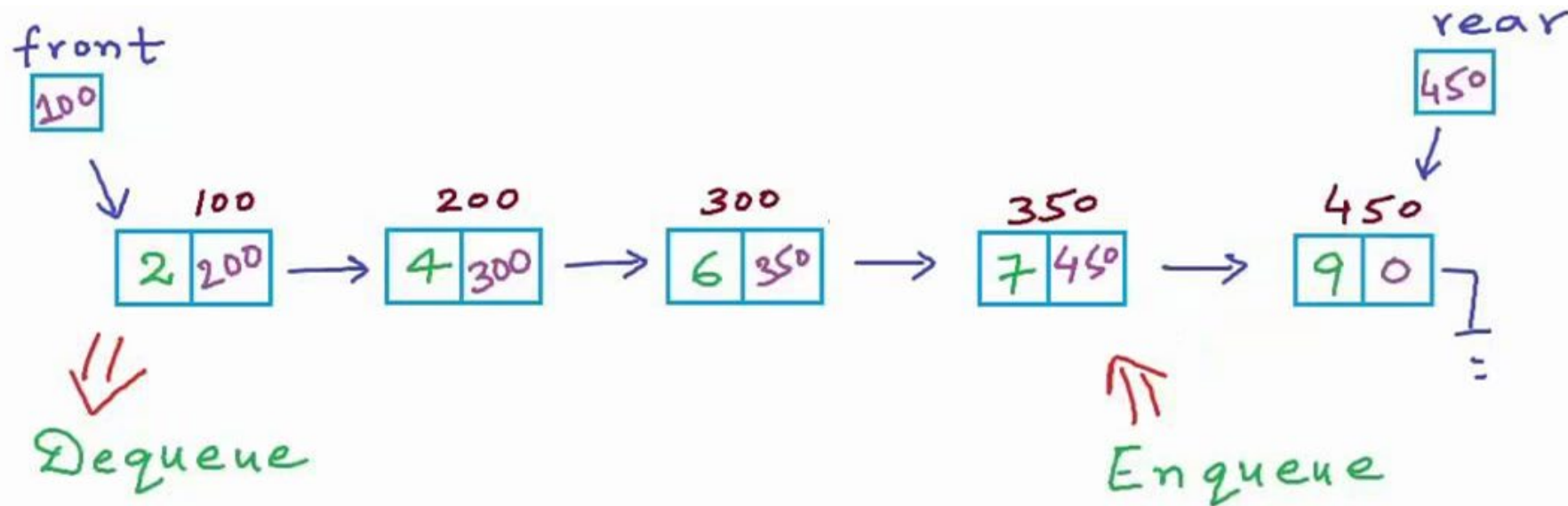
}
```



Antrian kosong

Implementasi Queue menggunakan Linked List

Pointer **front** menunjuk elemen pertama dalam antrian



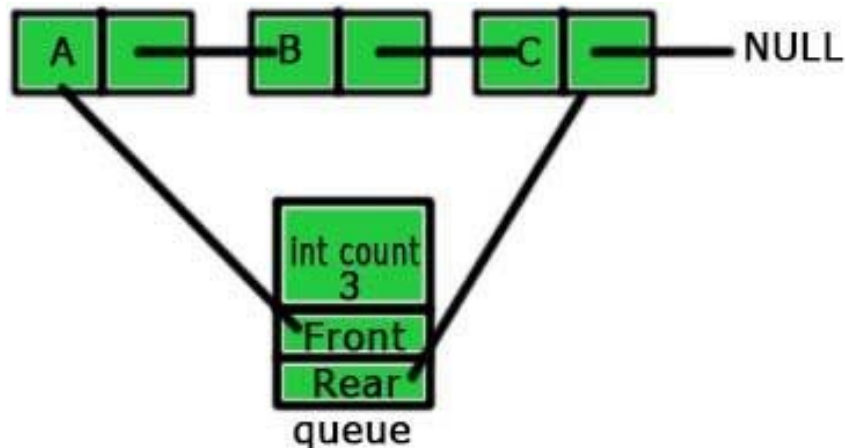
Pointer **rear** menunjuk elemen terakhir dalam antrian

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. Buat structure queue untuk menyimpan jumlah node dalam linked list, node front dan rear



```
struct queue
{
    int count;
    item front;
    item rear;
};
typedef struct queue* antrian;
```

Implementasi Queue menggunakan Linked List

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

```
void initialize(antrian q)
{
    q->count = 0;
    q->front = NULL;
    q->rear = NULL;
}
```

Antrian kosong

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

1. Buat node/item baru **new_node**
2. Masukkan 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(antrian q)
{
    item tmp;
    tmp = q->front;
    q->front = q->front->next;
    q->count--;
    free(tmp);
}
```

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

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

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

struct queue
{
    int count;
    item front;
    item rear;
};
typedef struct queue* antrian;

void initialize(antrian q);
bool isempty(antrian q);
void q_insert(antrian q, int value);
void q_remove(antrian q);
void display(item head);

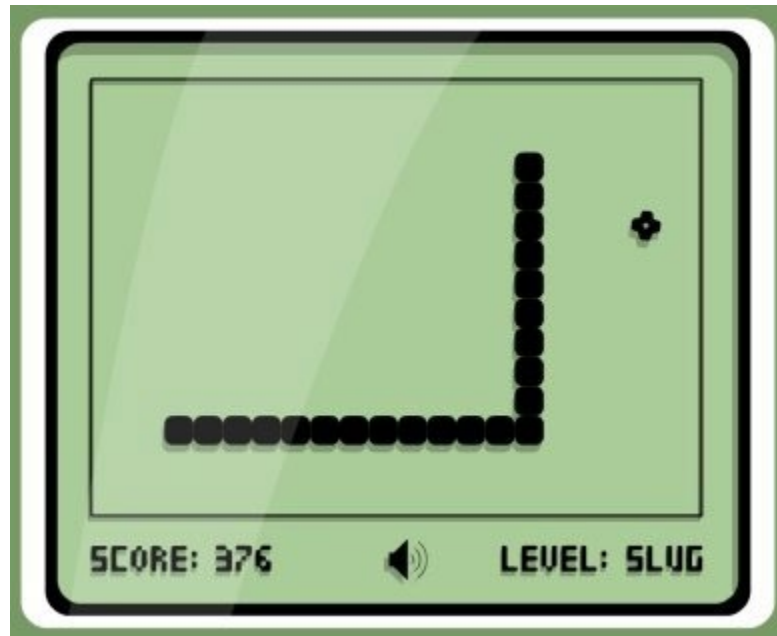
int main()
{
    antrian q;
    q = (antrian)malloc(sizeof(struct node));
    initialize(q);

    q_insert(q, 10);
    q_insert(q, 20);
    q_insert(q, 30);
    q_insert(q, 40);
    printf("Queue sebelum dequeue\n");
    display(q->front);
    q_remove(q);
    printf("Queue setelah dequeue\n");
    display(q->front);

    return 0;
}
```

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
- Simulasi
- Games



Latihan

1. Stack

Buat sebuah program menggunakan stack untuk menentukan apakah input string yang berisi pasangan tanda kurung sudah benar/seimbang.

Contoh:

Input : `{(())}` Output : TRUE / “Benar” / 1

Input : `[()]` Output : FALSE / “Salah” / 0

Input : `[()]{}{[()()]()}` Output : TRUE / “Benar” / 1

2. Queue

Buat sebuah program menggunakan queue untuk men-generate 1 sampai N binary numbers.

Contoh:

N = 10

Angka Binary: 1 10 11 100 101 110 111 1000 1001 1010 (lihat pola perubahan angka-nya)



TERIMA KASIH