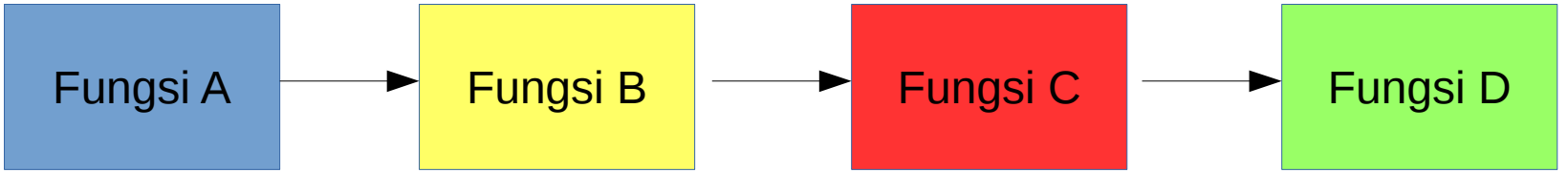
Stack dan Queue

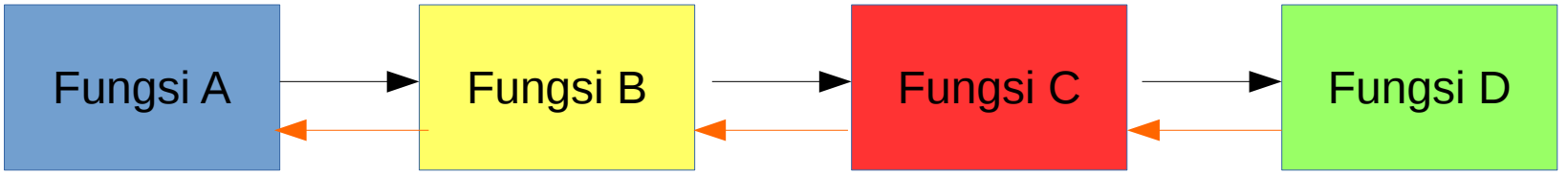
Pengantar

- Pemanggilan fungsi dalam program




```
void Func_A( ) { Func_B(); }
void Func_B( ) { Func_C(); }
void Func_C( ) { Func_D(); }
void Func_D( ) {  }
```

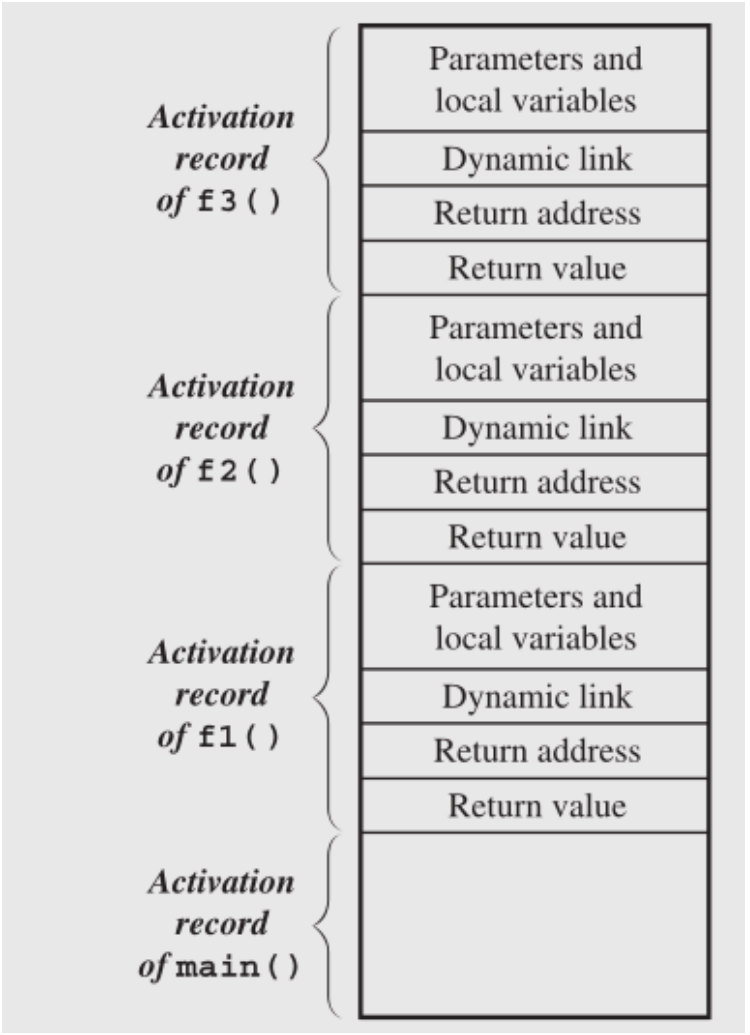
- Ketika eksekusi selesai, kontrol kembali ke fungsi awal



Apa itu Stack

- Komputer bekerja mengeksekusi fungsi-fungsi / prosedur
- Melacak proses eksekusi dan kembali ke main( )
- Sistem komputer harus melacak kembali, dari mana fungsi/prosedur dipanggil !
- Ini yang disebut dengan istilah Return Address
- Tiap kali pemanggilan fungsi, state dan informasi yang diperlukan direkam (variabel lokal, parameter, dll)

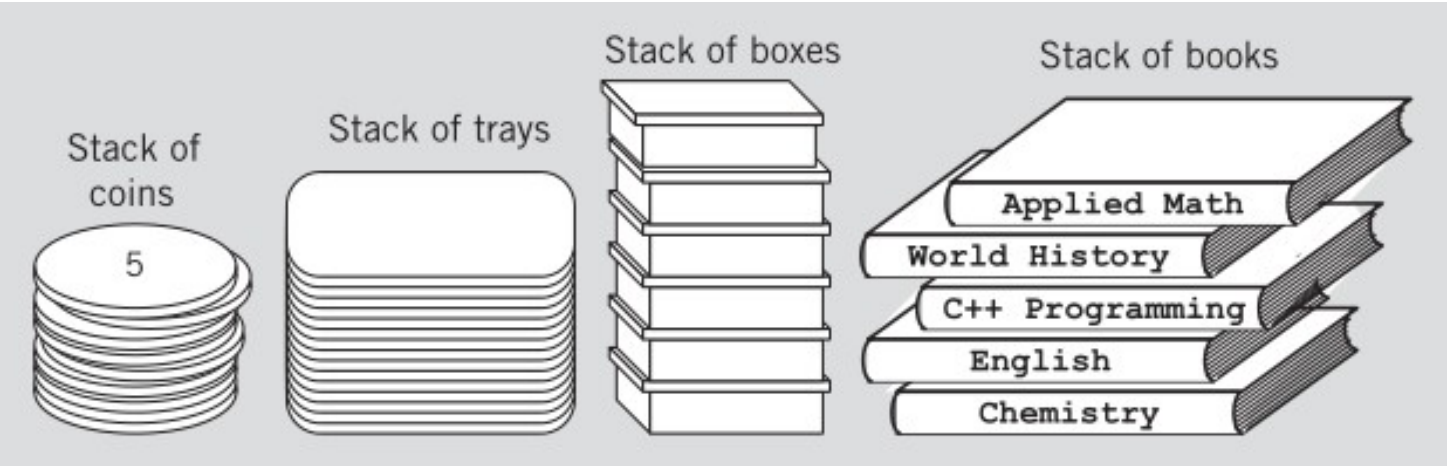
Runtime Stack



- Mekanisme pelacakan function call dengan mencatat dalam bentuk **record**
- main( ) → f1( ) → f2( ) → f3( )
- Menumpuk record dalam bentuk tumpukan
- Program Komputer bekerja dengan prinsip ini

Stack (Tumpukan)

- Koleksi items dimana operasi penambahan dan pengurangan hanya bisa dilakukan pada bagian atas (top)



- Bagian bawah yang pertama ditaruh, atas paling akhir masuk atau disebut LIFO (Last In First Out)

Stack sebagai LIFO

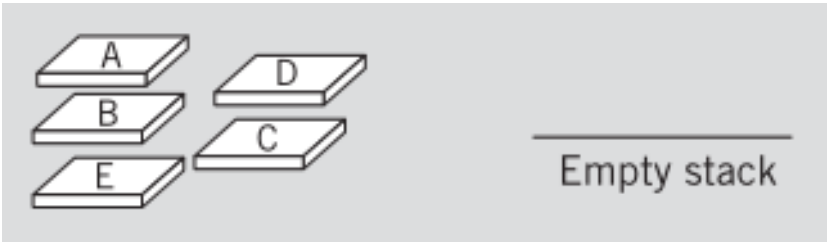
- LIFO Last In First Out

Penambahan dan pengurangan hanya Bisa dilakukan dari puncak (merah)

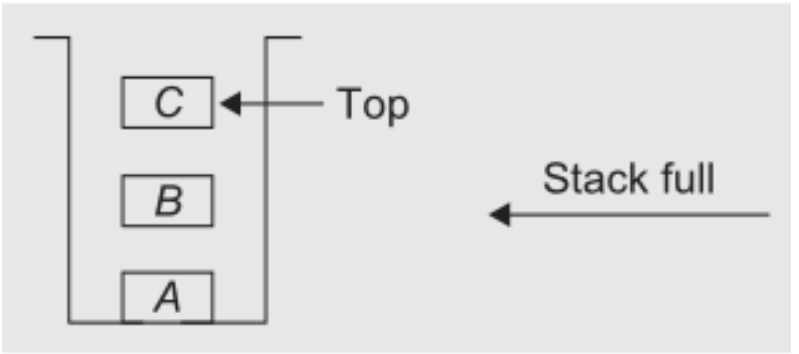
Elemen yang paling awal ditambah akan Ada dibagian bawah

Operasi pada Stack

- Kondisi awal (Stack kosong)

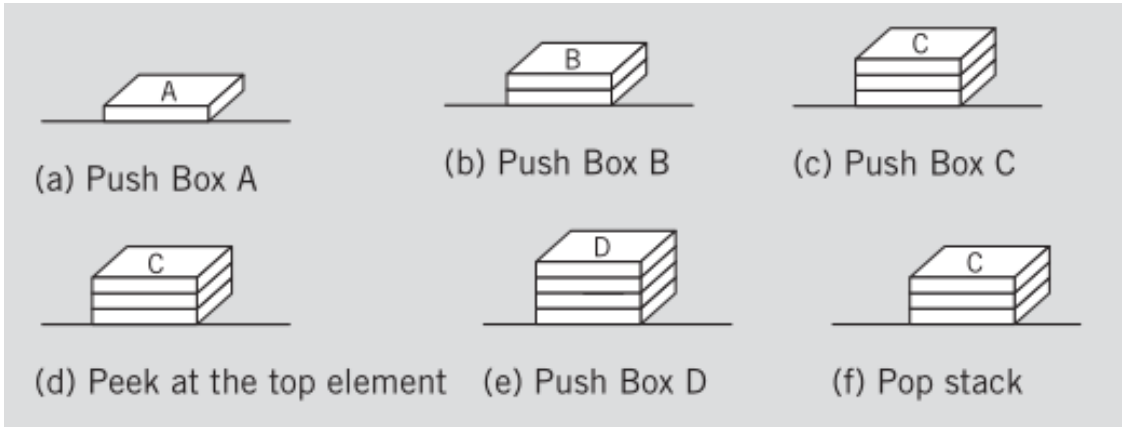


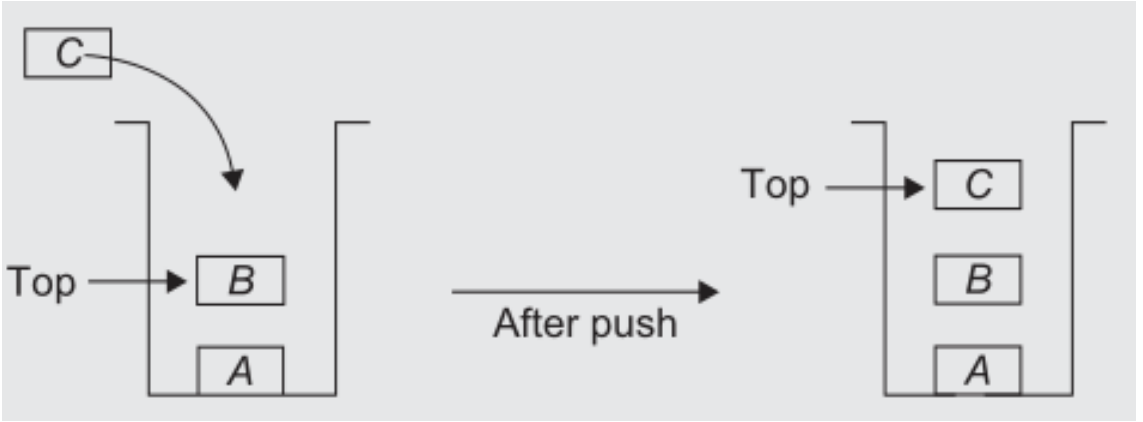
- Kondisi full (penuh) sampai pada kapasitasnya



Operasi (2)

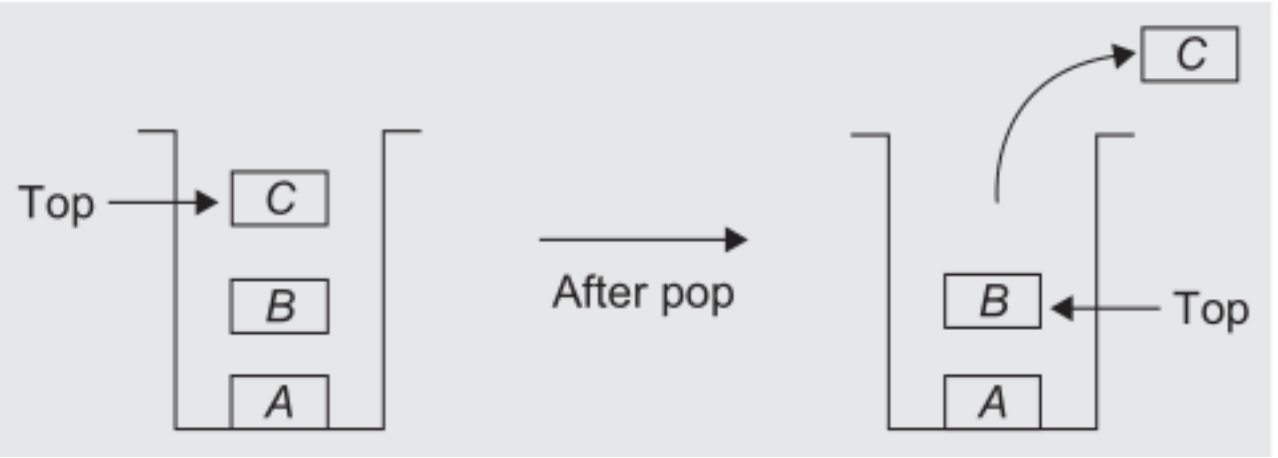
- Push (menaruh) item / elemen ke tumpukan



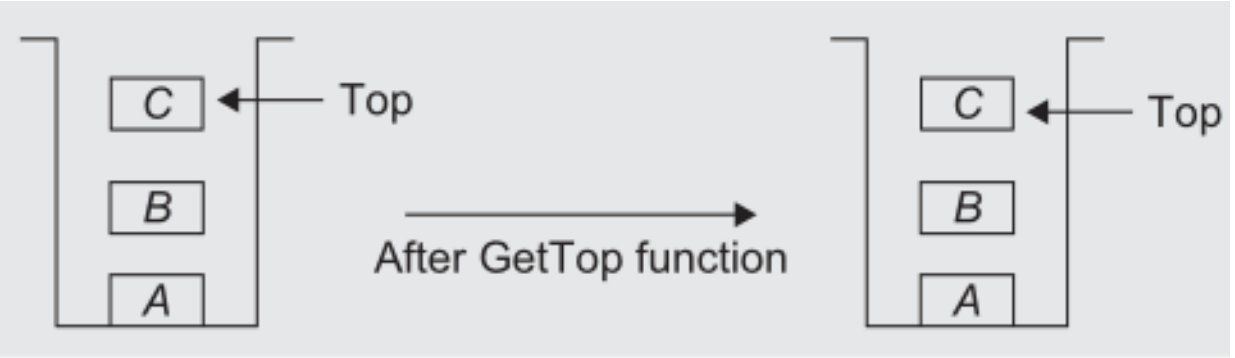


Operasi(3)

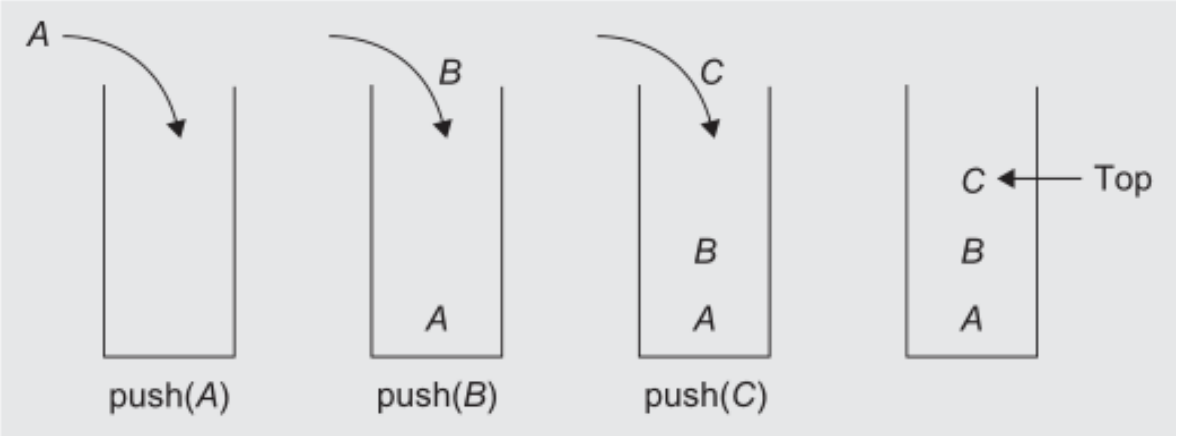
- Pop = mengambil elemen paling atas (top)



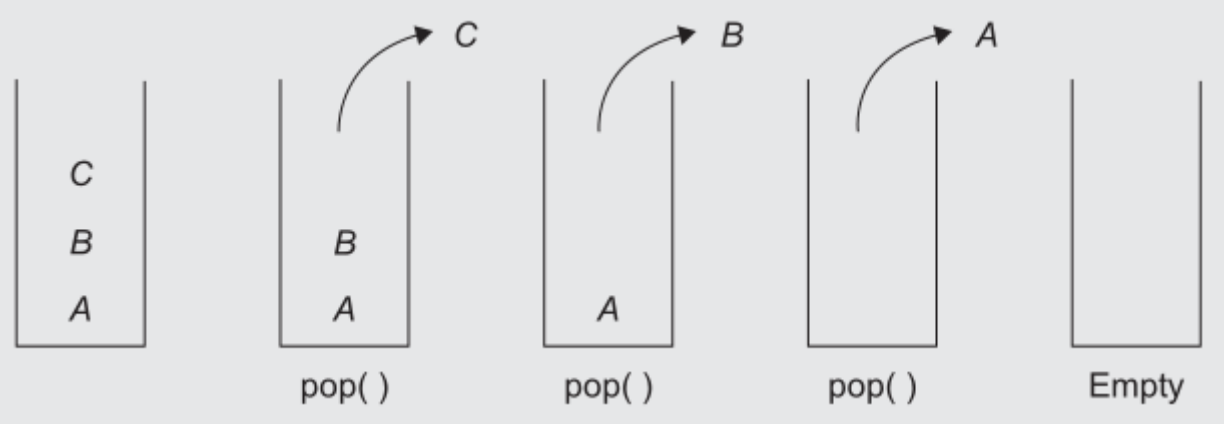
- GetTop



- Ilustrasi push



- Ilustrasi pop

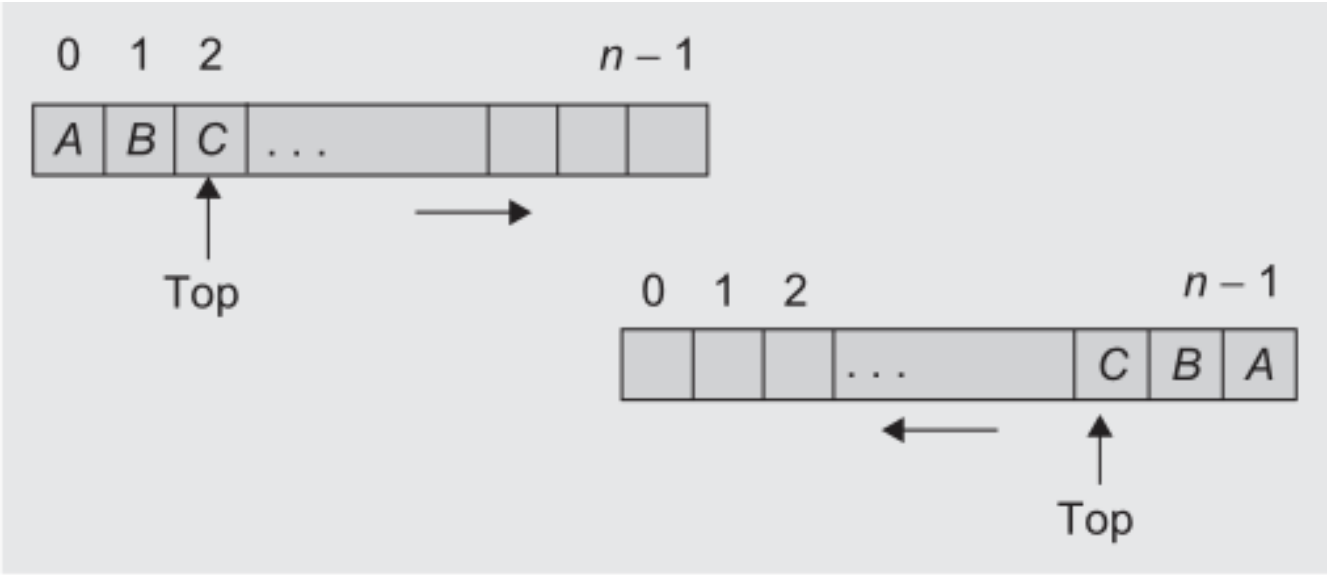


Operasi dasar sebagai ADT

Nam Operasi	Keterangan
Create (S)	Membuat stack kosong
Push(i, S)	Insert elemen i ke dalam stack S dan return Stack yang state /kondisi-nya sudah berubah
Pop (S)	Buang elemen yang ada di posisi top, update state Stack
GetTop	Return elemen yang ada di top, tanpa membuangnya
Is_Empty	Cek jika stack kosong
is_Full	Cek posisi top sudah sampai pada kapasitas Stack

Implementasi Stack

- Stack dapat diimplementasikan dengan struktur data sequential seperti Array
- Pendekatan seperti ini disebut Contiguous Stack



Implementasi(2)

- Create : alokasi memori statis dalam bentuk array (untuk menampung data), top diset -1 (stack kosong)

```
int data[ 100 ]; // kapasitas 100
int top = -1;
```

- Is empty : fungsi mengecek apakah stack kosong

```
if ( top == -1)
    return 1;
else
    Return 0;
```

Implementasi(3)

- Get Top : mengembalikan data pada top

```
if ( top == -1 )
        cout << “Stack underflow (empty)” << endl;
else
        return data[ top ];
```

- Push

```
if ( top == MaxCapacity - 1 )
     cout << “Stack overflow (full)” ;
else
{
     top++;
     data[top] = element;     //menambah pada posisi top
}
```

Implementasi (4)

- Pop

```
if ( top == -1 )
        cout << “Stack underflow\n”;
else
        return data[ top-- ];
```

- Coba Anda implementasikan **clear** : mengosongkan isi stack dan set kembali stack dalam status empty

Penerapan Stack

- Di awal slide sudah disinggung, sistem program komputer dieksekusi dengan bantuan Runtime-Stack yang melacak eksekusi program dalam bentuk activation record
- Aplikasi lain :
  - Membalik string
  - Cek pasangan delimiter (pembatas) pada bahasa C++
  - Big integer

Membalik string

- Dengan stack, kita bisa membalik string tanpa perlu melakukan looping
- Loop hanya cukup ketika membaca data awal karakter penyusun string
- Tiap kali kita membaca karakter, hasil baca tersebut di push ke stack
- Setelah data selesai terbaca, stack kita balik pop sehingga print ke output akan menghasilkan urutan terbalik

Contoh membalik string

Input	Action	Stack	Display
ABCDEF	Push *A*	*A* ← top of stack	–
BCDEF	Push *B*	*AB* ← top of stack	–
CDEF	Push *C*	*ABC* ← top of stack	–
DEF	Push *D*	*ABCD* ← top of stack	–
EF	Push *E*	*ABCDE* ← top of stack	–
F	Push *F*	*ABCDEF* ← top of stack	–
End	Pop and display	*ABCDE* ← top of stack	*F*
	Pop and display	*ABCD* ← top of stack	*FE*
	Pop and display	*ABC* ← top of stack	*FED*
	Pop and display	*AB* ← top of stack	*FEDC*
	Pop and display	*A* ← top of stack	*FEDCB*
	Pop and display	Stack empty	*FEDCBA*
	Stop		

Queue (antrian)

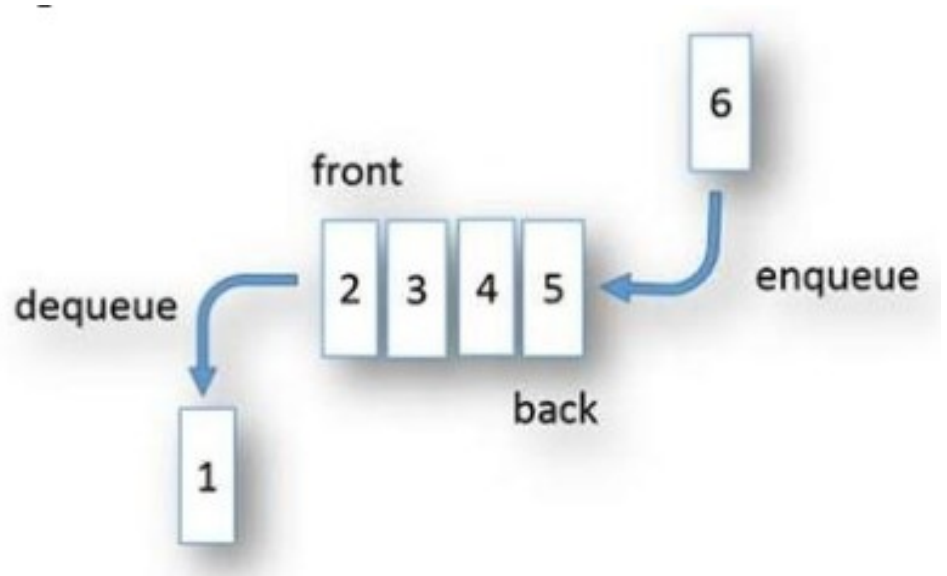
- Antrian orang pada layanan publik
- Yang pertama masuklah yang dilayani

Queue(2)

- Antrian diperlukan karena resource yang tersedia terbatas sedangkan jumlah request lebih banyak
- Yang pertama join dalam antrian adalah yang pertama diberi layanan dan keluar (dari antrian)
- **FIFO** : First In First Out
- Queue memiliki banyak penerapan dalam dunia nyata karena untuk mengatur sistem layanan
- Queue juga banyak digunakan dalam teknik informatika yaitu bidang studi Simulasi

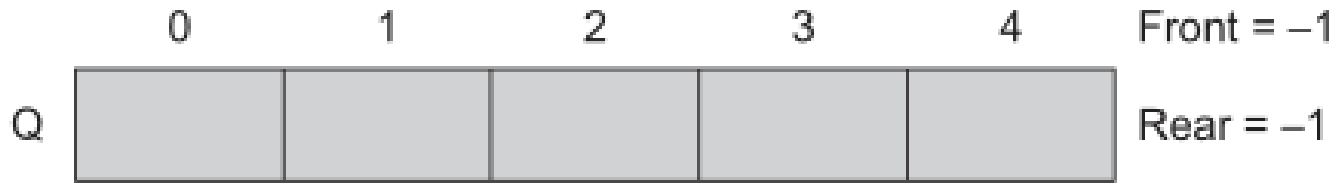
Queue sebagai ADT

Operasi / fungsi	Keterangan
Is Empty	Cek apakah kondisi queue kosong
enqueue	Memasukkan elemen baru ke bagian paling belakang
dequeue	Ambil elemen yang ada dibagian depan
First Elemen	Mengembalikan elemen depan tanpa membuang dari queue
clear	Menghapus seluruh isi queue dan reset ke kondisi empty



Contoh operasi pada Queue

- Create Empty(kapasitas=5)

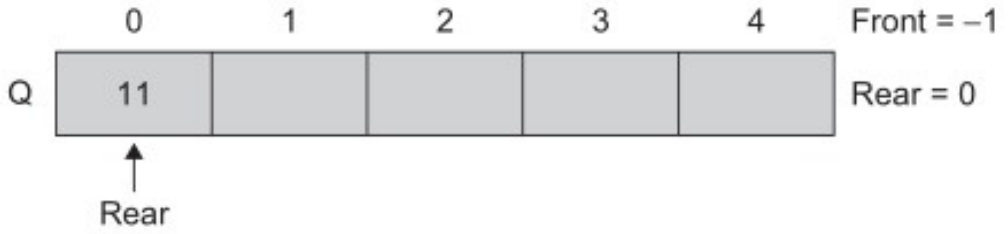


- enqueue and dequeue

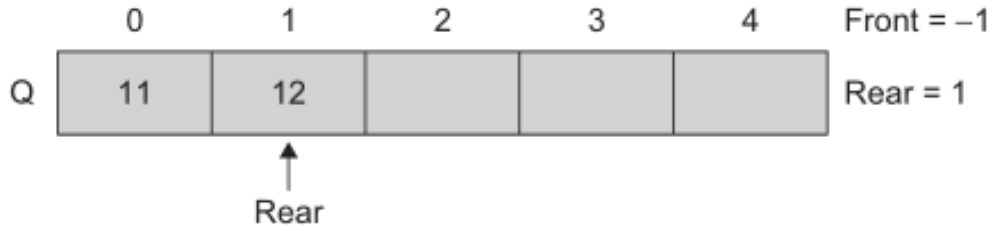
Operation	Front	Queue after operation
aQueue = *an empty queue*	↓	
aQueue.enqueue(5)		5
aQueue.enqueue(2)		5 2
aQueue.enqueue(7)		5 2 7
aQueue.peekFront()		5 2 7 (Returns 5)
aQueue.dequeue()		2 7
aQueue.dequeue()		7

Contoh operasi(2)

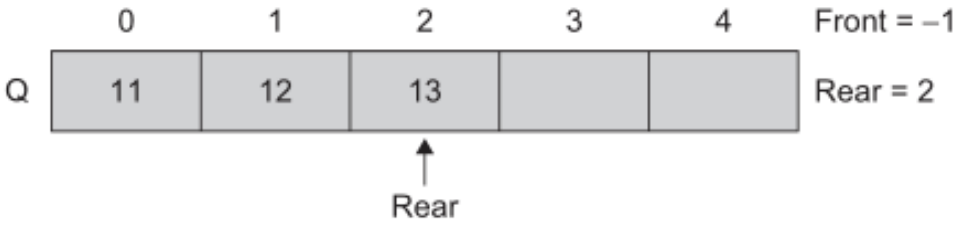
- Enqueue (11)



- Enqueue (12)

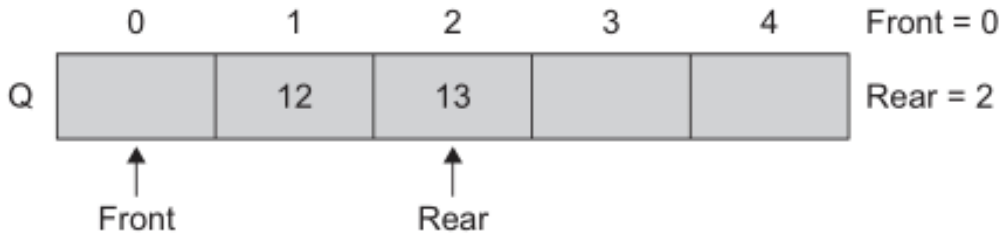


- Enqueue (13)

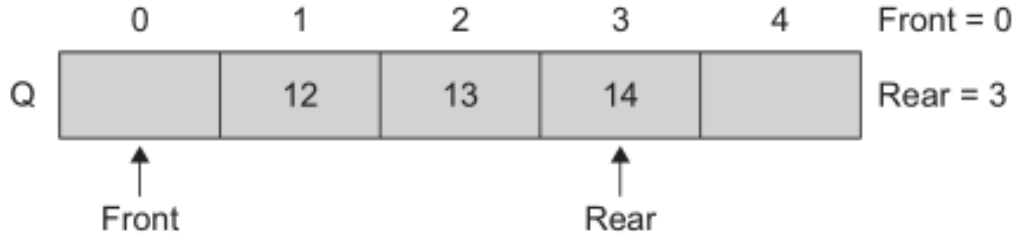


Contoh operasi(3)

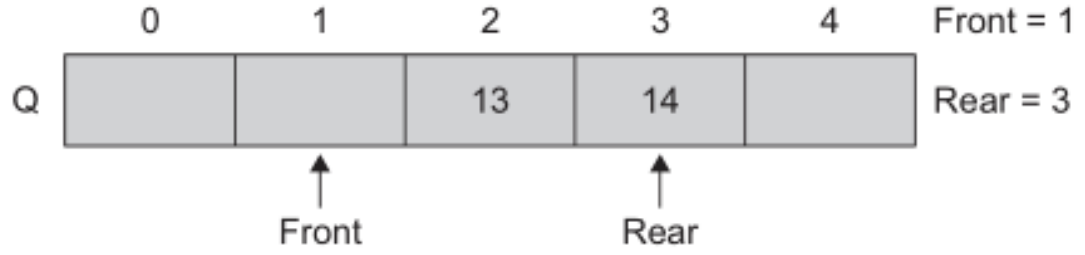
- Dequeue



- Enqueue (14)



- dequeue



Contoh implementasi Queue

- Create/constructor : alokasi memori static (array)

```
#define max 50
int data[ max ];
int Front = Rear = -1;
```

- Is empty

```
bool is_empty( )
{
      If( Front == Rear )
            return 1;
      Else
            return 0;
}
```

Implementasi (2)

- Is Full

```
Bool is_Full( )
{
    If( Rear == max – 1 ) // elemen belakang mencapai maksimum
        Return 1;
    Else
        Return 0;
}
```

Add / enqueue

```
Void add( int Element )
{
     If ( is_Full( ) )
         Cout << “Error Queue is full”;
     Else
         Queue[ ++Rear] = Element;
}
```

Implementasi(3)

- Delete / dequeue

```
int dequeue( )
{
    if ( is_empty( ) )
        cout << “Queue is empty”;
    else
        return (data[ ++Front ] );
}
```

- Get front

```
int getFront( )
{
    if( Is_empty( ) )
        cout << “Queue is empty”;
    else
        return (data[Front]);]
}
```

Referensi

- Adam Drozdek
- Hermant Jain
- Varsha patil
- D.S Malik