

## Modul 7 STACK

### TUJUAN PRAKTIKUM

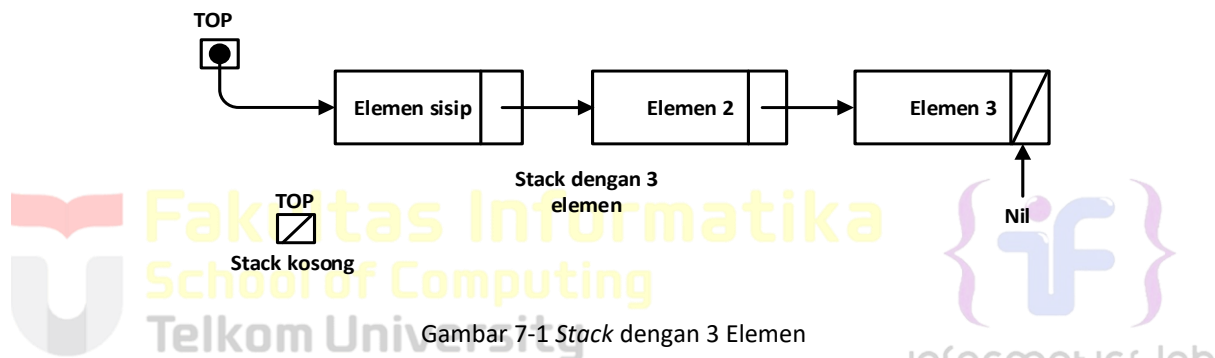
1. Memahami konsep *stack*.
2. Mengimplementasikan *stack* dengan menggunakan representasi *pointer* dan tabel.
3. Memahami konsep *queue*.
4. Mengimplementasikan *queue* dengan menggunakan *pointer* dan tabel.

### 7.1 Pengertian Stack

*Stack* merupakan salah satu bentuk struktur data dimana prinsip operasi yang digunakan seperti tumpukan. Seperti halnya tumpukan, elemen yang bisa diambil terlebih dahulu adalah elemen yang paling atas, atau elemen yang pertama kali masuk, prinsip ini biasa disebut **LIFO** (*Last In First Out*).

### 7.2 Komponen-Komponen dalam Stack

Komponen – komponen dalam *stack* pada dasarnya sama dengan komponen pada *single linked list*. Hanya saja akses pada *stack* hanya bisa dilakukan pada awal *stack* saja.



Gambar 7-1 Stack dengan 3 Elemen

Seperti terlihat pada gambar diatas bentuk *stack* mirip seperti *list* linier, yang terdiri dari elemen – elemen yang saling terkait. Komponen utama dalam *stack* yang berfungsi untuk mengakses data dalam *stack* adalah elemen paling awal saja yang disebut “Top”. Pendeklarasian tipe data *stack*:

```
1  #ifndef stack_H
2  #define stack_H
3
4  #define Nil NULL
5  #define info(P) (P)->info
6  #define next(P) (P)->next
7  #define Top(S) ((S).Top)
8
9  typedef int infotype; /* tipe data dalam stack */
10 typedef struct elmStack *address;
11 /* tipe data pointe untuk elemen stack */
12 struct elmStack {
13     infotype info;
14     address next;
15 }; /*tipe data elemen stack */
16
17 /* pendeklarasian tipe data stack */
18 struct stack {
19     address Top;
20 };
21 #endif
```

Keterangan:

1. Dalam *stack* hanya terdapat TOP.

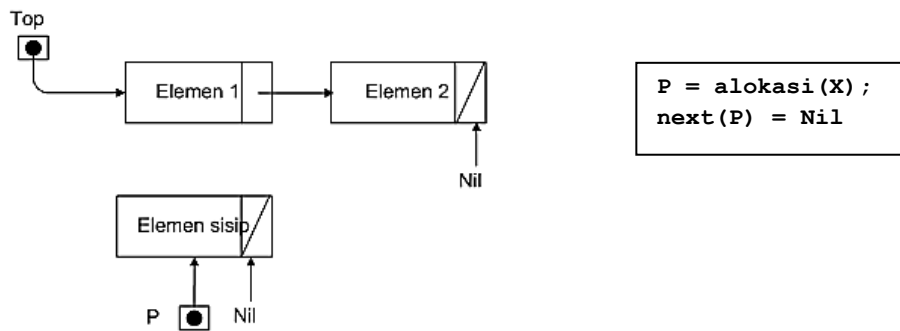
2. Tipe *address* adalah tipe elemen *stack* yang sama dengan elemen dalam *list* lainnya.

### 7.3 Operasi-Operasi dalam Stack

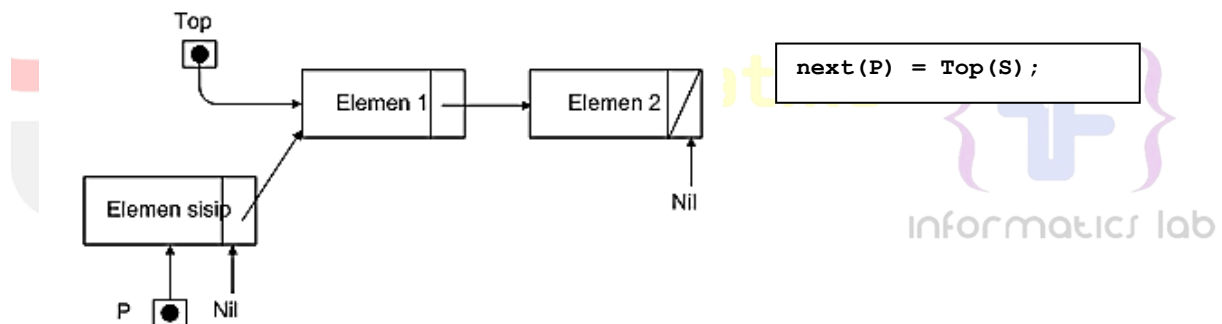
Dalam *stack* ada dua operasi utama, yaitu operasi penyisipan (*Push*) dan operasi pengambilan (*Pop*).

#### 7.3.1 Push

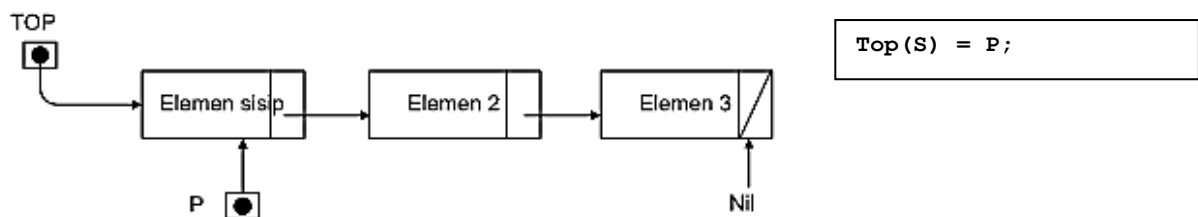
Adalah operasi menyisipkan elemen pada tumpukan data. Fungsi ini sama dengan fungsi *insert first* pada *list* biasa. Langkah – langkah dalam proses *Push*:



Gambar 7-2 Stack Push 1



Gambar 7-3 Stack Push 2



Gambar 7-4 Stack Push 3

```

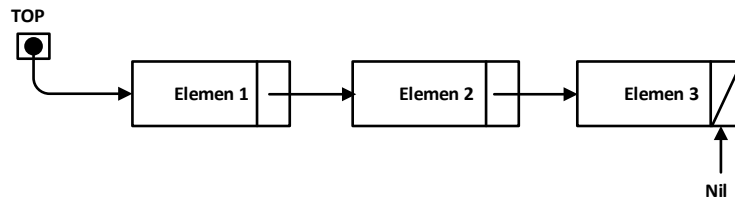
/* buat dahulu elemen yang akan disisipkan */
address createElm(int x){
    address p = alokasi(x);
    next(p) = null;
    return p;
}

/* contoh sintak push */
void push(address p){
    next(p) = top(s);
    top(s) = p;
}

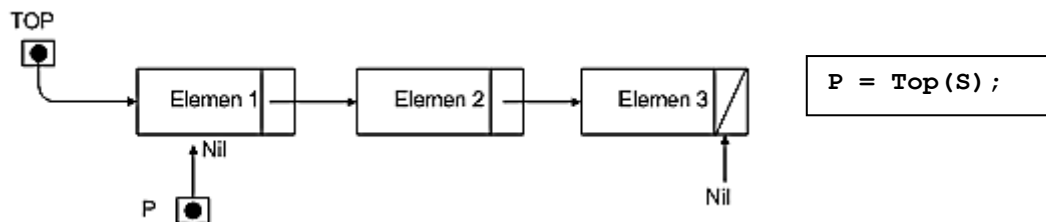
```

### 7.3.2 Pop

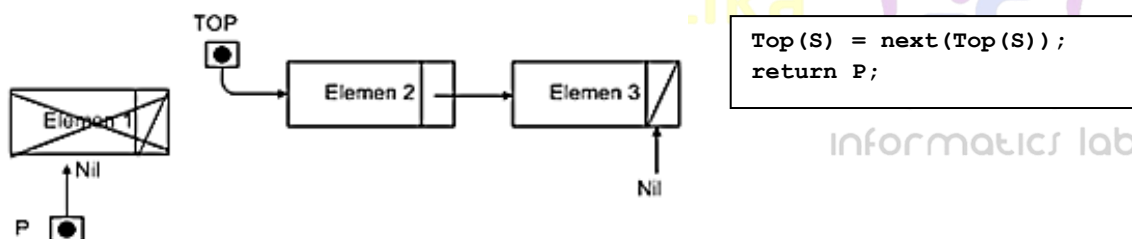
Adalah operasi pengambilan data dalam *list*. Operasi ini mirip dengan operasi *delete first* dalam *list linear*, karena elemen yang paling pertama kali diakses adalah elemen paling atas atau elemen paling awal saja. Langkah-langkah dalam proses *Pop*:



Gambar 7-5 Stack Pop 1



Gambar 7-6 Stack Pop 2



Gambar 7-7 Stack Pop 3

```
/* contoh sintak pop */
address pop(address p) {
    p = top(s);
    top(s) = next(top(s));
    return p;
}
```

## 7.4 Primitif-Primitif dalam Stack

Primitif-primitif dalam *stack* pada dasarnya sama dengan primitif-primitif pada *list* lainnya. Malahan primitif dalam *stack* lebih sedikit, karena dalam *stack* hanya melakukan operasi-operasi terhadap elemen paling atas.

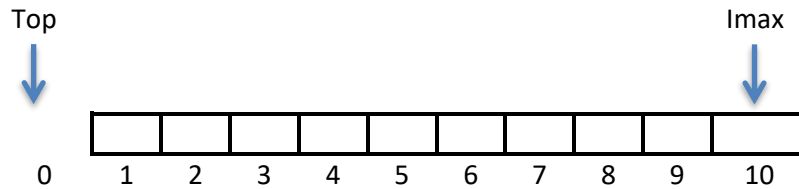
Primitif -primitif dalam *stack* :

1. createStack().
2. isEmpty().
3. alokasi().
4. dealokasi().
5. Fungsi – fungsi pencarian.
6. Dan fungsi – fungsi primitif lainnya.

Seperti halnya pada model *list* yang lain, primitif-primitifnya tersimpan pada *file \*.c* dan *file \*.h*.

## 7.5 Stack (Representasi Tabel)

Pada prinsipnya representasi menggunakan tabel sama halnya dengan menggunakan *pointer*. Perbedaannya terletak pada pendeklarasian struktur datanya, menggunakan *array* berindeks dan jumlah tumpukan yang terbatas.



Gambar 7-8 Stack Kosong dengan Representasi Table

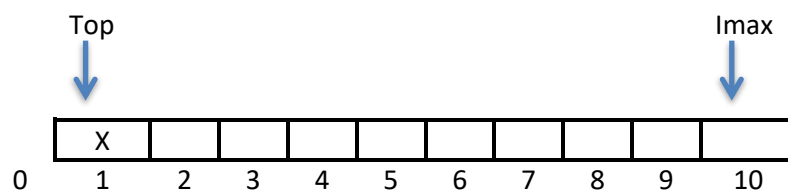
Gambar di atas menunjukkan *stack* maksimum terdapat pada indeks  $lmax=10$ , sedangkan *Stack* masih kosong karena  $Top = 0$ .

### 7.5.1 Operasi-operasi Dalam Stack

Operasi-operasi dalam *stack* representasi tabel pada dasarnya sama dengan representasi *pointer*, yaitu **PUSH** dan **POP**.

#### A. Push

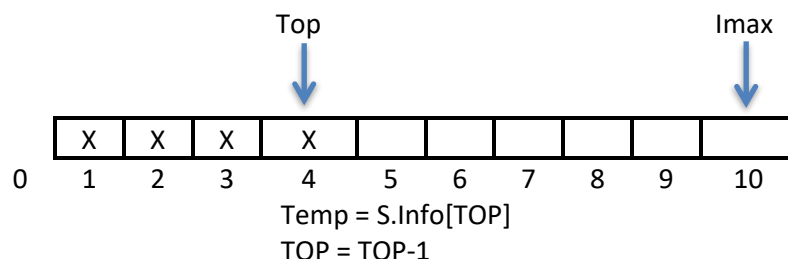
*Push* merupakan operasi penyisipan data ke dalam *stack*, penyisipan dilakukan dengan menggeser indeks dari TOP ke indeks berikutnya. Perhatikan contoh dibawah ini:

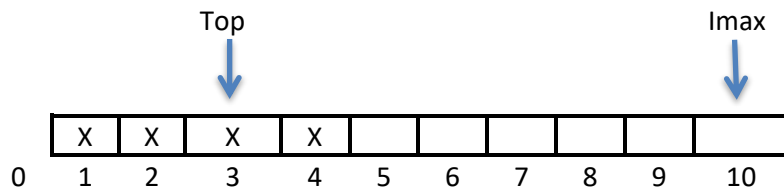


Gambar 7-9 Push Elemen dengan Representasi Tabel

#### B. Pop

*Pop* merupakan operasi pengambilan data di posisi indeks TOP berada dalam sebuah *stack*. Setelah data diambil, indeks TOP akan bergeser ke indeks sebelum TOP tanpa menghilangkan *info* dari indeks TOP sebelumnya. Perhatikan contoh dibawah ini:





Gambar 7-10 Pop Elemen dengan Representasi Tabel

## 7.5.2 Primitif-primitif Dalam Stack

Primitif-primitif pada *stack* representasi tabel pada dasarnya sama dengan representasi *pointer*. Perbedaan hanya pada manajemen memori, pada representasi tabel tidak diperlukan manajemen memori. antara lain sebagai berikut,

1. createStack().
2. isEmpty().
3. Fungsi – fungsi pencarian.
4. Dan fungsi – fungsi primitif lainnya.

Seperti halnya pada model *list* yang lain, primitif-primitifnya tersimpan pada *file \*.c* dan *file \*.h*.

Untuk lebih memahami struktur data dari *stack* representasi tabel, berikut ini contoh ADT *stack* representasi tabel dalam *file \*.h*:

```

1  /* file : stack.h
2      contoh ADT stack dengan representasi tabel */
3  #ifndef STACK_H_INCLUDED
4  #define STACK_H_INCLUDED
5
6  #include <stdio.h>
7  #include <conio.h>
8  struct infotype {
9      char nim[20];
10     char nama[20];
11 };
12 struct Stack {
13     infotype info[10];
14     int Top;
15 };
16
17 /* prototype */
18 /***** pengecekan apakah Stack kosong *****/
19 int isEmpty(Stack S);
20 /* mengembalikan nilai 0 jika stack kosong */
21 /***** pembuatan Stack *****/
22 void createStack(Stack &S);
23 /* I.S. sembarang
24     F.S. terbentuk stack dengan Top=0 */
25 /***** penambahan elemen pada Stack *****/
26 void push(Stack &S, infotype X);
27 /* I.S. Stack mungkin kosong
28     F.S. menambahkan elemen pada stack dengan nilai X, TOP=TOP+1 */
29 /***** penghapusan elemen pada list *****/
30 void pop(Stack &S, infotype &X);
31 /* I.S. stack tidak kosong
32     F.S. nilai info pada indeks TOP disimpan pada X, kemudian TOP=TOP-1 */
33 /***** proses semua elemen list *****/
34 void viewStack(Stack S);
35 /* I.S. stack mungkin kosong
36     F.S. jika stack tidak kosong menampilkan semua info yang ada pada stack
37 */
37 #endif // STACK_H_INCLUDED

```

## 7.6 Latihan Stack

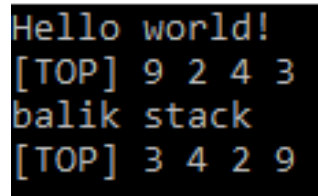
1. Buatlah ADT *Stack* menggunakan *ARRAY* sebagai berikut di dalam file “*stack.h*”:

```
Type infotype : integer
Type Stack <
    info : array [20] of integer
    top : integer
>
prosedur CreateStack( in/out S : Stack )
prosedur push( in/out S : Stack, in x : infotype)
fungsi pop( in/out S : Stack ) : infotype
prosedur printInfo( in S : Stack )
prosedur balikStack( in/out S : Stack )
```

Program 2 *Stack.h*

Buatlah implementasi ADT *Stack* menggunakan *Array* pada file “*stack.cpp*” dan “*main.cpp*”

```
int main()
{
    cout << "Hello world!" <<
endl;
    Stack S;
    createStack(S);
    Push(S,3);
    Push(S,4);
    Push(S,8);
    pop(S);
    Push(S,2);
    Push(S,3);
    pop(S);
    Push(S,9);
    printInfo(S);
    cout<<"balik stack"<<endl;
    balikStack(S);
    printInfo(S);
    return 0;
}
```

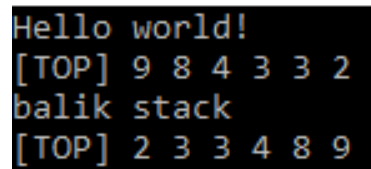


Gambar 7-11 *Output stack*

Gambar 7-12 *Main stack*

2. Tambahkan prosedur **pushAscending**( in/out S : *Stack*, in x : *integer*)

```
int main()
{
    cout << "Hello world!" << endl;
    Stack S;
    createStack(S);
    pushAscending(S,3);
    pushAscending(S,4);
    pushAscending(S,8);
    pushAscending(S,2);
    pushAscending(S,3);
    pushAscending(S,9);
    printInfo(S);
    cout<<"balik stack"<<endl;
    balikStack(S);
    printInfo(S);
    return 0;
}
```



Gambar 7-13 *Output stack push ascending*

Gambar 7-14 *Main stack dengan push ascending*

3. Tambahkan prosedur **getInputStream**( in/out S : *Stack* ). Prosedur akan terus membaca dan menerima *input* user dan memasukkan setiap *input* ke dalam *stack* hingga user menekan tombol enter. Contoh: gunakan cin.get() untuk mendapatkan inputan user.

```
int main()
{
    cout << "Hello world!" << endl;
    Stack S;
    createStack(S);
    getInputStream(S);
    printInfo(S);
    cout<<"balik stack"<<endl;
    balikStack(S);
    printInfo(S);
    return 0;
}
```

Gambar 7-16 Main stack dengan *input* stream

```
Hello world!
4729601
[TOP] 1 0 6 9 2 7 4
balik stack
[TOP] 4 7 2 9 6 0 1
```

Gambar 7-15 Output stack