BAB IV
Tumpukan (Stack)

Tujuan:

1. Memahami terminologi yang terkait dengan struktur data stack
2. Memahami operasi-operasi yang ada dalam stack
3. Dapat mengidentifikasi permasalahan-permasalahan pemrograman yang harus diselesaikan dengan menggunakan stack, sekaligus menyelesaikannya

---

Salah satu konsep yang sangat berguna dalam Ilmu Komputer adalah struktur data yang disebut stack. Dalam bab ini dibahas mengenai stack dan implementasinya dalam pemrograman.

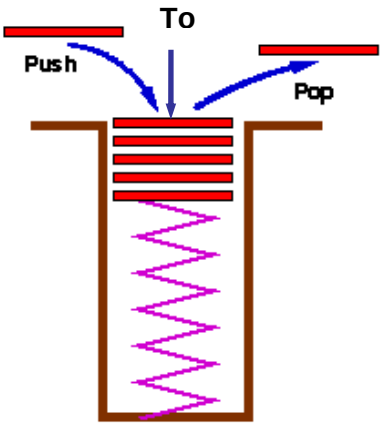
3.1 DESKRIPSI STACK

Salah satu konsep yang efektif untuk menyimpan dan mengambil data adalah "yang terakhir masuk sebagai yang pertama keluar" (*Last In First Out* / LIFO). Dengan konsep ini, pengambilan data akan berkebalikan urutannya dengan penyimpanan data.

Stack adalah sebuah kumpulan data di mana data yang diletakkan di atas data yang lain. Dengan demikian stack adalah struktur data yang menggunakan konsep LIFO, yaitu elemen terakhir yang disimpan dalam stack menjadi elemen pertama yang diambil. Dalam proses komputasi, untuk meletakkan sebuah elemen pada bagian atas dari stack, maka kita melakukan push. Dan untuk memindahkan dari tempat yang atas tersebut, kita melakukan pop.

Untuk menjelaskan pengertian di atas kita mengambil contoh sebagai berikut. Misalnya kita mempunyai dua buah kotak yang kita tumpuk, sehingga kotak kita letakkan di atas kotak yang lain. Jika kemudian stack dua buah kotak tersebut kita tambah dengan kotak ketiga dan seterusnya, maka akan kita peroleh sebuah stack kotak, yang terdiri dari N kotak.

Secara sederhana, sebuah stack bisa kita ilustrasikan seperti pada gambar 4.1 di bawah ini. Dari gambar di bawah ini, kita dapat melihat bahwa kita hanya bisa menambah atau mengambil sebuah kotak lewat satu ujung, yaitu ujung bagian atas. Nampak pula bahwa stack merupakan kumpulan data yang sifatnya dinamis, artinya kita bisa menambah atau mengambil data darinya.



Gambar 4.1 Ilustrasi Sebuah Stack

3.2 PENYAJIAN STACK

Ada beberapa cara untuk menyajikan sebuah stack tergantung pada permasalahan yang akan kita selesaikan. Kali ini kita akan menggunakan cara yang paling sederhana dengan tipe data yang sudah kita kenal, yaitu array. Kita dapat menggunakan array untuk menyajikan sebuah stack, dengan anggapan bahwa banyaknya elemen maksimum dari stack tersebut tidak akan melebihi batas maksimum banyaknya elemen dalam array.

Pada saat ukuran stack mencapai nilai maksimumnya, kalau kita teruskan menambah data lagi, akan terjadi *overflow*. Dengan demikian perlu data tambahan untuk mencatat posisi ujung stack. Dengan kebutuhan seperti ini, kita dapat menyajikan stack dengan menggunakan tipe data struktur (*struct*) yang terdiri dari dua field.

- Field pertama bertipe array untuk menyimpan elemen stack
- Field kedua bertipe integer untuk mencatat posisi ujung stack.

Dengan demikian kita mendefinisikan stack sebagai berikut :

```
#define MAXSTACK 100

typedef char itemType;          //if item bertipe char

typedef struct {                //Definisi struktur stack
  itemType data[MAXSTACK];      //Array yg berisi data tumpukan
  int indeks;                   //indeks data teratas dari stack
} stack;
```

3.3 OPERASI PADA STACK

Operasi-operasi Dasar pada stack adalah sebagai berikut:

3.3.1 Inisialisasi

Operasi inisialisasi dilakukan di awal untuk memastikan stack dalam keadaan kosong

```
void inisialisasiStack (stack *s){
   s->indeks = 0;
}
```

3.3.2 Cek Stack kosong

Fungsi yang melakukan pengecekan apakah stack dalam kondisi kosong

```
int empty (stack *s){
   if(s->indeks == 0)           //return(s->indeks == 0);
     return 1;
   else
     return 0;
}
```

3.3.3 Cek Stack Penuh

Fungsi yang melakukan pengecekan apakah stack dalam kondisi penuh

```
int full (stack *s){
    if(s->indeks == MAXSTACK)   //return(s->indeks == MAXSTACK);
       return 1;
    else
       return 0;
}
```

3.3.4 Operasi Push

Operasi push adalah operasi memasukkan elemen yang akan diletakkan pada posisi teratas dari tumpukan. Implementasinya dapat dijelaskan sebagai berikut:

- Menyisipkan data x ke dalam `s->data` pada indeks yang ditunjuk oleh `s->indeks` :
  `s->data[s->indeks] = x`
- Mengupdate posisi indeks, yaitu dengan menambah nilai `s->indeks` dengan `1` :
  `++(s->indeks)`
- Cara memanggil fungsi tsb adalah sebagai berikut : `push(x, &tumpukan);`

Fungsi selengkapnya adalah sebagai berikut :

```
void push(itemType x, stack *s) {
   if (full(s)) {                                  //stack penuh
       puts("\nSTOP!!...");
       puts("Stack PENUH!! Data terakhir gak bisa masuk");
   }
   else {
       s->data[s->indeks] = x;
       ++(s->indeks);
   }
}
```

3.3.5 Operasi Pop

Operasi pop adalah operasi untuk mengambil/menghapus elemen yang terletak pada posisi paling atas dari sebuah tumpukan. Cara pengambilan satu data teratas dengan fungsi `pop()` adalah sebagai berikut:

- Meng-update nilai `s->indeks` dengan mengurangi `1` : `--(s->indeks)`
- Mengambil isi tumpukan pada indeks teratas, dan ditampung pada variabel `tampung` :
  `tampung = s->data[s->indeks];`
- Cara memanggil fungsi tsb adalah sebagai berikut : `pop(&tumpukan);`
- Fungsi ini memberikan return value bertipe `itemType`, yaitu berupa item teratas yang berhasil di-pop

Fungsi selengkapnya adalah sebagai berikut :

```
itemType pop(stack *s){
   itemType tampung;

   if(empty(s)) {
      puts("Stack kosong, gak bisa ngambil data apapun!");
      tampung = ' ';                //spasi sbg penanda data kosong
   }
   else {
      --s->indeks;
      tampung = s->data[s->indeks];
   }
   return tampung;
}
```

3.4 Notasi POLISH

Salah satu pemanfaatan stack adalah untuk menulis ungkapan menggunakan notasi tertentu. Seperti kita ketahui, dalam penulisan ungkapan, khususnya ungkapan numeris, kita selalu menggunakan tanda kurung untuk mengelompokkan bagian mana yang akan dikerjakan lebih dahulu. Sebagai contoh, dalam ungkapan:

(A + B) * (C – D)

suku (A + B) akan dikerjakan lebih dahulu, kemudian suku (C – D), dan terakhir mengalikan hasil yang diperoleh dari dua suku tersebut.

Sedangkan pada ungkapan:

A + B * C – D

maka B * C akan dikerjakan lebih dahulu, diikuti yang lain. Dalam hal ini pemakaian tanda kurung akan sangat mempengaruhi hasil akhir. Cara penulisan ungkapan tersebut sering disebut dengan notasi infix, yang artinya adalah bahwa operator ditulis diantara dua operand.

Dalam ungkapan-ungkapan yang rumit, pemakaian tanda kurung tidak bisa dihindari. Semakin rumit suatu ungkapan semakin banyak dibutuhkan tanda kurung. Hal ini membawa suatu konsekuensi bahwa penulisan tanda kurung itupun harus benar-benar terhindar dari kesalahan.

Seorang ahli matematika yang bernama Jan Lukasiewicz kemudian mengembangkan satu cara penulisan ungkapan numeris yang selanjutnya disebut notasi Polish atau notasi prefix, yang artinya adalah operator ditulis sebelum kedua operand yang akan disajikan. Berikut disajikan beberapa contoh notasi prefix dari notasi infix:

Infix	**Prefix**
A + B	+ A B
A + B – C	- + A B C
(A + B) * (C – D)	* + A B – C D
A – B / (C * D ^ E)	- A / B * C ^ D E

Notasi lain, yang merupakan kebalikan notasi prefix, adalah notasi postfix atau notasi suffix, atau lebih dikenal dengan notasi Polish Terbalik (*Reverse Polish Notation* atau RPN). Dalam hal ini operator ditulis sesudah operand. Sama halnya dengan notasi prefix, maka dalam notasi postfix inipun tidak diperlukan adanya tanda kurung pengelompokan. Sebagai contoh, ungkapan:

(A + B) * (C – D)

dengan kurung bantuan kita peroleh:

[A B +] * [C D -]

kemudian dengan memisalkan [A B +] sebagai p, dan [C D -] sebagai q, dan kemudian dilakukan konversi dan substitusi kembali, akan diperoleh notasi postfix sebagai berikut:

A B + C D - *

Dalam hal ini urutan penulisan operator menentukan operasi mana yang harus dikerjakan lebih dahulu. Berikut ini disajikan beberapa contoh lain hasil konversi notasi infix menjadi postfix.

Infix	**Postfix**
A + B – C	A B + C -
(A + B) * (C – D)	A B + C D - *
A – B / (C * D ^ E)	A B C D E ^ * / -

3.5 Algoritma Merubah Notasi Infix Menjadi Notasi Postfix

Adapun algoritma untuk merubah notasi infix menjadi notasi postfix dapat dijelaskan sebagai berikut :

1. Baca ungkapan dalam notasi infix, misalnya S, tentukan panjang ungkapan tersebut, misalnya N karakter, siapkan sebuah stack kosong dan siapkan derajad masing-masing operator, misalnya: ‘^’ (pangkat) berderajad 3, ‘*’ dan ‘/’ berderajad 2, ‘+’ dan’ –‘ berderajad 1 dan ‘(‘ berderajad 0.

2. Dimulai dari i = 1 sampai N kerjakan langkah-langkah sebagai berikut:
   a. `R = S[i]`
   b. Test nilai R. Jika R adalah:
      operand : langsung ditulis
      kurung buka : push ke dalam tumpukan
      kurung tutup : pop dan tulis semua isi tumpukan sampai ujung tumpukan = ‘(‘. Pop juga tanda ‘(‘ ini, tetapi tidak usah ditulis

operator : jika tumpukan kosong atau derajad R lebih tinggi dibanding derajad ujung tumpukan, push operator ke dalam tumpukan. Jika tidak, pop ujung tumpukan dan tulis; lakukan selama derajat r lebih rendah dari derajat ujung tumpukan. Kemudian R di-push

c. Jika akhir notasi infix telah tercapai, dan tumpukan masih belum kosong, pop semua isi tumpukan dan tulis hasilnya

Untuk memahami algoritma di atas, kita coba mengubah ungkapan berikut, yang ditulis menggunakan notasi infix, menjadi notasi postfix

(A + B) / ((C – D) * E ^ F)

Ilustrasi pengubahan notasi infix di atas menjadi notasi postfix secara lengkap tersaji dalam tabel sebagai berikut:

Karakter Dibaca	**Isi Tumpukan**	**Karakter tercetak**	**Hasil Notasi Postfix Yang Terbentuk**
(	(		
A	(	A	A
+	(+		
B		B	A B
)		+	A B +
/	/		
(	/ (		
(	/ ((		
C	/ ((	C	A B + C
-	/ ((-		
D	/ ((-	D	A B + C D
)	/ (	-	A B + C D -
*	/ (*		
E	/ (*	E	A B + C D – E
^	/ (* ^		
F	/ (* ^	F	A B + C D – E F
)	/ (*	^	A B + C D – E F ^
	/ (	*	A B + C D – E F ^ *
	/		
		/	A B + C D – E F ^ * /

Dari ilustrasi di atas, bisa kita lihat bahwa notasi postfix dari ungkapan:

(A + B) / ((C – D) * E ^ F)

adalah

A B + C D – E F ^ * /

3.6 Kesimpulan

1. Stack atau tumpukan merupakan data yang sifatnya terurut yang dapat dilakukan operasi penyisipan dan penghapusan pada ujung data

2. Stack sering disebut LIFO (*Last In First Out*) yaitu elemen yang paling akhir disisipkan menjadi elemen yang paling dulu diambil

3.7 Latihan

1. Buatlah sebuah program yang melakukan konversi dari bilangan desimal ke bilangan biner, octal, heksa dengan menggunakan stack

2. Buatlah sebuah program yang melakukan pembalikan terhadap kalimat dengan menggunakan stack

   Contoh:

   Kalimat : Struktur Data

   Hasil setelah dibalik : ataD rutkurtS