



stack

==tumpukan==

Tenia wahyuningrum

st3 telkom purwokerto

www.st3telkom.ac.id

Stacks everywhere ...



definisi...

“A stack is an **ordered collection** of items into which new items may be inserted and from which items may **be deleted at one end**, called top of the stack”

* Yedidyah L, Moshe J. A., and Aaron M. Tenenbaum; Data Structures Using C and C++

definisi...

“Secara sederhana, tumpukan bisa diartikan sebagai suatu kumpulan data yang **seolah-olah** ada data yang diletakan **diatas data** yang lain.”

Satu hal yang perlu kita ingat adalah bahwa kita bisa **menambah** (menyisipkan) data, dan **mengambil** (menghapus) data lewat ujung yang sama, yang disebut sebagai ujung atas tumpukan (*top of stack*)

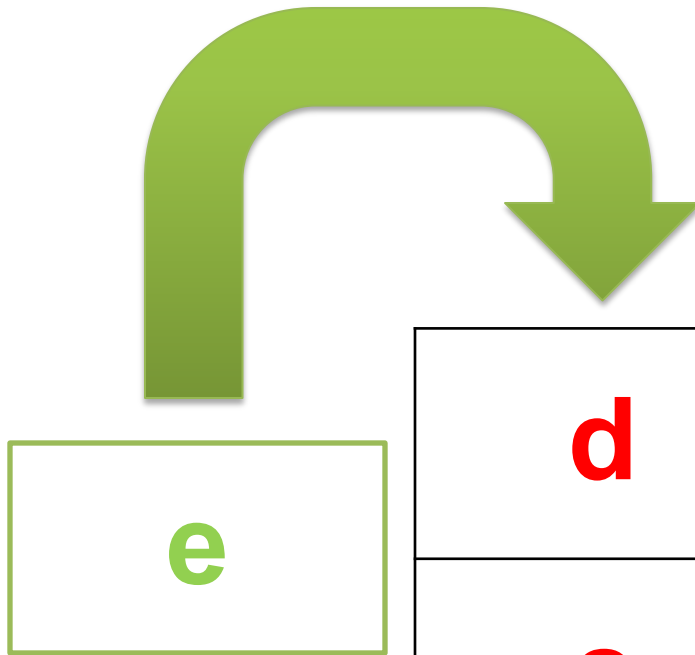


LIFO (Last In First Out)

“data yang disimpan terakhir akan diambil lebih dahulu, data yang disimpan pertama kali akan diambil paling akhir”

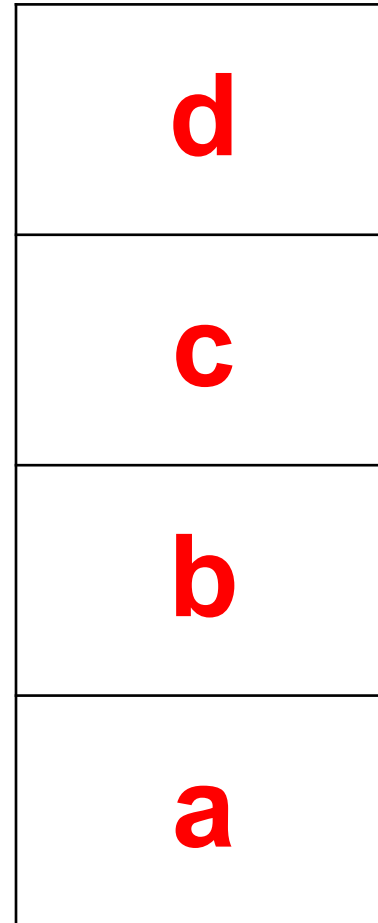


operasi stack...

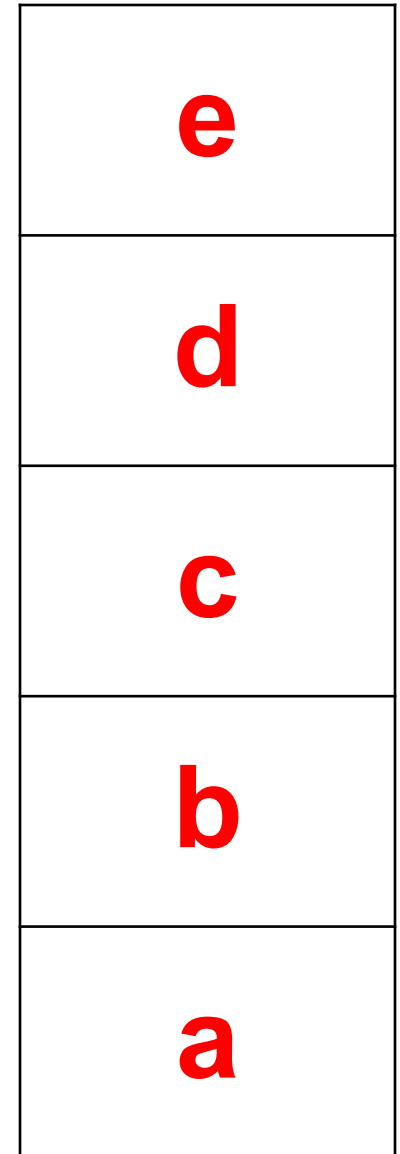


push

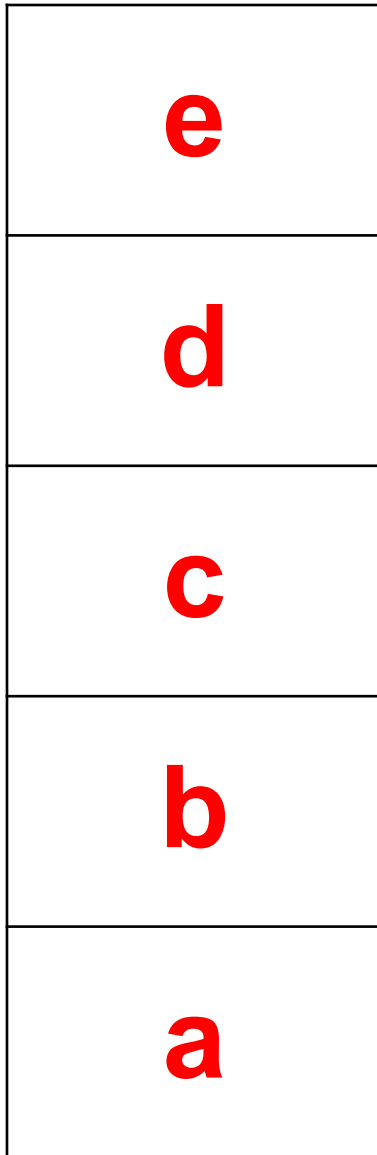
“Menyisipkan / menambahkan data pd ujung atas stack”



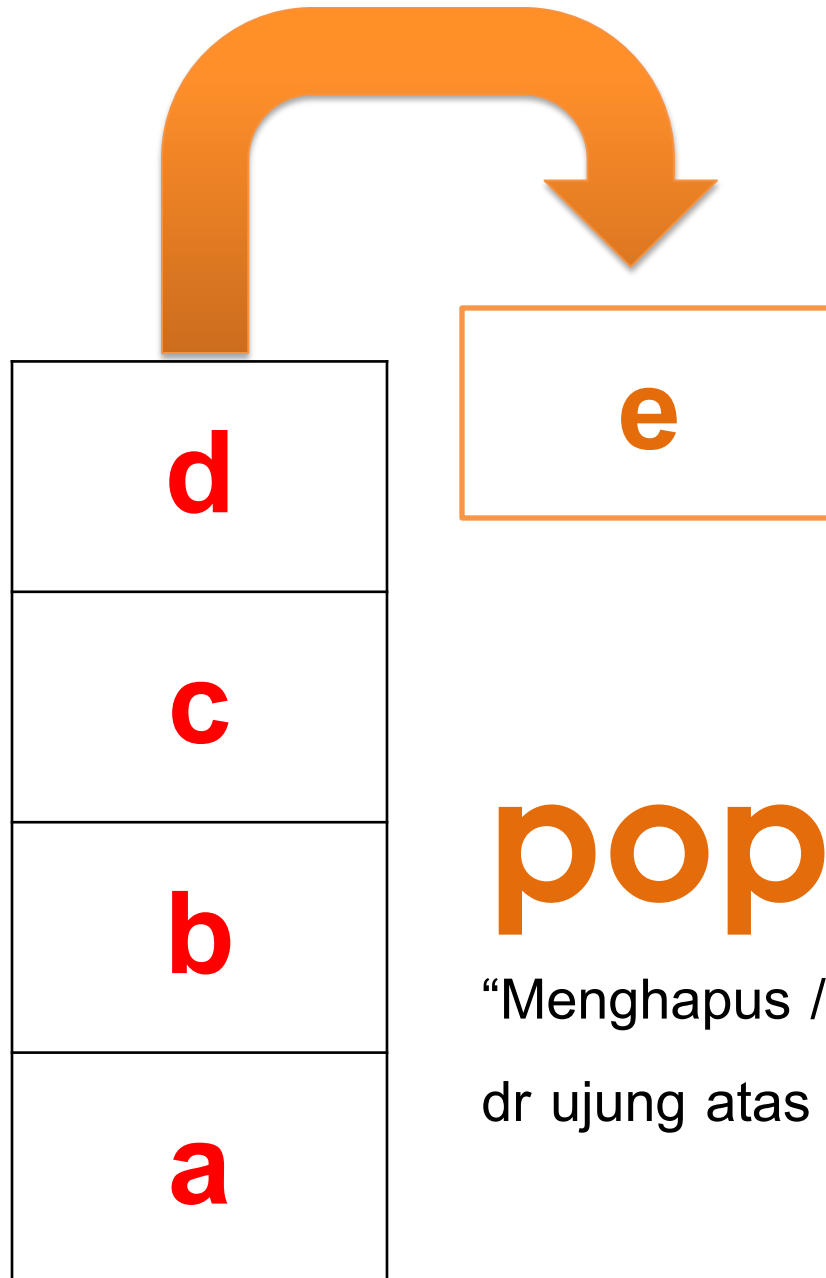
mula-mula



setelah ditumpuk



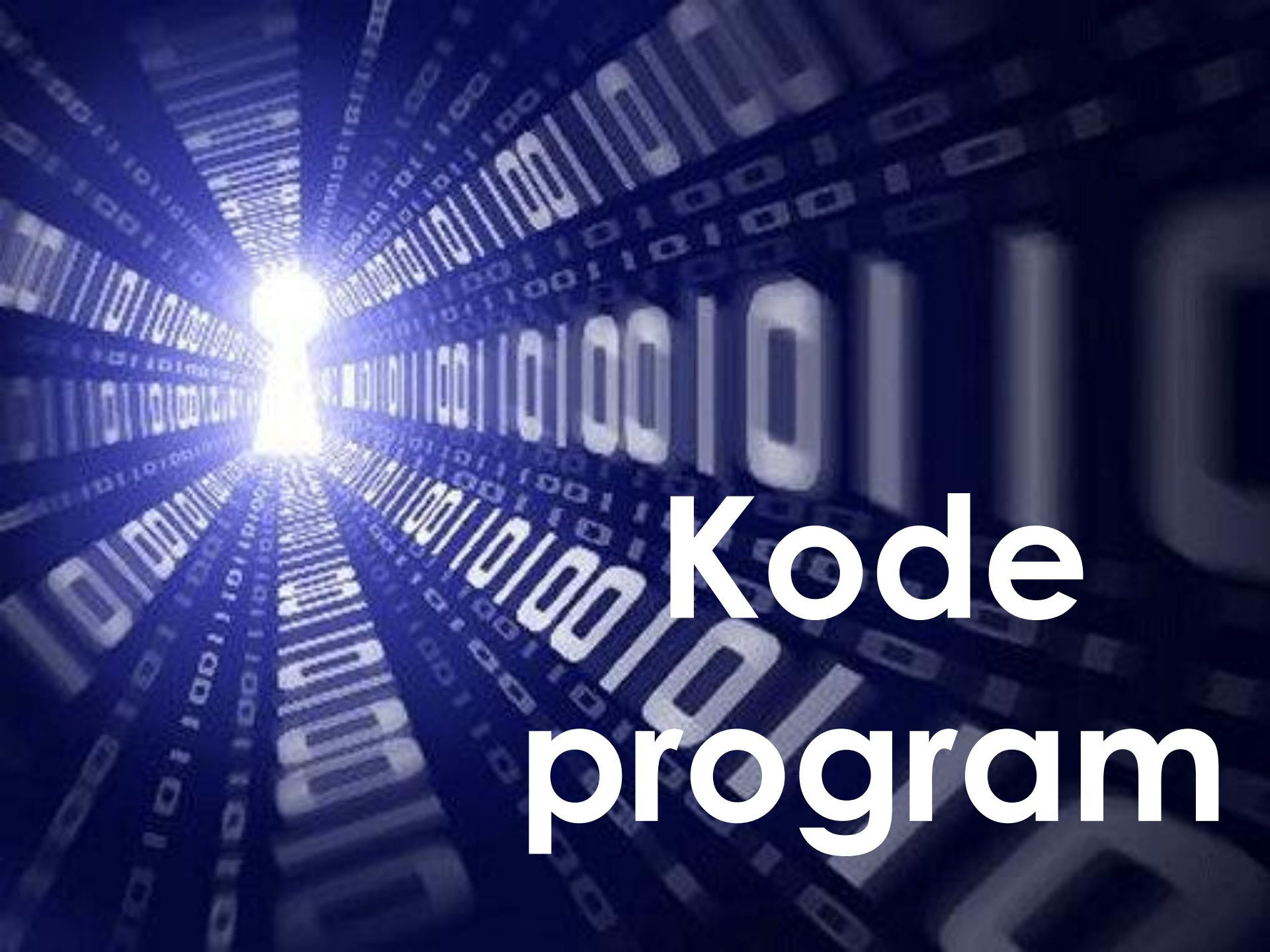
mula-mula



setelah tumpukan diambil

pop

“Menghapus / mengeluarkan data
dr ujung atas stack”



Kode program

deklarasi

```
#define max 10
```

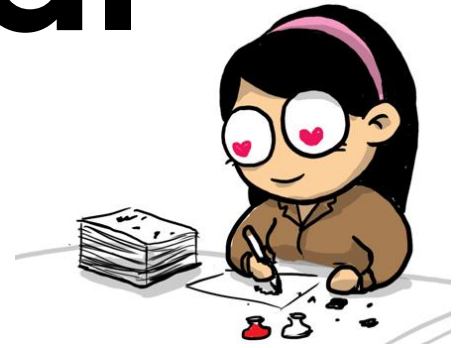
```
struct Tumpukan{  
    int atas;  
    int data[max];  
}T;
```

nilai awal

```
void awal() {  
    T.atas=-1;  
}
```

```
int kosong() {  
    if (T.atas== -1)  
        return 1;  
    else  
        return 0;  
}
```

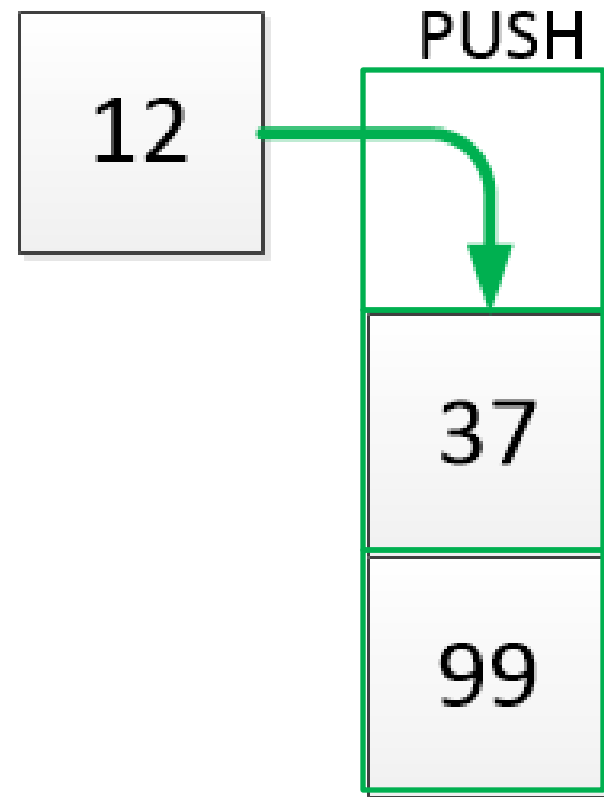
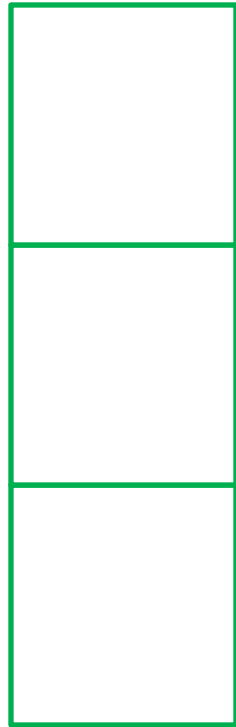
```
int penuh() {  
    if (T.atas==max-1)  
        return 1;  
    else  
        return 0;  
}
```



push

Mula-mula stack kosong

T.atas = -1
Return = 1

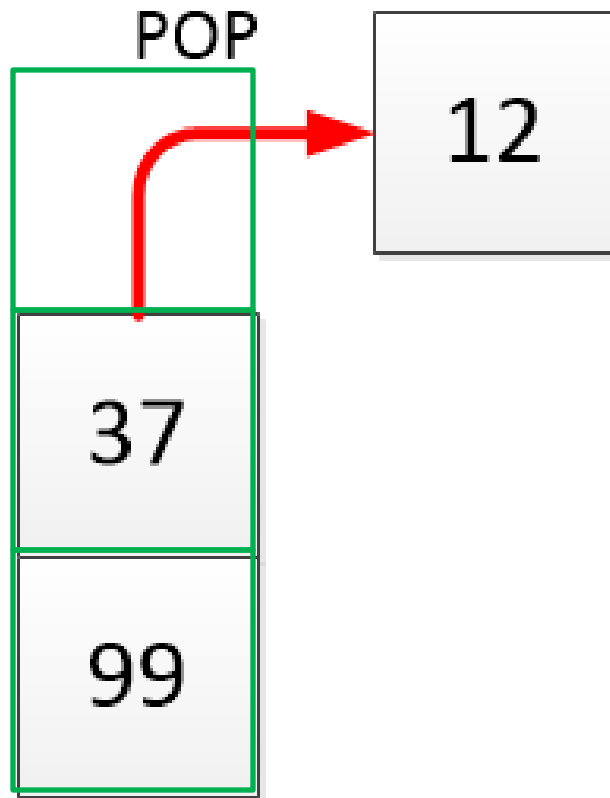


push

```
void push(int data) {  
    if (kosong() == 1)  
        {T.atas++;  
         T.data[T.atas]=data;  
         cout<<"Data " <<T.data[T.atas]<<" masuk ke stack";}  
  
    else if (penuh() == 0)  
        {T.atas++;  
         T.data[T.atas]=data;  
         cout<<"Data " <<T.data[T.atas]<<" masuk ke stack";}  
  
    else  
        cout<<"Tumpukan penuh";  
}
```



pop



T.Atas dikurangi 1



pop

```
void pop() {  
    if (kosong() == 0) {  
        cout<<"Data teratas sudah terambil";  
        T.atas--;  
    }  
    else  
        cout<<"Data kosong";  
}
```




```
void tampil() {  
if (kosong() == 0)  
    {for (int i=T.atas; i>=0; i--)  
        {cout<<T.data[i];}  
        }  
else  
    cout<<"Tumpukan kosong";  
}
```

tampil

12
37
99



- Dicetak mulai dari tumpukan teratas, lalu sampai tumpukan terbawah

Penerapan stack



Palindrom

“sebuah kata, frasa, angka maupun susunan lainnya **yang dapat dibaca dengan sama** baik dari depan maupun belakang (spasi antara huruf-huruf biasanya diperbolehkan)”

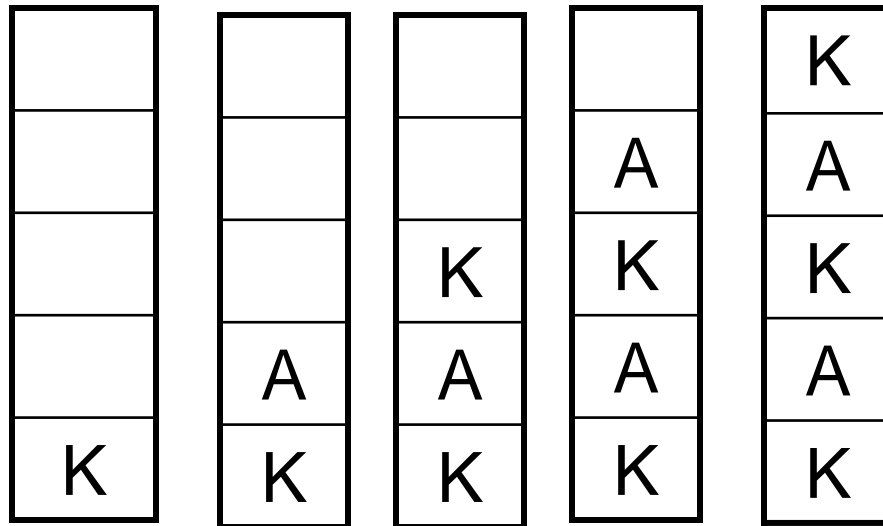
* Kata "palindrom" berasal dari bahasa Yunani: *palin* ("balik") dan *dromos* ("pacuan kuda").

contoh

- Kasur ini rusak.
- Ada asa, ada apa
- Malam ada nabababab.
- ada, apa, ara, asa, bab, ini, katak, kodok,
makam, malam, radar, taat

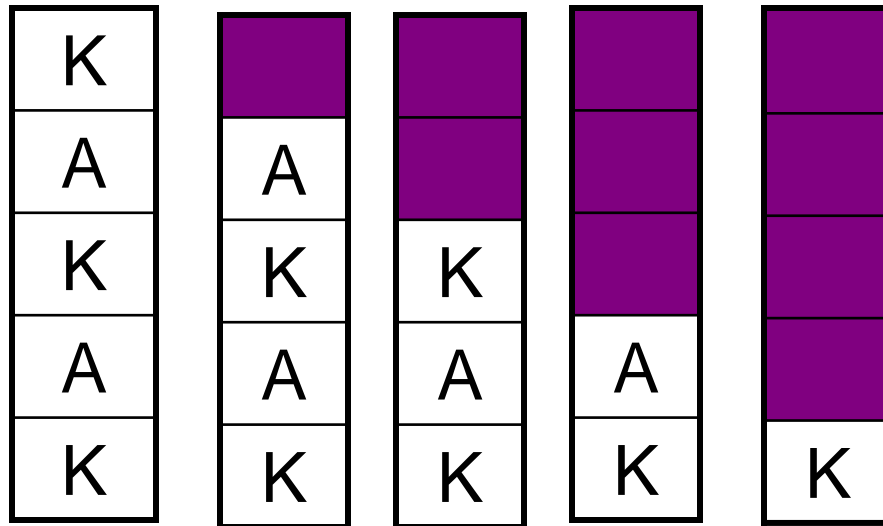
Mengecek Kalimat Polindrom

Kalimat : KAKAK



Operasi Push

Operasi Pop



Hasil = "

Hasil = K

Hasil = KA

Hasil = KAK

Hasil = KAKA

Hasil = KAKAK

Jika Kalimat = hasil

➔ Palindrom

Ungkapan Aritmatika

Untuk menuliskan ungkapan aritmatika dapat dilakukan dengan tiga metode :

infix, prefix, postfix



No.	<i>Infix</i>	<i>Postfix</i>	<i>Prefix</i>
1.	$A + B$	$A B +$	$+ A B$
2.	$(A + B) * C$	$A B + C *$	$* + A B C$
3.	$A * (B + C)$	$A B C + *$	$* A + B C$

Derajat **operator**

- $()$ ----- tanda kurung
- \wedge ----- pangkat
- $*$ dan $/$ ----- kali dan bagi
- $+$ dan $-$ ----- tambah dan kurang

Infix $A * B + C * D$

Prefix

$*AB$ + C * D

$*AB$ + $*CD$

$+*AB*CD$

Postfix

$AB*$ + C * D

$AB*$ + $CD*$

$AB*CD*+$



Infix : $A + B * (C - D) / E$

Prefix

$A + B * \underline{-CD} / E$

$A + *B-CD / E$

$A + /*B-CDE$

$+A/*B-CDE$

Postfix

$A + B * CD- / E$

$A + BCD-* / E$

$A + BCD-*E/$

$ABCD-*E/+$





Langkah manual

**infix-prefix-
postfix**

$$A * (B + C) / D ^ E - F$$

- Langkah 1: tentukan (berdasarkan derajat operasi) mana yang akan diproses terlebih dulu.
Diperoleh $(B + C)$. Jika $(B + C)$ dianggap G, maka notasi *infix* tadi menjadi:
 $A * G / D ^ E - F$
- Langkah 2: dari hasil langkah 1, disederhanakan lagi, kali ini ((berdasarkan derajat operasi) akan disederhanakan $D ^ E$. Bila $D ^ E$ dianggap H, maka notasi *infix* tadi menjadi: $A * G / H - F$
- Langkah 3: dari hasil langkah 2, disederhanakan lagi, kali ini ((berdasarkan derajat operasi) akan disederhanakan $A * G$. Bila $A * G$ dianggap I, maka notasi *infix* tadi menjadi: $I / H - F$
- Langkah 4: dari hasil langkah 3, disederhanakan lagi, kali ini ((berdasarkan derajat operasi) akan disederhanakan I / H . Bila I / H dianggap J, maka notasi *infix* tadi menjadi:
 $J - F$

Setelah diperoleh bentuk seperti itu, maka satu per satu kita kembalikan ke notasi semula sambil mengubahnya menjadi notasi *postfix*.

Langkah 5: hasil akhir $J - F$, dibentuk *postfix*nya, menjadi $J F -$

Langkah 6: J sebenarnya adalah I / H yang jika ditulis dalam bentuk *postfix* menjadi $I H /$, lalu kita gabung dengan hasil di langkah 5 tadi, diperoleh: $I H / F -$

Langkah 7: H sebenarnya adalah $D \wedge E$ yang jika ditulis dalam bentuk *postfix* menjadi $D E \wedge$, lalu kita gabung dengan hasil di langkah 6 tadi, diperoleh: $I D E \wedge / F -$

Langkah 8: I sebenarnya adalah $A * G$ yang jika ditulis dalam bentuk *postfix* menjadi $A G *$, lalu kita gabung dengan hasil di langkah 7 tadi, diperoleh: $A G * D E \wedge / F -$

Langkah 9: G sebenarnya adalah $B + C$ yang jika ditulis dalam bentuk *postfix* menjadi $B C +$, lalu kita gabung dengan hasil di langkah 8 tadi, diperoleh: $A B C + * D E \wedge / F -$

Dengan demikian, untuk notasi *infix*: $A * (B + C) / D \wedge E - F$ maka notasi *postfix*nya menjadi: $A B C + * D E \wedge / F -$

Dengan demikian, untuk notasi *infix*: $A * (B + C) / D ^ E - F$ maka notasi *postfix*nya menjadi: $A B C + * D E ^ / F -$

Postfix tidak memerlukan tanda kurung, prosesnya berjalan sebagai berikut:

$$\begin{array}{r}
 2 \ 3 \ 5 + * \ 4 \ 2 ^ / 3 - \\
 \hline
 2 \quad 8 \quad * \ 4 \ 2 ^ / 3 - \\
 16 \quad \quad 4 \ 2 ^ / 3 - \\
 \hline
 16 \quad \quad 16 / 3 - \\
 \hline
 \quad 1 \quad \quad 3 - \\
 \hline
 \quad \quad -2
 \end{array}$$

Sama hasilnya pada *infix*: $2 * (3 + 5) / 4 ^ 2 - 3 = -2$

Menggunakan stack

infix-prefix- postfix

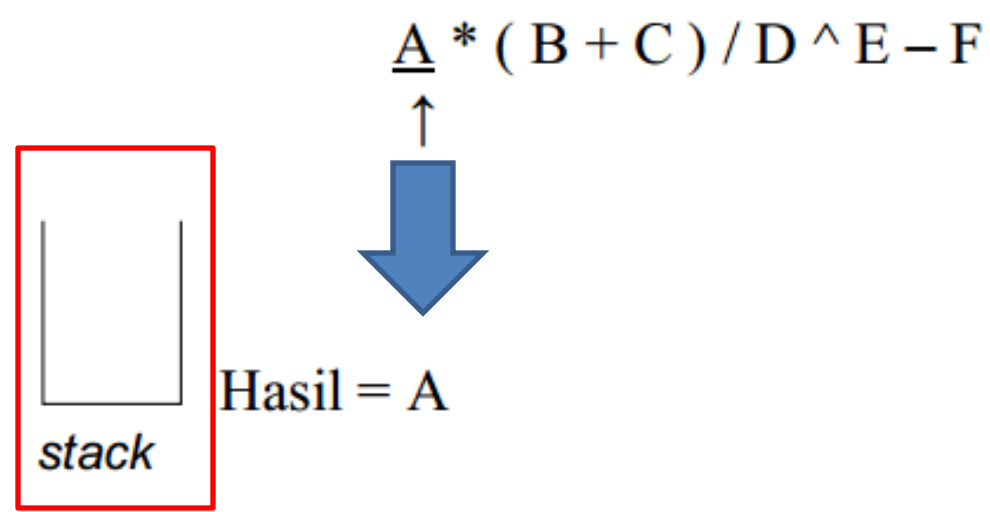


- Stack adalah tumpukan (jadi, memori diibaratkan dengan tumpukan) yang memiliki cara kerja, “yang pertama masuk ke kotak, maka akan terakhir kali diambil kembali” atau “**first in last out**”.

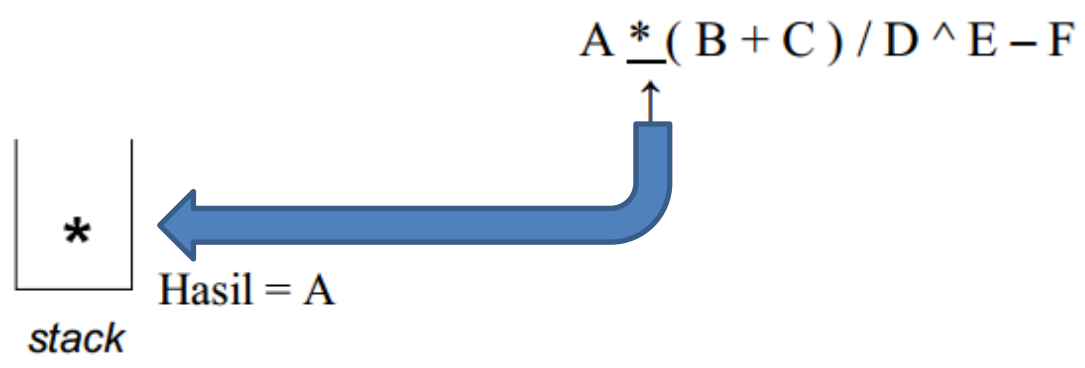
Langkah- langkah



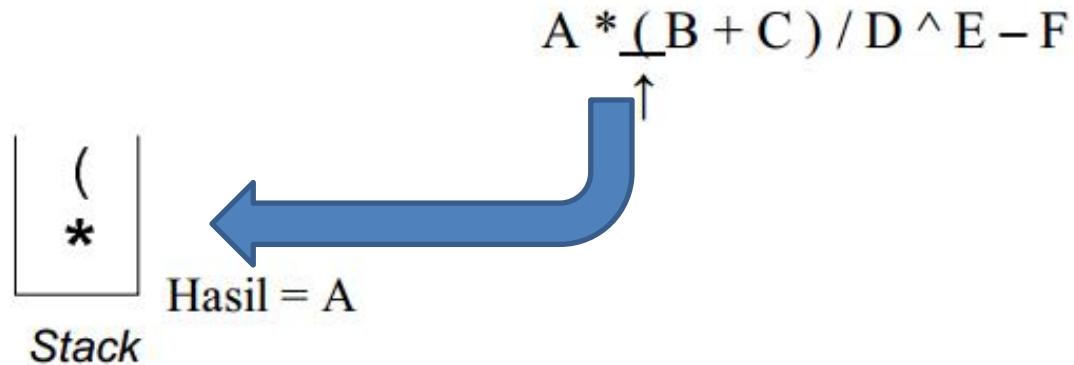
- 1. Proses akan dilakukan dari kiri ke kanan
- 2. Bila yang diproses adalah *operand*, maka tulis di hasil. Di sini *operand* “A”:



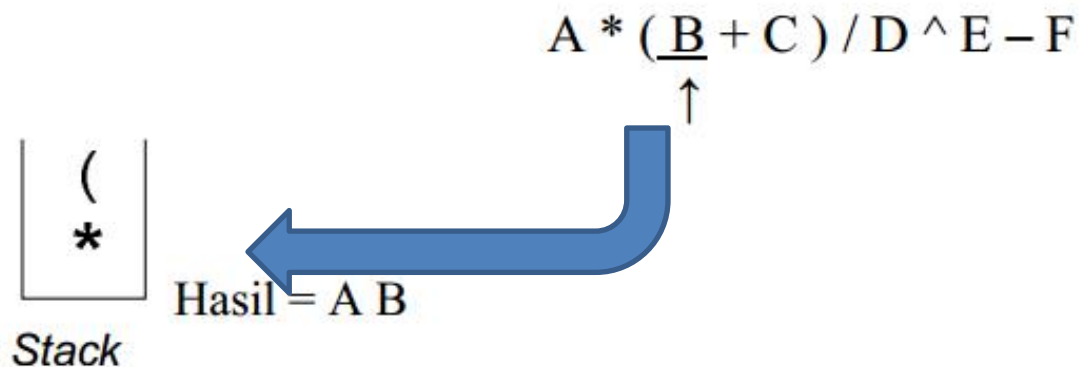
- 3. Lanjutkan ke operator “*”, karena *stack* masih dalam keadaan kosong, maka masukkan operator tersebut ke dalam *stack*;



4. Lanjutkan ke operator “(“, operator ini masukkan (tumpuk) saja ke dalam *stack*;



5. Lanjutkan ke *operand* “B”, karena sebagai *operand*, maka “B” dijadikan hasil saja.



6. Lanjutkan ke operator “+”, operator ini masukkan (tumpuk) saja ke dalam *stack*;

$$A * (B \pm C) / D ^ E - F$$



Bila *top stack* (posisi teratas tumpukan) adalah “(“ maka apapun operator yang sedang diproses, masukkan saja ke dalam *stack*.

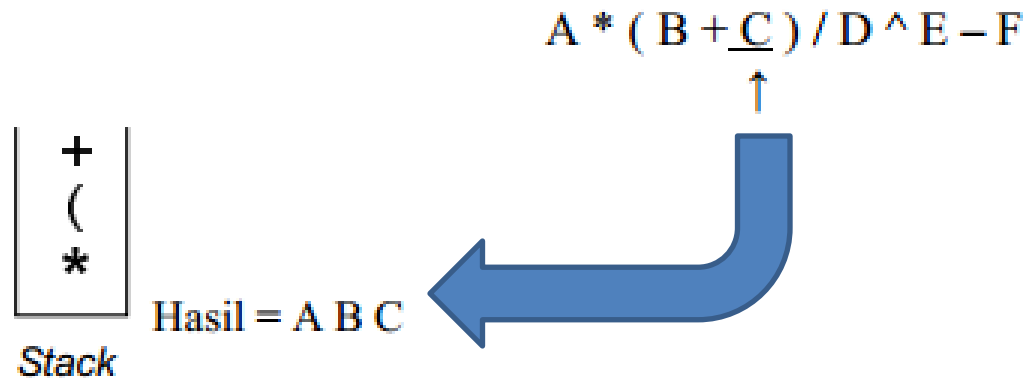
+
(
*

Stack

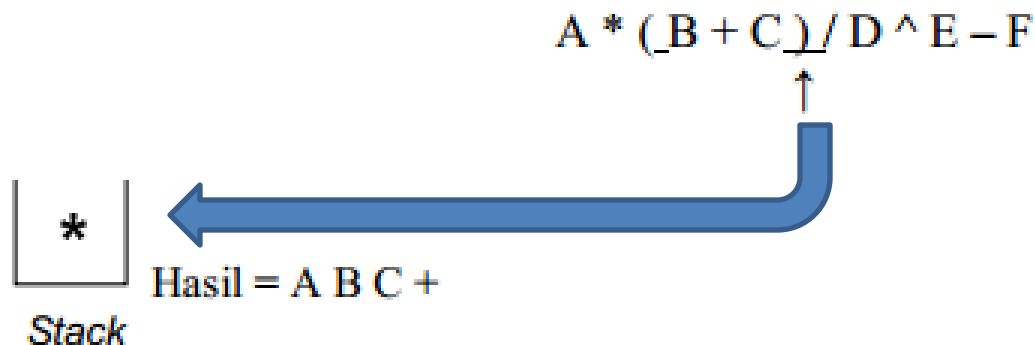
Hasil = A B



7. Lanjutkan ke *operand* “C”, karena sebagai *operand*, maka “C” dijadikan hasil saja.

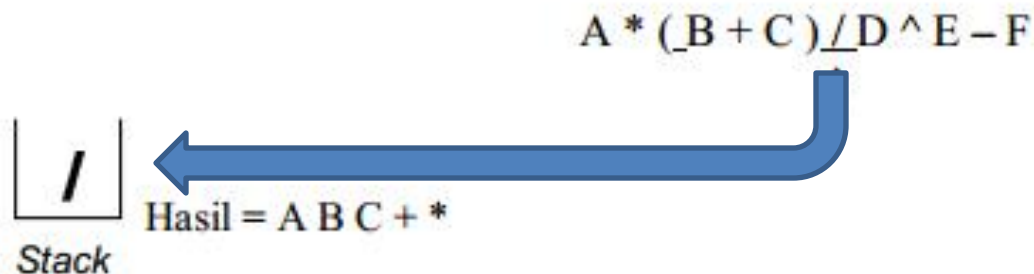


8. Lanjutkan ke operator “)”, operator ini akan mengeluarkan seluruh isi *stack* (mulai dari atas) hingga bertemu operator “(“ yang menjadi pasangannya. Karena di antara “(“ dan “)” hanya ada “+” maka “+” saja yang dijadikan hasil. Tanda kurung dibuang saja.

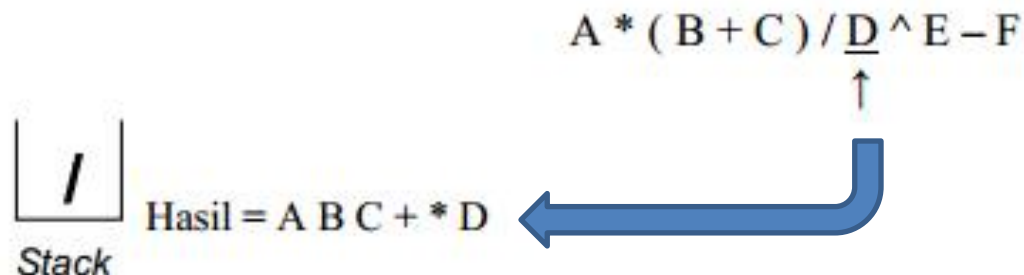


9. Lanjutkan ke operator “/”, operator ini akan dimasukkan ke dalam *stack*. Karena di *top stack* sudah ada isinya, maka bandingkan keduanya. Bila yang akan masuk memiliki derajat yang lebih besar, maka tumpuk saja. Sebaliknya, bila yang akan masuk memiliki derajat yang sama atau lebih kecil, maka keluarkan *top stack* hingga operator yang berada di *top stack* berderajat lebih kecil dari operator yang akan masuk.

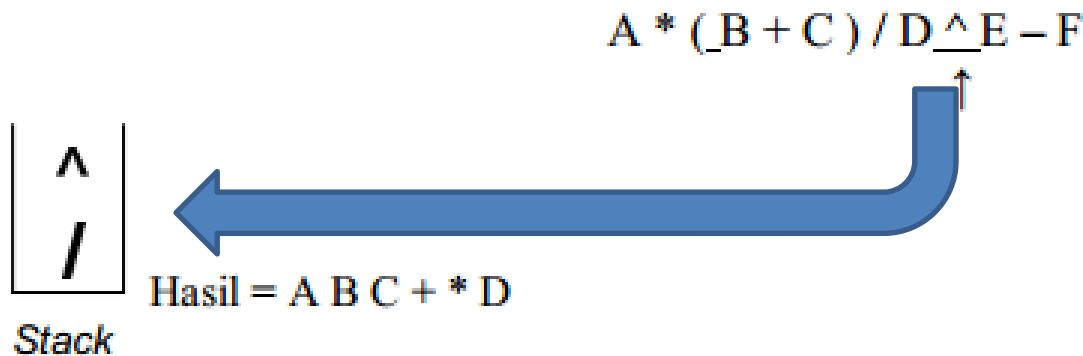
Karena “/” berderajat sama dengan “*” maka keluarkan *top stack* (“*”). Karena *stack* sudah hampa, maka operator “/” dimasukkan ke dalam *stack* sebagai *top stack*nya.



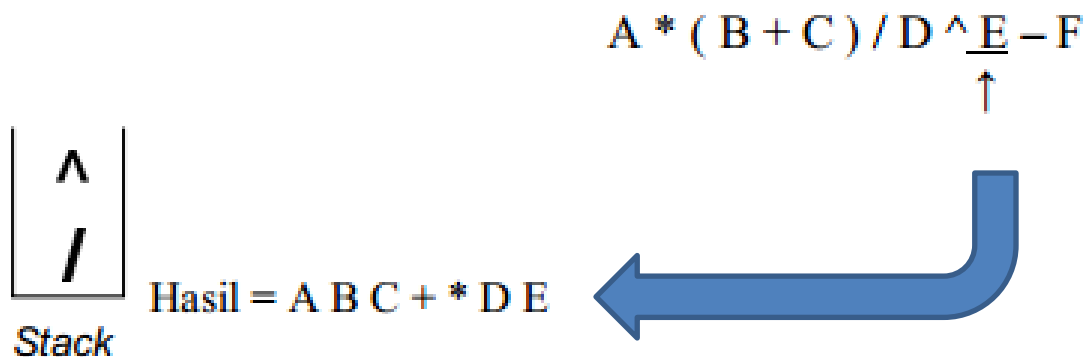
10. Lanjutkan ke *operand* “D”, karena sebagai *operand*, maka “D” dijadikan hasil saja.



11. Lanjutkan ke operator “^”, operator ini akan dimasukkan ke dalam *stack*. Karena di *top stack* sudah ada isinya, maka bandingkan keduanya. Karena “^” berderajat lebih besar dari *top stack*nya (“/”) maka masukkan (tumpuk) saja.

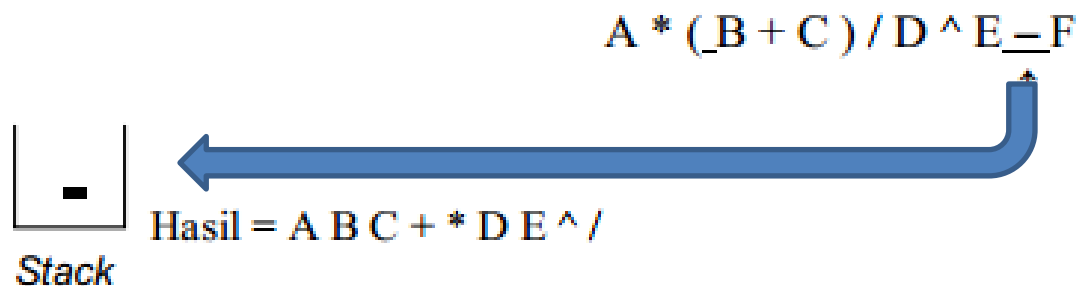


12. Lanjutkan ke *operand* “E”, karena sebagai *operand*, maka “E” dijadikan hasil saja.

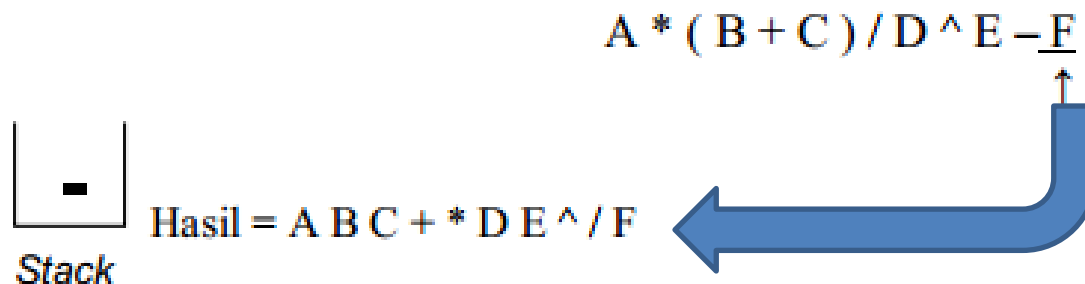


13. Lanjutkan ke operator “-”, operator ini akan dimasukkan ke dalam *stack*. Karena di *top stack* sudah ada isinya, maka bandingkan keduanya. Karena “-” berderajat lebih kecil dari “^” maka operator “^” dikeluarkan dari tumpukan dan dijadikan hasil.

Ketika “-“ akan masuk, di top stack kini ada “/” yang berderajat lebih besar dari “-“, akibatnya top stack (“/”) dikeluarkan juga dan dijadikan hasil. Kini “-“ menjadi top stacknya.



14. Lanjutkan ke *operand* “F”, karena sebagai *operand*, maka “F” dijadikan hasil saja.



15. Karena proses telah selesai, maka keluarkan seluruh isi stack mengikuti kaidahnya, *last in first out*. Karena hanya ada “-“ maka hasil akhirnya menjadi:

$A B C + * D E ^ / F -$



Hasil ini harus sama dengan *postfix* yang menggunakan cara manual. Terlihat langkahnya lebih panjang dari cara manual, namun jika telah terbiasa, cara ini dapat dilakukan dengan lebih mudah dari pada cara manual. Kalau dipersingkat, bentuknya menjadi:

A	*	(B	+	C)	/	D	^	E	-	F	;
	*	((+	+				^	^			
	*	*	*	*	*	*	/	/	/	/	-	-	
A			B		C	+	*	D		E	^ / F	-	



good bye !