3 STACK ATAU TUMPUKAN

3.1 DAFTAR LINEAR

Sebuah daftar *linear* atau *linear list*, merupakan suatu struktur data umum yang terbentuk dari barisan hingga (yang terurut) dari satuan data ataupun dari *record*. Untuk mudahnya, elemen yang terdapat di dalam daftar disebut dengan simpul atau *node*. Daftar disebut *linear* (lurus), karena elemen tampak seperti berbaris, yakni bahwa setiap simpul, kecuali yang pertama dan yang terakhir, selalu memiliki sebuah elemen penerus langsung (suksesor langsung) dan sebuah elemen pendahulu langsung (*predesesor* langsung).

Di sini, banyak simpul atau elemen, tersebut dapat berubah-ubah, berbeda dengan *array* yang banyak elemennya selalu tetap. Kita menyatakan *linear list* A yang mengandung T elemen pada suatu saat, sebagai A = [A1, A2, …AT]. Jika T = 0, maka A disebut *list* hampa atau *null list*.

Suatu elemen dapat dihilangkan atau dihapus (*deletion*) dari sembarang posisi dalam *linear list*, dan suatu elemen baru dapat pula dimasukkan (*insertion*) sebagai anggota *list* pada posisi sembarang (di mana saja).

File, merupakan salah satu contoh dari daftar *linear* yang elemen-elemennya berupa *record*. Selain *file*, contoh lain dari daftar *linear* adalah *stack* atau tumpukan, *queue* atau antrean, dan daftar berkait atau *linear linked list* atau *one-way list*. Pada Bab 3 ini kita bahas tentang *stack* tersebut. Selanjutnya pada Bab 4 kita bahas tentang antrean dan Bab 5 tentang *linked list*.

3.2 *STACK* atau TUMPUKAN

Stack atau tumpukan adalah bentuk khusus dari *linear list*. Pada *stack*, penghapusan serta pemasukan elemennya hanya dapat dilakukan di satu posisi, yakni posisi akhir dari *list*. Posisi ini disebut puncak atau *top* dari *stack*. Elemen *stack* S pada posisi ini dinyatakan dengan TOP(S).

Jelasnya, bila *stack* S [S_1, S_2, …, S_T], maka TOP(S) adalah S_T. Banyaknya elemen *stack* S pada suatu saat tertentu biasa kita sebut sebagai NOEL(S). Jadi untuk *stack* kita di atas, NOEL(S) = T. Seperti halnya pada semua *linear list*, pada *stack* dikenal operasi penghapusan dan pemasukan.

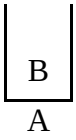
Operator penghapusan elemen pada *stack* disebut POP, sedangkan operator pemasukan elemen, disebut PUSH. Untuk menggambarkan kerja kedua operator di atas, berikut ini suatu contoh bermula dari *stack* hampa S[ ], yang kita gambar sebagai :

S NOEL(S) = 0, TOP(S) tidak terdefinisi

mula-mula kita PUSH elemen A, diperoleh Stack S = [A]

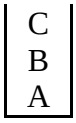
S NOEL(S) = 1, TOP(S) = A

Apabila kemudian kita PUSH elemen B, diperoleh Stack S = [A,B]



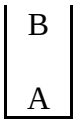
S NOEL(S) = 2, TOP(S) = B

Selanjutnya bila PUSH elemen C, diperoleh Stack S = [A,B,C]



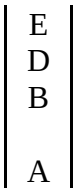
S NOEL(S) = 3, TOP(S) = B

Kemudian bila kita POP elemen C, diperoleh Stack S = [A,B]



S NOEL(S) = 2, TOP(S) = B

Kita dapat pula PUSH 2 elemen D dan E. Akan dihasilkan Stack S = [A,B,D,E]



S NOEL(S) = 4, TOP(S) = E, dan seterusnya.

Terlihat bahwa kedua operasi di atas, pada *stack* adalah bersifat 'terakhir masuk pertama keluar' atau '*last in first out* (LIFO)'. Pada hakekatnya kita tidak membatasi berapa banyak elemen dapat masuk ke dalam *stack*. Untuk suatu *stack* $S[S_1, S_2,..., S_{NOEL}]$, kita katakan bahwa elemen S_i, berada di atas elemen S_j, jika i lebih besar dari j. Suatu elemen tidak dapat kita POP ke luar, sebelum semua elemen di atasnya dikeluarkan.

3.3 OPERASI PADA *STACK*

Terdapat empat operasi pada *stack*, yakni CREATE (*stack*), ISEMPTY(*stack*), PUSH(elemen, *stack*), dan POP (*stack*). CREATE(S) adalah operator yang menyebabkan *stack* S menjadi satu *stack* hampa. Jadi NOEL(CREATE(S)) adalah 0, dan TOP(CREATE(S)) tak terdefinisi.

Sedangkan operator ISEMPTY(S) bermaksud memeriksa apakah *stack* S hampa atau tidak. *Operand*nya adalah data bertipe *stack*, sedangkan hasilnya merupakan data bertipe *boolean*. ISEMPTY(S) adalah *true*, jika S hampa, yakni bila NOEL(S) = 0, dan *false* dalam hal lain. Jelas bahwa ISEMPTY(CREATE(S)) adalah *true*.

Operator PUSH (E,S) akan bekerja menambahkan elemen E pada *stack* S. E ditempatkan sebagai TOP(S). Operator POP(S) merupakan operator yang bekerja mengeluarkan elemen TOP(S) dari dalam *stack*. POP(S) akan mengurangi nilai NOEL(S) dengan 1. Suatu kesalahan akan terjadi apabila, kita mencoba melakukan POP(S) terhadap *stack* S yang hampa.

Kesalahan *overflow* akan terjadi jika kita melakukan operasi pemasukan data (PUSH) pada *stack* yang sudah penuh (dalam hal ini jika banyaknya elemen yang kita masukkan ke dalam sebuah *stack* sudah melampaui batas kemampuan memori atau telah didefinisikan sebelumnya).

Sebaliknya, kesalahan *underflow* akan terjadi jika *stack* sudah dalam keadaan hampa, kita lakukan operasi pengeluaran atau penghapusan (POP).

3.4 DEKLARASI *STACK* DALAM COBOL DAN PASCAL

Meskipun *stack* amat luas digunakan, banyak bahasa pemrograman tidak mempu-nyai tipe data *stack* secara *built-in*. Dalam hal ini, Pemrogram harus memanipulasi sendiri fasilitas yang dimiliki bahasa pemrograman tersebut, untuk dapat melakukan operasi *stack* terhadap variabel *stack*.

Mungkin cara yang paling sederhana adalah membentuk *stack* dalam bentuk se-macam *array*. Jelas kita harus membedakan suatu *stack* dengan suatu *array* yang sesungguhnya. Pemrogram harus memaksakan berlakunya aturan LIFO bagi *stack*. Selain itu juga, penempatan *stack* dalam bentuk *array* mengakibatkan suatu keterbatasan, yakni bahwa elemen *stack* harus homogen. Keterbatasan lain yang timbul adalah keharusan Pemrogram untuk menentukan batas atas dari *subscript array*, walaupun *stack* secara teori tidak memiliki batas maksimum dalam jumlah elemen. Jika diinginkan, seharusnya kita dapat membuat *stack* yang panjangnya tak hingga.

Satu hal yang nyata membedakan *stack* dengan *array* adalah banyaknya elemen *stack* yang dapat bertambah atau berkurang setiap waktu, sementara banyaknya elemen sebuah *array* selalu tetap.

Sekarang marilah kita bicarakan deklarasi dari variabel S yang bertipe data *stack*. Diasumsikan bahwa elemen dari S masing-masing bertipe data integer dan panjang *stack* maksimum adalah 100 elemen. Kita mendeklarasikan sebuah *array* yang dilengkapi dengan variabel TOP-PTR.

Variabel TOP-PTR ini menyatakan *subscript* dari elemen TOP(S) dari *stack*. Kita menamakan kombinasi dari *array* dan indikator untuk TOP tersebut dengan nama STACK-STRUC. Dengan penyajian seperti ini, berlaku bahwa NOEL(S) = TOP-PTR, ISEMPTY(S) adalah *true* bila TOP-PTR = 0, dan *false* bila TOP-PTR lebih besar dari 0.

Dalam COBOL

```
01 STACK-STRUCT.
      02 S PIC 9(5)
            OCCURS 100 TIMES.
      02 TOP-PTR PIC 9(3).
```

Dalam Pascal

```
type stackstruct;
    record stack: array[1..100[of integer;
            top-ptr : integer
      end
var S : stackstruct;
```

Kompilator tidak dapat mengerti aturan LIFO yang kita inginkan. Untuk itu Pemrogram harus berhati-ati dan tidak memberi indeks pada S di sembarang tempat, selain dengan nilai TOP-PTR.

Operasi PUSH dan POP dapat kita program sebagai berikut : kita gunakan EON untuk menyatakan elemen yang di-PUSH ke dalam S dan EOFF untuk elemen yang di-POP ke luar S. NOEL-MAX menyatakan panjang maksimum *stack*. Jadi di sini NOEL-MAX = 100.

Dalam paragraf COBOL :

```
PUSH.
   IF TOP-PTR < NOEL-MAX.
        THEN COMPUTE TOP-PTR = TOP-PTR+1
          MOVE EON TO S(TOP-PTR)
    ELSE overflow condition
    POP
       IF TOP-PTR > 0
          THEN MOVE S(TOP-PTR) TO EOFF
                COMPUTE TOP-PTR = TOP-PTR-1
          ELSE underflow condition
```

Dalam *procedure* Pascal :

```
procedure PUSH (eon : integer);
begin
   if (s.topptr < noelmax)
   then
      begin s.topptr <= s.topptr + 1;
            s.stack [s.topptr] := eon
      end
   else OVERFLOW-CONDITION
end;
procedure POP (var eoff : integer);
begin
   if (s.topptr > 0)
   then
      begin eoff := s.Stack(s.topptr);
                    s.topptr := s.topptr-1
      end
   else UNDERFLOW-CONDITION
end;
```

3.5 APLIKASI *STACK*

Stack sangat luas pemakaiannya dalam menyelesaikan berbagai macam problema. Kompilator, sistem operasi, dan berbagai program aplikasi banyak menggunakan konsep *stack* tersebut. Salah satu contoh adalah problema Penjodohan Tanda Kurung atau *matching parantheses.*

Sebuah kompilator mempunyai tugas, salah satu di antaranya adalah menyelidiki apakah Pemrogram telah dengan cermat mengikuti aturan tata bahasa, atau sintaks dari bahasa pemrograman yang bersangkutan. Misalnya untuk *parantheses* kiri (tanda kurung buka) yang diberikan, harus dipastikan adanya *parantheses* kanan (tanda kurung tutup) yang bersangkutan.

Stack dapat digunakan dalam prosedur *matching* yang digunakan. Algoritmanya sederhana, kita amati barisan elemen dari kiri ke kanan. Bila kita bertemu dengan suatu *parantheses* kiri, maka *parantheses* kiri tersebut kita PUSH ke dalam sebuah *stack*. Selanjutnya bila kita bertemu dengan suatu *parantheses* kanan, kita periksa *stack*, apakah hampa atau tidak. Kalau *stack* hampa, berarti terdapat *parantheses* kanan tanpa adanya *parantheses* kiri. Suatu kesalahan, atau *error*, apabila *stack* tidak hampa, berarti tidak diperoleh sepasang *parantheses* kiri, dan kanan, kita POP elemen ke luar *stack*.

Jika sampai berakhirnya barisan elemen, *stack* tidak hampa berarti terdapat *parantheses* kiri yang tidak tertutup dengan *parantheses* kanan. Lagi suatu kesalahan. Kita akan membuat programnya dalam COBOL. Barisan elemen yang diamati kita tampung karakter demi karakter dalam variabel *array* bernama STRING. *Stack* ditempatkan dalam *array* STACK. Kita asumsikan bahwa jumlah maksimum karakter dalam barisan elemen adalah 80 dan barisan berakhir dengan karakter titik-koma..

Struktur datanya didefinisikan sebagai berikut :

```
01 STACK-STRUCT.
      02 STACK                PIC X
                              OCCURS 80 TIMES.
      02 TOP-PTR              PIC 99 VALUE 0.
01 STRING
      02 CHAR                 PIC X
                              OCCURS 80 TIMES
01 NEXT-CHAR                  PIC 99
```

Struktur di atas kita manipulasi dengan prosedur sebagai berikut :

```
PERFORM SCAN-NEXT-CHAR
      VARIYING NECT CHAR FROM 1 BY 1
      UNTIL  NEXT-CHAR > 80
            OR CHAR(NEXT-CHAR) = ";".
      IF TOP-PTR NOT = 0 THEN invalid syntax.
                  parenthesis kiri tak tertutup
      ELSE valid syntax
      SCAN NEXT-CHAR
      IF CHAR(NEXT-CHAR) = "("
            PERFORM PUSH
      ELSE
      IF CHAR(NEXT-CHAR) = ")"
            PERFORM  POP

      PUSH
            COMPUTE TOP-PTR = TOP-PTR + 1
            MOVE CHAR (NEXT-CHAR) TO STACK (TOP-PTR).
            IF TOP-PTR > 0
                  COMPUTE TOP-PTR – 1
            ELSE invalid syntax, tak ada parenthesis
```

Silakan Anda buat programnya dalam bahasa pemrograman yang Anda kuasai.

NOTASI POSTFIX

Aplikasi lain dari *stack* adalah pada kompilasi dari ekspresi dalam bahasa pemrograman tingkat tinggi. Kompilator harus mampu menyerahkan bentuk yang biasa, misalnya ((A+B)*C/D+E^F)/G ke suatu bentuk yang dapat lebih mudah dipergunakan dalam pembentukan kode objeknya.

Cara yang biasa kita lakukan dalam menulis ekspresi aritmetik seperti di atas, dikenal sebagai notasi *infix*. Untuk operasi binar seperti menjumlah, membagi, mengurangi, mengalikan ataupun memangkatkan, operator tampil di antara dua *operand*, misalnya operator + tampil di antara *operand* A dan B pada operasi A + B.

Stack dapat digunakan untuk mentransformasikan notasi *infix* ini menjadi notasi *posfix*. Pada notasi *posfix*, kedua *operand* tampil bersama di depan operator, misalnya AB+ atau PQ* dan sebagainya. Kompilator akan lebih mudah menangani ekspresi dalam notasi *posfix* ini.

Berikut contoh melakukan pengalihan ekspresi *infix* ke *postfix* secara manual. Ekspresi *infix* = A + B / C * D akan dialihkan menjadi ekspresi *postfix*.

1. Pilih sub-ekspresi yang berisi "dua *operand* dan satu *operator*" yang memiliki *level* tertinggi di ekspresi di atas. Didapat B / C dan C * D. Pilih yang paling kiri, maka kita peroleh : B / C.
2. Ubah sub-ekspresi tersebut menjadi sebuah *operand*, misalkan B / C menjadi E, maka ekspresi semula menjadi : A + E * D.
3. Lakukan langkah ke (2) hingga ekspresi di atas menjadi "dua *operand* dan satu *operator*" saja. Didapat : A + F
4. Alihkan menjadi bentuk *postfix* : *operand-operand-operator*, diperoleh A F +
5. Kembalikan setiap *operand* menjadi ekspresi semula. F tadinya adalah E * D, maka nilai F = E * D. Satukan dengan ekspresi yang telah menjadi *postfix*. Hasilnya = A * E + D
6. Ulangi langkah ke (5) hingga terbentuk ekspresi postfix. Didapat A * B + C / D

Dengan demikian, ekspresi *infix* : A+B/C*D akan menjadi ABC/D*+ dalam notasi *postfix*. Perhatikan dan pelajari tabel berikut ini :

Tabel 3.1 *Postfix-Infix*

Ekspresi *Infix*	Ekspresi *Postfix*
A + B	A + B
A + B * C	A + B * C
(A + B) * C	A + B * C
A * B + C	A + B * C

Bila ada sub-ekspresi di dalam tanda kurung, maka sub-ekspresi tersebut harus dikerjakan terlebih dulu.

Berikut ini diberikan sebuah algoritma untuk mengubah notasi *infix* ke dalam notasi *posfix*. Sebuah *stack* digunakan untuk keperluan ini. Ekspresi diamati satu persatu dari kiri ke kanan. Pada algoritma ini terdapat 4 aturan dasar, sebagai berikut :

1. Jika simbol adalah "(" (kurung buka), maka ia kita PUSH ke dalam *stack*
2. Jika simbol adalah ")" (kurung tutup), POP dari *stack* elemen-elemen *stack*, sampai pertama kali kita POP simbol "(". Semua elemen *stack* yang di POP tersebut merupakan *output*, kecuali "(" tadi.
3. Jika simbol adalah sebuah *operand*, tanpa melakukan perubahan elemen *stack*, *operand* tersebut langsung mcrupakan *output*.

4. Jika simbol adalah sebuah operator, maka jika TOP *stack* adalah operator dengan *level* lebih tinggi atau sama, maka elemen TOP kita POP, sekaligus keluar sebagai *output*, dilanjutkan proses seperti ini sampai TOP merupakan "(" atau operator dengan *level* lebih rendah. Kalau hal ini terjadi, operator (yang diamati) kita PUSH ke dalam *stack*. Biasanya ditambahkan simbol ; (titik-koma) sebagai penutup ekspresi. Dalam keadaan ini, kita POP semua elemen *stack*, sehingga *stack* menjadi hampa.

Dapat dicatat bahwa terdapat 3 *level* operator, yakni pemangkatan (*level* tertinggi), *level* menengahnya adalah perkalian (*) dan pembagian (/) dan *level* terendah adalah penjumlahan (+) dan pengurangan (-). Tabel berikut menunjukkan pelaksanaan algoritma di atas untuk mengubah ekspresi ((A+B)*C/D+E^F)/G; ke dalam notasi *postfix*.

Tabel 3.2

Simbol yang diamati	(	(	A	+	B	)	*	C	/	D	+	E	^	F	)	/	G	;
TOP(S)	(	(	(	+	+	(	*	*	/	/	+	+	^	^		/		
		(	(	(	(		(	(	(	(	(	(	+	+				
				(	(								(	(				
Output			A		B	+		C	*	D	/	E		F	^+		G	/

Jadi, ((A+B)*C/D+E^F)/G akan menjadi AB+C*D/EF^+G/ dalam notasi *postfix*.

Bukti lain :

((A + B) * C / D + E ^ F) / G
(H * C / D + E ^ F) / G
(H * C / D + I) / G
(J / D + I) / G
(K + I) / G
L / G

Infix : L / G menjadi L G / dalam *postfix.*

L adalah K + I yang *postfix*nya : K + I
Bila disatukan dengan hasil sebelumnya menjadi : K + I /G
K adalah J / D yang *postfix*nya : J / D
Bila disatukan dengan hasil sebelumnya menjadi : J/D+I/G
J adalah H * C yang *postfix*nya : H * C
Bila disatukan dengan hasil sebelumnya menjadi : H*C/D+I/G
I adalah E ^ F yang *postfix*nya : E ^ F

Bila disatukan dengan hasil sebelumnya menjadi : H*C/D^E+F/G
H adalah A + B yang *postfix*nya : A + B
Bila disatukan dengan hasil sebelumnya menjadi : A+B*C/D^E+F/G

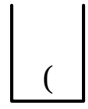
Perhatikan, bahwa dalam notasi *postfix*, tidak ada tanda kurung buka maupun kurung tutup. Silakan Anda buat program (dengan bahasa pemrograman apapun yang Anda kuasai) untuk menyelesaikan soal seperti di atas.

Untuk lebih jelasnya, mari kita ulangi langkah-demi langkah. Misalkan untuk notasi *infix* : (A * (B + C) ^ D – E) / F + G, bagaimana bentuk *postfix*nya ?

Langkah pertama : kita mulai dari karakter pertama mulai dari kiri :

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

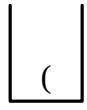
Kita dapat operator '(' dan tentu saja *stack* masih dalam keadaan hampa sehingga TOP(S) belum terdefinisi. Masukkan saja tanda '(' ke dalam *stack* dan menjadi TOP(S)-nya.



TOP(S)

Selanjutnya, kita bergeser ke kanan. Karakter berikutnya adalah *operand* 'A'. *Operand* (langsung) dijadikan hasil saja.

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

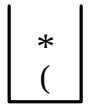


TOP(S) Hasil : A

Lanjutkan kembali dengan karakter berikutnya, yakni operator '*'. Bandingkan operator tersebut dengan TOP(S)-nya.

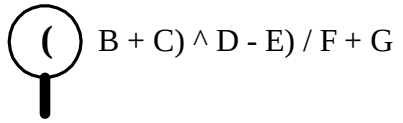
(B + C) ^ D - E) / F + G

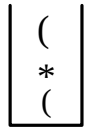
Karena TOP(S) adalah operator '(', maka masukkan operator '*' tersebut, sehingga TOP(S) sekarang adalah operator '*'



Hasil : A

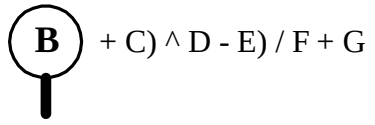
Selanjutnya, kita dapatkan operator '(' yang akan kembali kita masukkan ke dalam *stack*. Tanpa perlu ragu-ragu lagi, operator '(' dimasukkan ke dalam *stack* dan akan menjadi TOP(S) yang baru.

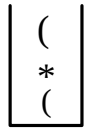




Hasil : A

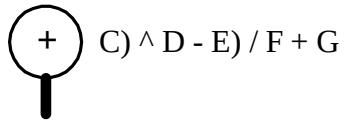
Kita lanjutkan lagi untuk memeriksa karakter berikutnya. Kita dapatkan *operand* 'B'. Seperti biasa, *operand* (langsung) dijadikan hasil saja.

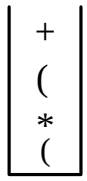




Hasil : A B

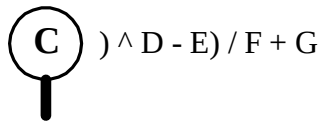
Selanjutnya, bergeser lagi ke kanan, kita dapatkan operator '+' yang akan kembali kita masukkan ke dalam *stack*. Tanpa perlu ragu-ragu lagi, jika TOP(S) adalah '(', maka operator apapun dimasukkan saja ke dalam *stack* dan akan menjadi TOP(S) yang baru.

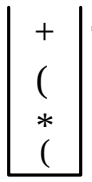




Hasil : A B

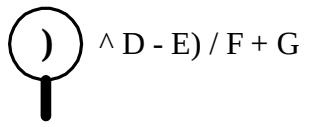
Kita lanjutkan lagi untuk memeriksa karakter berikutnya. Kita dapatkan *operand* 'C.' Seperti biasa, *operand* (langsung) dijadikan hasil saja.



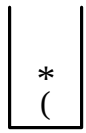


Hasil : A B C

Bergeser ke kanan lagi, kita peroleh operator ')' yang akan dibandingkan dengan TOP(S)-nya.

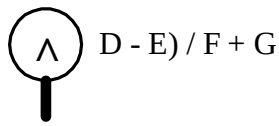


Jika yang akan kita (coba) masukkan adalah operator ')', maka keluarkan isi *stack* satu per satu (menjadi hasil) hingga ketemu operator '(' yang terdekat. Tetapi, baik operator '(' maupun operator ')'-nya dibuang saja (tidak dijadikan hasil). Hasilnya :

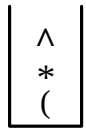


Hasil : A B C +

Berikutnya, kembali kita bergeser ke kanan satu langkah, kita peroleh operator '^'.

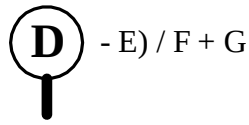


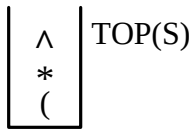
Kita bandingkan operator '^' dengan TOP(S)-nya yakni '*'. Karena derajat operasi '^' lebih tinggi dari '*', maka masukkan saja.



Hasil : A B C +

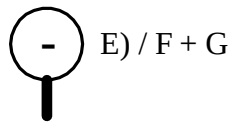
Kita lanjutkan lagi, dan kembali kita dapatkan *operand* 'D'. Seperti biasa, langsung jadikan hasil saja.



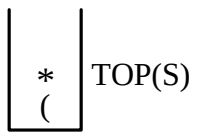


Hasil : A B C + D

Geser lagi ke kanan. Kini kita dapatkan operator '-'. Bandingkan operator tersebut dengan posisi TOP(S).

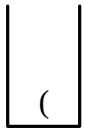


Karena operator '-' berderajat operasi lebih rendah dari operator '^', maka keluarkan operator '^' dari dalam *stack* untuk dijadikan hasil.



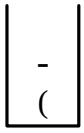
Hasil : A B C + D ^

Kini TOP(S) adalah operator '*'. Karena operator '*' yang merupakan TOP(S) tersebut masih berderajat operasi lebih tinggi dari operator '-' yang akan kita masukkan, maka operator '*' tersebut dikeluarkan dari *stack* dan dijadikan hasil. Proses pengulangan seperti ini (selama TOP(S) berderajat operasi lebih tinggi dari operator yang akan dimasukkan ke dalam *stack*) disebut dengan rekursif.



Hasil : A B C + D ^ *

Perhatikan, kini TOP(S) berisi operator '('. Operator tersebut hanya akan bereaksi dengan operator ')'. Dengan demikian, operator '-' kini kita masukkan ke dalam *stack* dan menempati TOP(S).

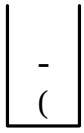


Hasil : A B C + D ^ *

Langkah berikutnya, kembali kita periksa satu karakter di kanan '-' tadi, kita dapatkan *operand* 'E'.

) / F + G

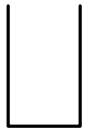
Seperti biasa, *operand* (langsung) kita jadikan hasil :



Hasil : A B C + D ^ * E

Di kanan *operand* 'E' kita peroleh operator ')'. Ingat, operator ')' akan mengeluarkan semua isi *stack* yang dimulai dari posisi TOP(S) hingga ke operator '(', tetapi operator '(' dan ')' dibuang saja, tidak dimasukkan menjadi hasil.

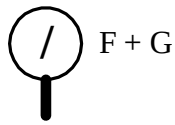
/ F + G



Hasil : A B C + D ^ * E -

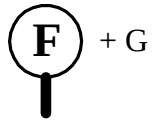
Tampak bahwa *stack* dalam keadaan hampa, ISEMPTY(S) = *true*, NOEL(S) = 0, TOP(S) tidak terdefinisi.

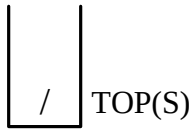
Kita lanjutkan dengan operator '/'. Karena *stack* dalam keadaan hampa, maka operator '/' dimasukkan saja ke dalam *stack.*



Hasil : A B C + D ^ * E -

Kini *stack* berisi kembali, ISEMPTY(S) = *false*, NOEL(S) = 1, TOP(S) = '/' Kembali kita lanjutkan, kita dapatkan operator 'F' yang tentu saja langsung dijadikan hasil.

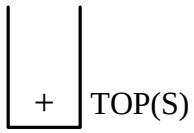




Hasil : A B C + D ^ * E - F

Berlanjut lagi, kini kita dapatkan operator '+' yang akan kita bandingkan dengan TOP(S)-nya yaitu '/'.

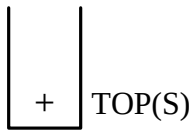
Karena operator '+' berderajat operasi lebih rendah dari '/', maka TOP(S) dikeluarkan sebagai hasil dan posisinya digantikan oleh operator '+'.



Hasil : A B C + D ^ * E - F /

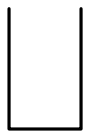
Pemeriksaan sampai pada elemen terakhir dari notasi *infix* yaitu *operand* 'G'.

Seperti biasa, *operand* langsung dijadikan hasil saja.



Hasil : A B C + D ^ * E - F / G

Nah, karena tidak ada lagi elemen dari notasi yang akan diperiksa, maka akhiri pengerjaan dengan mengeluarkan semua isi *stack* yang dimulai dari posisi TOP(S)-nya. Ingat, prinsip kerja *stack* adalah *last in first out*. Karena kebetulan cuma ada satu elemen di dalam *stack* yang juga merupakan TOP(S)-nya, maka hanya satu itu yang dikeluarkan. Hasil akhirnya menjadi :



Hasil : A B C + D ^ * E - F / G +

Dengan demikian, untuk notasi *infix* : (A * (B + C) ^ D – E) / F + G akan menjadi A +B ^C * D – E / F + G dalam notasi *postfix*-nya. Periksalah !

L A T I H A N 3

1. *Stack* termasuk (a) *linear list* atau (b) *non linear list* ?

2. Jelaskan perbedaan antara *linear list* dan *non linear list*, beri contoh.

3. Penghapusan elemen dari *stack* dilakukan mulai dari bagian (a) awal/*bottom* atau (b) akhir/ *top* ?

4. Apa fungsi dari perintah NOEL(S) ?

5. Perintah (operator) apa yang digunakan untuk memasukkan elemen ke dalam *stack* ?

6. Perintah (operator) apa yang digunakan untuk menghapus elemen dari dalam *stack* ?

7. Apa saja perlakuan yang dimungkinkan sehingga suatu *stack* dikatakan dalam kondisi hampa ?

8. Prinsip kerja *stack* adalah (a) FIFO atau (b) LIFO ?

9. Sebutkan 4 (empat) operasi pada *stack* dan beri contoh-contohnya.

10. Dari keempat operasi di atas, mana yang menghasilkan data yang bertipe *boolean* ?

11. Pada sebuah *stack*, kapan posisi TOP = BOTTOM ?

12. Sebutkan dua kesalahan yang mungkin terjadi pada pengoperasian *stack* dan pada kondisi seperti apa kesalahan itu bisa terjadi ?

13. Apa saja keterbatasan mendeklarasikan tipe data yang seharusnya *stack*, tetapi harus menggunakan tipe data *array* ? (di bahasa pemrograman itu tidak ada *stack*).

14. Apa yang dimaksud dengan istilah *matching parantheses ?*

15. Operasi pada *stack* dapat mengalihkan notasi *infix* menjadi (a) *prefix* atau (b) *postfix*?

16. Alihkan notasi *infix* A-B*C^D^(E*F/(G-H)) ke notasi *postfix*

17. Buat program dengan bahasa pemrograman apapun untuk menyelesaikan soal nomor 16 di atas.