Stack (Tumpukan)

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Stack

Stack adalah sekumpulan elemen yang:

- dikenali elemen puncaknya (**Top**),
- aturan penambahan dan penghapusan elemennya tertentu:
  - Penambahan selalu dilakukan "di atas" **Top**,
  - Penghapusan selalu dilakukan pada **Top**.

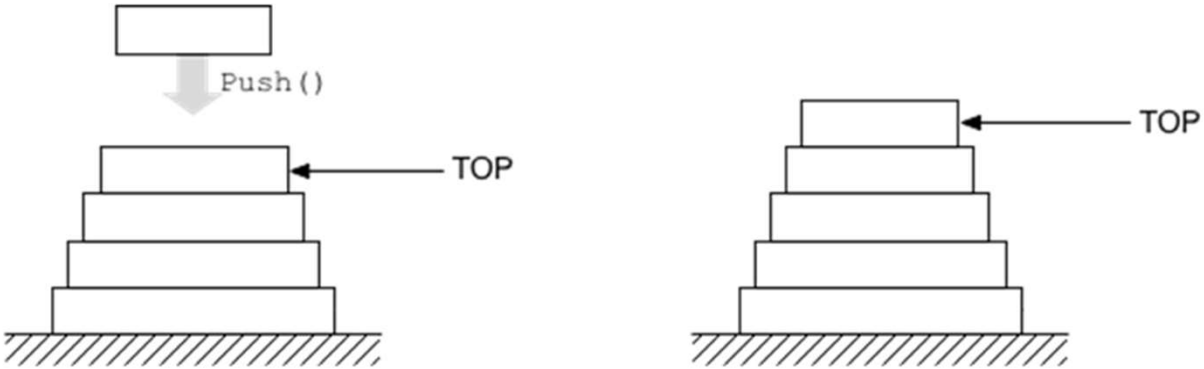
Top adalah satu-satunya alamat tempat terjadi operasi.

Elemen Stack tersusun secara **LIFO** (*Last In First Out*).

Designed by lifeforstock / Freepik

Lakeside 8206 Two Stack Plate Dispenser

Stack



Stack seperti sebuah List dengan batasan lokasi penambahan & pengurangan elemen.

- Pada **Stack**: operasi penambahan dan pengurangan hanya dilakukan di **salah satu** “ujung” list.
- Pada **List**: operasi **boleh di manapun**.

Tower of Hanoi

https://en.wikipedia.org/wiki/File:Tower_of_Hanoi_4.gif

Pemakaian Stack

- Pemanggilan prosedur/fungsi
- Perhitungan ekspresi aritmatika
- Rekursivitas
- *Backtracking*

dan algoritma lanjut yang lain

Definisi operasi

Jika diberikan *S* adalah Stack dengan elemen *ElmtS*

CreateStack: $\rightarrow S$	{ Membuat sebuah tumpukan kosong }
top: $S \rightarrow ElmtS$	{ Mengirimkan elemen teratas S saat ini }
length: $S \rightarrow$ *integer*	{ Mengirimkan banyaknya elemen S saat ini }
push: $ElmtS \times S \rightarrow S$	{ Menambahkan sebuah elemen ElmtS sebagai TOP, TOP berubah nilainya }
pop: $S \rightarrow S \times ElmtS$	{ Mengambil nilai elemen TOP, sehingga TOP yang baru adalah elemen yang datang sebelum elemen TOP, mungkin S menjadi kosong }
isEmpty: $S \rightarrow$ *boolean*	{ Test stack kosong, true jika S kosong, false jika S tidak kosong }

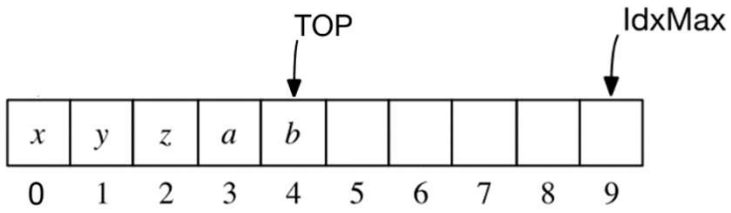
Axiomatic Semantics (fungsional)

1) new() returns a stack
2) pop(push(*v*, *S*)) = *S*
3) top(push(*v*, *S*)) = *v*

Di mana *S* adalah Stack dan *v* adalah value.

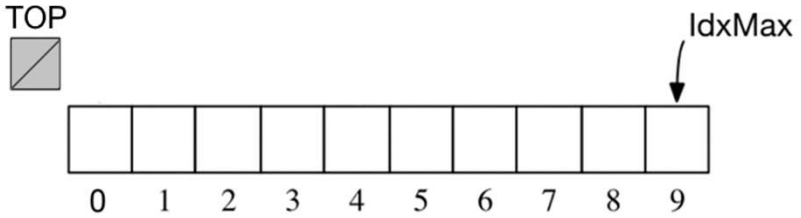
Implementasi Stack dengan List (array)

- Ilustrasi Stack tidak kosong, dengan 5 elemen:



*dengan IdxMax = CAPACITY-1

- Ilustrasi Stack kosong, maka Top diset = IDX_UNDEF.



ADT Stack (dengan array eksplisit-statik)

```
KAMUS UMUM
constant IDX_UNDEF: integer = -1
constant CAPACITY: integer = 10

type ElType: integer { elemen Stack }

{ Stack dengan array statik }
type Stack: < buffer: array [0..CAPACITY-1] of ElType, { penyimpanan elemen }
              idxTop: integer > { indeks elemen teratas }
```

ADT Stack – Konstruktor, akses, & predikat

```
procedure CreateStack(output s: Stack)
{ I.S. Sembarang
  F.S. Membuat sebuah Stack s yang kosong berkapasitas CAPACITY
       jadi indeksnya antara 0..CAPACITY-1
       Ciri Queue kosong: idxTop bernilai IDX_UNDEF }

function top(s: Stack) → ElType
{ Prekondisi: s tidak kosong.
  Mengirim elemen terdepan s, yaitu s.buffer[q.idxTop]. }
function length(s: Stack) → integer
{ Mengirim jumlah elemen s saat ini }

function isEmpty(s: Stack) → boolean
{ Mengirim true jika s kosong: lihat definisi di atas }
function isFull(s: Stack) → boolean
{ Mengirim true jika penyimpanan s penuh }
```

ADT Stack – Operasi

```
{ *** Menambahkan sebuah elemen ke Stack *** }
procedure push(input/output s: Stack, input val: ElType)
{ Menambahkan val sebagai elemen Stack s.
  I.S. s mungkin kosong, tidak penuh
  F.S. val menjadi TOP yang baru, TOP bertambah 1 }

{ *** Menghapus sebuah elemen Stack *** }
procedure pop(input/output s: Stack, output val: ElType)
{ Menghapus X dari Stack S.
  I.S. S tidak mungkin kosong
  F.S. X adalah nilai elemen TOP yang lama, TOP berkurang 1 }
```

ADT Stack (dengan array eksplisit-statik)

```
procedure CreateStack(output s: Stack)
{ I.S. Sembarang
  F.S. Membuat sebuah Stack s yang kosong berkapasitas CAPACITY
       jadi indeksnya antara 0..CAPACITY-1
       Ciri stack kosong: idxTop bernilai IDX_UNDEF }
KAMUS LOKAL
    -
ALGORITMA
    s.idxTop ← IDX_UNDEF
```

ADT Stack (dengan array eksplisit-statik)

```
function isEmpty(s: Stack) → boolean
{ Mengirim true jika Stack kosong: lihat definisi di atas }
KAMUS LOKAL
    -
ALGORITMA
    → s.idxTop = IDX_UNDEF


function isFull(s: Stack) → boolean
{ Mengirim true jika penyimpanan s penuh }
KAMUS LOKAL
    -
ALGORITMA
    → s.idxTop = CAPACITY-1
```

ADT Stack (dengan array eksplisit-statik)

```
procedure push(input/output s: Stack, input val: ElType)
{ (keterangan tidak ditulis untuk menghemat tempat) }
KAMUS LOKAL
    -
ALGORITMA
    s.idxTop ← s.idxTop + 1
    s.buffer[s.idxTop] ← val

procedure pop(input/output s: Stack, output val: ElType)
{ (keterangan tidak ditulis untuk menghemat tempat) }
KAMUS LOKAL
    -
ALGORITMA
    val ← top(s)
    s.idxTop ← s.idxTop - 1
```

Thought exercise

Sama seperti pada Queue, contoh-contoh sebelumnya menggunakan buffer yang terbatas dan statis.

Renungkan perubahan seperti pada kasus Queue:

1) buffer menjadi dinamis (*s1*, *s2*: Stack;
   s1 dan *s2* bisa memiliki kapasitas yang berbeda)
2) stack tidak boleh memiliki batas length
   (length bisa ∞ secara teoretis)

Apa konsekuensinya terhadap implementasi?
Apa bedanya (jika ada) dengan konsekuensi di ADT Queue?

ADT Stack dalam Bahasa C

ADT Stack dalam Bahasa C (dengan array eksplisit-statik)

```c
#ifndef STACK_H
#define STACK_H

#include "boolean.h"

#define IDX_UNDEF -1
#define CAPACITY 10

typedef int ElType;
typedef struct {
    ElType buffer[CAPACITY];
    int idxTop;
} Stack;

#define IDX_TOP(s) (s).idxTop
#define     TOP(s) (s).buffer[(s).idxTop]
```

ADT Stack dalam Bahasa C (dengan array eksplisit-statik)

```
/*** Konstruktor/Kreator ***/
void CreateStack(Stack *s);
/* I.S. Sembarang */
/* F.S. Membuat sebuah Stack s yang kosong berkapasitas CAPACITY */
/* jadi indeksnya antara 0..CAPACITY-1 */
/* Ciri stack kosong: idxTop bernilai IDX_UNDEF */

/********** Predikat Untuk test keadaan KOLEKSI **********/
boolean isEmpty(Stack s);
/* Mengirim true jika Stack kosong: lihat definisi di atas */

boolean isFull(Stack s);
/* Mengirim true jika Stack penuh */

int length(Stack s);
/* Mengirim ukuran Stack s saat ini */
```

ADT Stack dalam Bahasa C (dengan array eksplisit-statik)

```
/*********** Menambahkan sebuah elemen ke Stack **********/
void push(Stack *s, ElType val);
/* Menambahkan val sebagai elemen Stack s.
   I.S. s mungkin kosong, tidak penuh
   F.S. val menjadi TOP yang baru, TOP bertambah 1 */

/*********** Menghapus sebuah elemen Stack **********/
void pop(Stack *s, ElType *val);
/* Menghapus X dari Stack S.
   I.S. S tidak mungkin kosong
   F.S. X adalah nilai elemen TOP yang lama, TOP berkurang 1 */

#endif
```

ADT Stack dalam Bahasa C (dengan array eksplisit-statik)

```
void CreateStack(Stack *s) {
/* ... */
  /* KAMUS LOKAL */
  /* ALGORITMA */
    IDX_TOP(*s) = IDX_UNDEF;
}

int length(Stack s) {
/* ... */
  /* KAMUS LOKAL */
  /* ALGORITMA */
    return (IDX_TOP(s) + 1);
}
```

```
boolean isEmpty(Stack s) {
/* ... */
  /* KAMUS LOKAL */
  /* ALGORITMA */
    return (IDX_TOP(s) == IDX_UNDEF);
}

boolean isFull(Stack s) {
/* ... */
  /* KAMUS LOKAL */
  /* ALGORITMA */
    return (IDX_TOP(s) == CAPACITY-1);
}
```

ADT Stack dalam Bahasa C (dengan array eksplisit-statik)

```c
void push(Stack *s, ElType val) {
/* ... */
  /* KAMUS LOKAL */
  /* ALGORITMA */
    IDX_TOP(*s)++;
    TOP(*s) = val;
}

void pop(Stack *s, ElType *val) {
/* ... */
  /* KAMUS LOKAL */
  /* ALGORITMA */
    *val = TOP(*s);
    IDX_TOP(*s)--;
}
```

Contoh Aplikasi ADT Stack

Evaluasi Ekspresi Aritmatika

Evaluasi ekspresi aritmatika yang ditulis dengan *Reverse Polish Notation* (postfix)

Diberikan sebuah ekspresi aritmatika postfix dengan operator ['*', '/', '+', '–', '^']

Operator mempunyai prioritas (prioritas makin besar, artinya makin tinggi)

Operator	Arti	Prioritas
^	pangkat	3
* /	kali, bagi	2
+ –	tambah, kurang	1

Evaluasi Ekspresi Aritmatika

Contoh:

Ekspresi postfix	Arti (ekspresi infix)
A B * C /	(A*B)/C
A B C ^ / D E * + A C * –	(A/(B^C))+(D*E)–(A*C)

Digunakan istilah token yaitu satuan “kata” yang mewakili sebuah operan (konstanta atau nama) atau sebuah operator.

Mesin Evaluasi Ekspresi

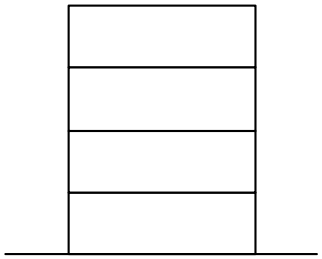
```
Program EKSPRESI
{ Menghitung sebuah ekspresi aritmatika yang ditulis secara postfix }

USE STACK  { paket stack sudah terdefinisi dgn elemennya bertipe token }

KAMUS
   type Token: ...                { terdefinisi }
   s: Stack                       { stack of token }
   currentToken, op1, op2: Token  { token: operan ∪ operator }

   procedure firstToken
   { Mengambil token yang pertama, disimpan di currentToken }
   procedure nextToken
   { Mengambil token yang berikutnya, disimpan di currentToken }
   function endToken → boolean
   { Menghasilkan true jika proses akuisisi mendapat hasil sebuah token kosong.
     Merupakan akhir ekspresi. }
   function isOperator → boolean
   { Menghasilkan true jika currentToken adalah operator }
   function evaluate(op1,op2,operator: token) → token
   { Menghitung ekspresi, mengkonversi hasil menjadi token}
```

(lanjutan)

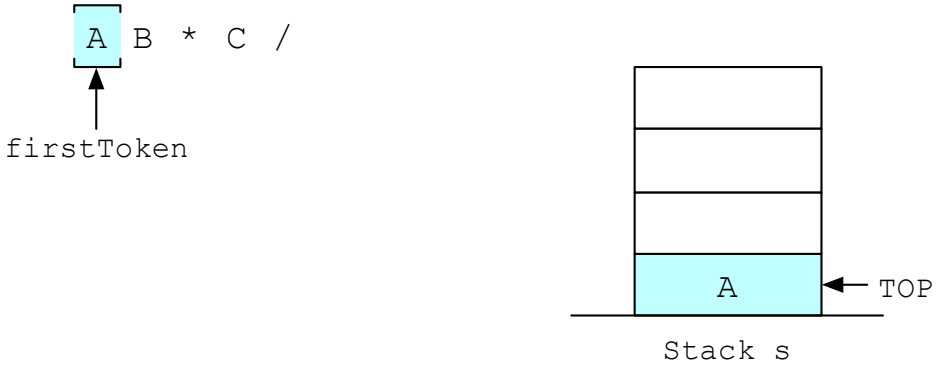
```
ALGORITMA
    firstToken
    if (endToken) then
        output ("Ekspresi kosong")
    else
        repeat
            if not isOperator then
                push(s,currentToken)
            else
                pop(s,op2)
                pop(s,op1)
                push(s,evaluate(op1,op2, currentToken))
            nextToken
        until (endToken)
        output (top(s)) { Menuliskan hasil }
```

Evaluasi Ekspresi Aritmatika

A B * C /



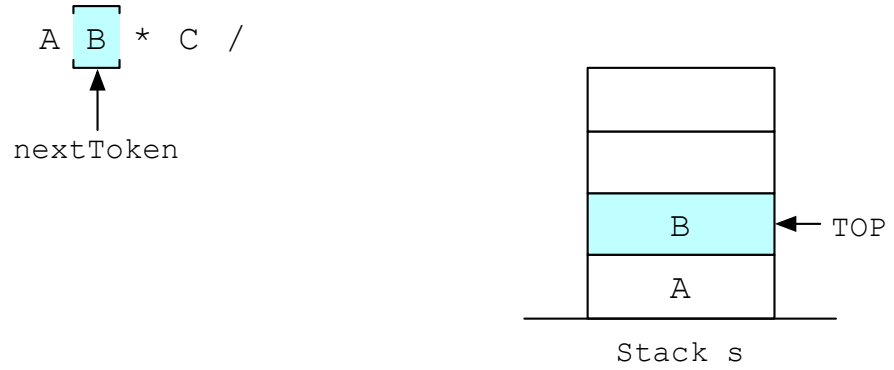
Evaluasi Ekspresi Aritmatika




```
endToken = false
currentToken = 'A'
isOperator(currentToken) = false
```

∴ push(s,currentToken)

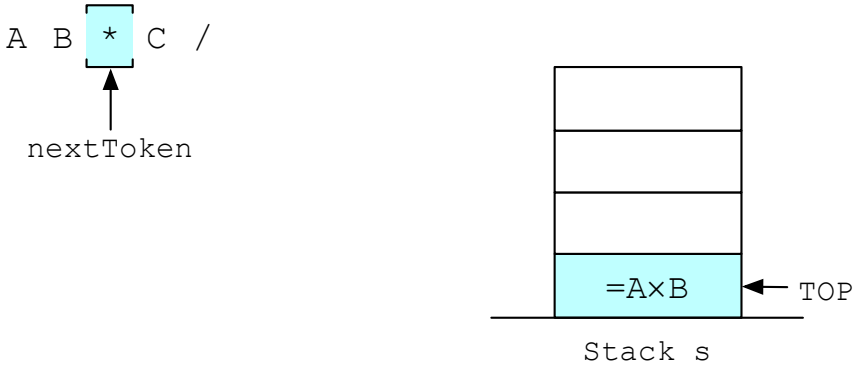
Evaluasi Ekspresi Aritmatika




```
endToken = false
currentToken = 'B'
isOperator(currentToken) = false
```

∴ `push(s,currentToken)`

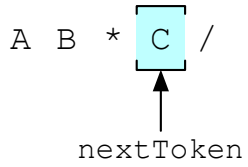
Evaluasi Ekspresi Aritmatika

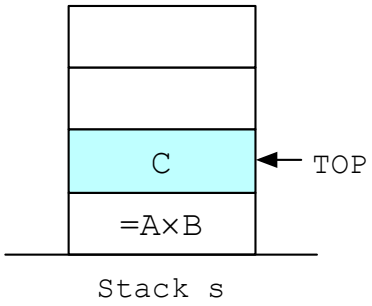



```
endToken = false
currentToken = '*'
isOperator(currentToken) = true
```

∴ `pop(s,op2)`
`pop(s,op1)`
`push(s,evaluate(op1,op2,currentToken))`

Evaluasi Ekspresi Aritmatika

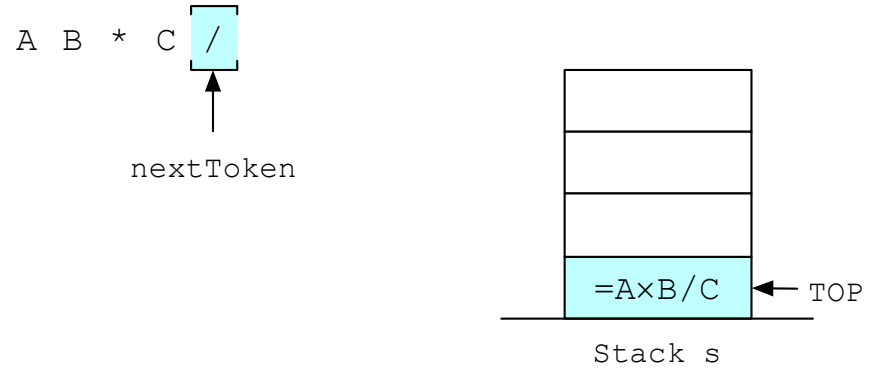





```
endToken = false
currentToken = 'C'
isOperator(currentToken) = false
```

∴ `push(s,currentToken)`

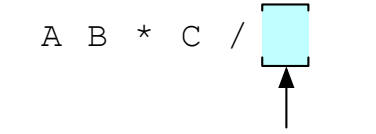
Evaluasi Ekspresi Aritmatika

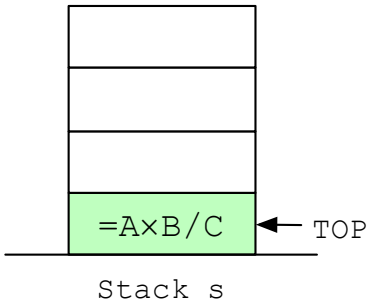



```
endToken = false
currentToken = '/'
isOperator(currentToken) = true
```

∴ `pop(s,op2)`
`pop(s,op1)`
`push(s,evaluate(op1,op2,currentToken))`

Evaluasi Ekspresi Aritmatika

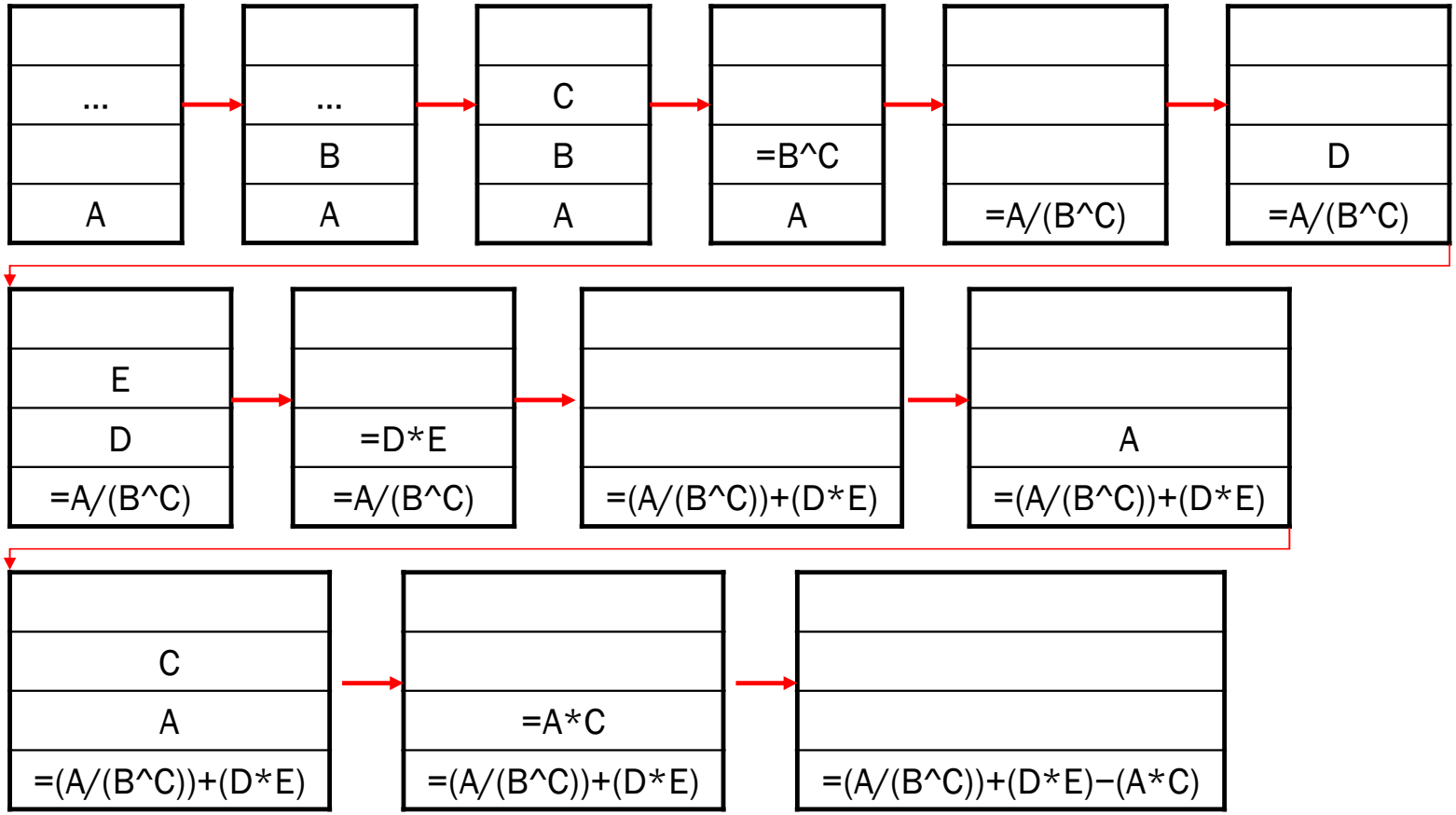




endToken = true

∴ output(top(s))

ABC^/DE*+AC*–


...
A
...
B
A
C
B
A
=B^C
A
=A/(B^C)
D
=A/(B^C)
E
D
=A/(B^C)
=D*E
=A/(B^C)
=(A/(B^C))+(D*E)
A
=(A/(B^C))+(D*E)
C
A
=(A/(B^C))+(D*E)
=A*C
=(A/(B^C))+(D*E)
=(A/(B^C))+(D*E)–(A*C)

Latihan Soal: ADT Stack

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Soal 1

Dengan menggunakan ADT Stack yang direpresentasikan sebagai array statik-eksplisit seperti yang dibahas di materi kuliah, realisasikan prosedur dan fungsi berikut ini:

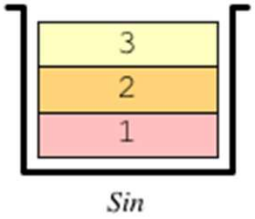
```
procedure copyStack(input sIn: Stack, output sOut: Stack)
{ Membuat salinan sOut }
{ I.S. sIn terdefinisi, sOut sembarang }
{ F.S. sOut berisi salinan sIn yang identik }
```

Soal 1

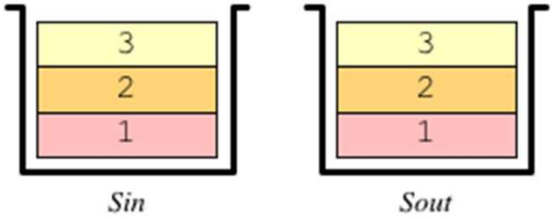
```
procedure inverseStack(input/output s: Stack)
{ Membalik isi suatu stack }
{ I.S. s terdefinisi }
{ F.S. Isi s terbalik dari posisi semula }

function mergeStack(s1,s2: Stack) → Stack
{ Menghasilkan sebuah stack yang merupakan hasil penggabungan s1
  dengan s2 dengan s1 berada di posisi lebih “bawah”. Urutan kedua
  stack harus sama seperti aslinya. }
{ Stack baru diisi sampai seluruh elemen s1 dan s2 masuk ke dalam
  stack, atau jika stack baru sudah penuh, maka elemen yang
  dimasukkan adalah secukupnya yang dapat ditampung. }
```

CopyStack(*Sin*, *Sout*)

Initial State:

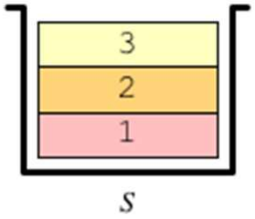


Final State:

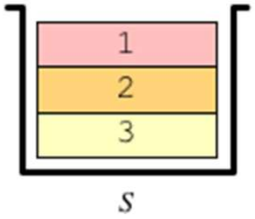


InverseStack(*S*)

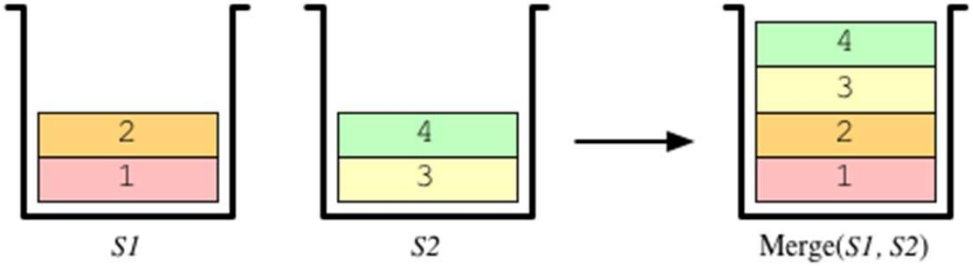
Initial State:



Final State:



MergeStack(*S1*, *S2*)



Soal 2

Dengan memanfaatkan mesin kata (versi 1), modifikasi dan lengkapi program Evaluasi Ekspresi Aritmatika pada slide materi kuliah.

Pita karakter berisi ekspresi aritmatika dengan masing-masing “kata” dipisahkan oleh satu atau lebih BLANK.

Kata yang merupakan operan merepresentasikan sebuah bilangan, contoh: “123” merepresentasikan bilangan 123

Contoh isi pita: 123 3 *

Dengan demikian, harus dibuat ADT Stack of Kata. Gunakan ADT Stack dengan representasi array statik-eksplisit.

Harus dibuat pula fungsi yang mengubah suatu Kata yang merepresentasikan operan menjadi angka/integer. Diasumsikan hanya ada integer positif atau 0.