

# ADT Stack dalam Bahasa C

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

```
#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)

```
/** Kontruktor/Kreator */
void CreateStack(Stack *s);
/* I.S. Sembarang */
/* F.S. Membuat sebuah Stack s yang kosong berkapasitas CAPACITY */
/* jadi indeksanya 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)

```
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: operand U 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 }

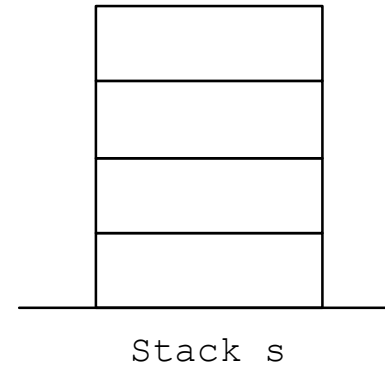
# (lanjutan)

## ALGORITMA

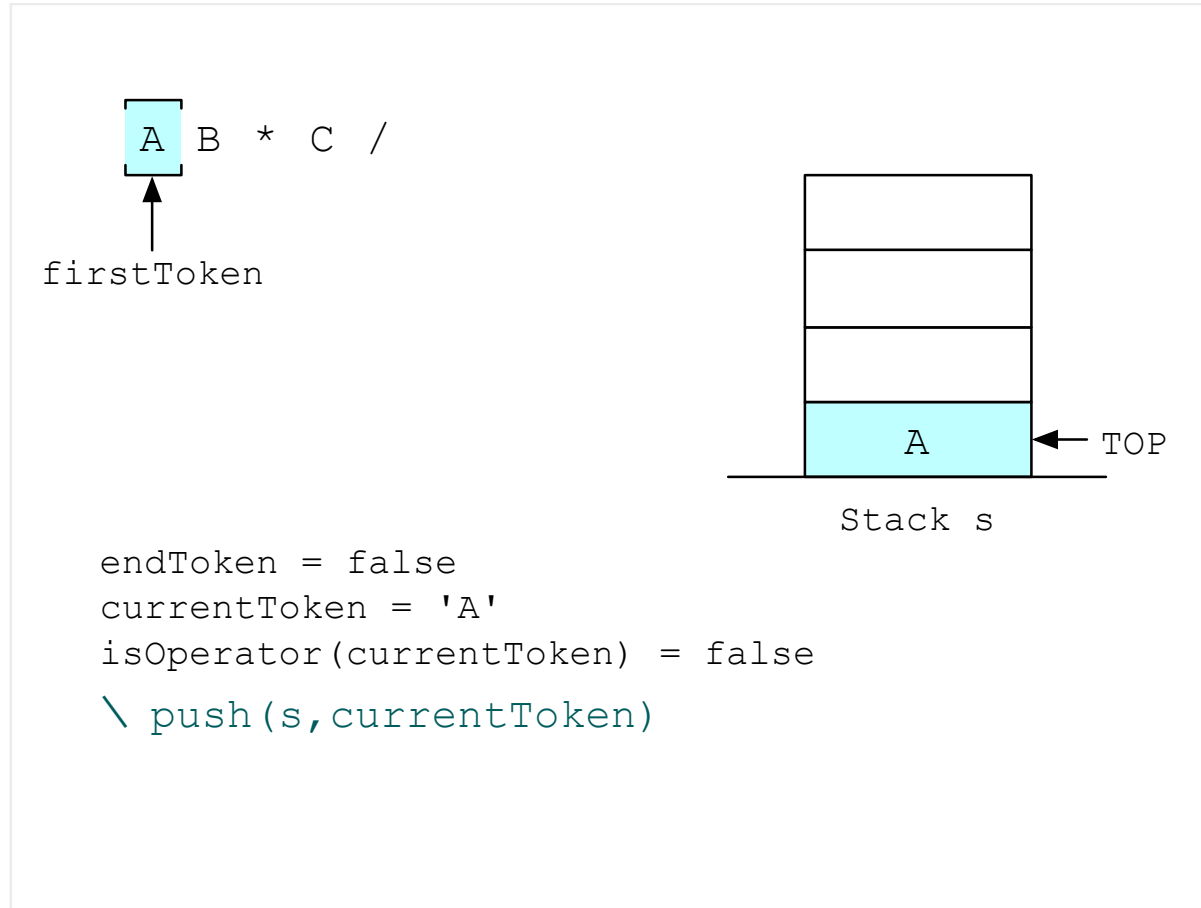
```
firstToken
if (endToken) then
    output ("Ekspresi kosong")
else
    repeat
        if not isOperator then
            push(s,currentToken)
        else
            pop(s,op2)
            pop(s,op1)
            push(s,evaluate(op1,op2, currentToken))
    nextToken
until (endToken)
output (top(s))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

# Evaluasi Ekspresi Aritmatika

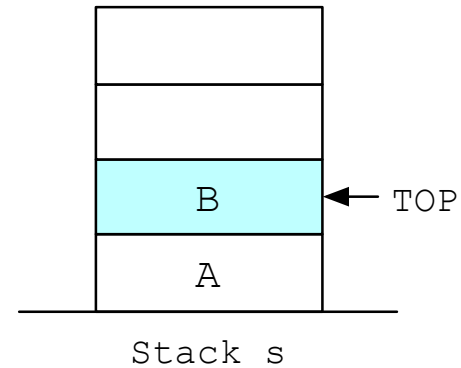
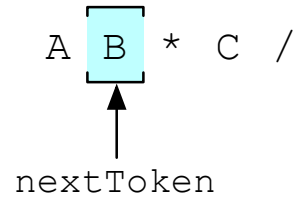
A B \* C /



# Evaluasi Ekspresi Aritmatika

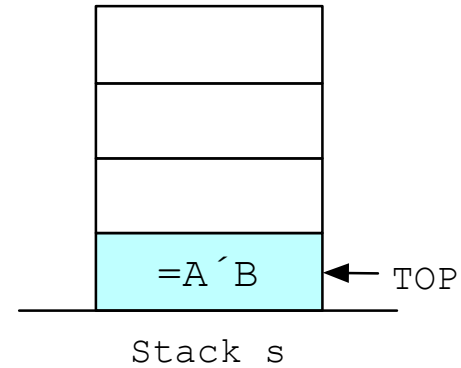
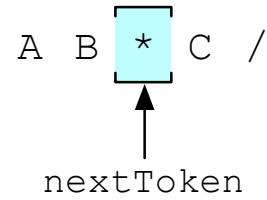


# Evaluasi Ekspresi Aritmatika



```
endToken = false  
currentToken = 'B'  
isOperator(currentToken) = false  
\ push(s, currentToken)
```

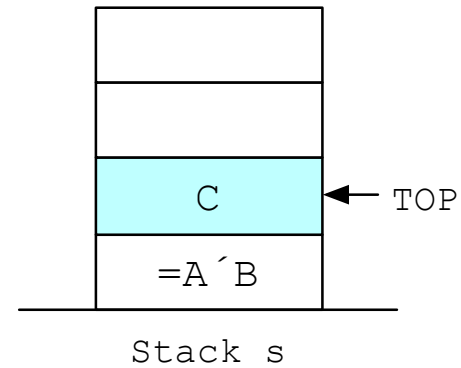
# Evaluasi Ekspresi Aritmatika



```
endToken = false  
currentToken = '*'  
isOperator(currentToken) = true  
\ pop(s, op2)  
  pop(s, op1)  
  push(s, evaluate(op1, op2, currentToken))
```

# Evaluasi Ekspresi Aritmatika

A B \* C /  
          ↑  
      nextToken



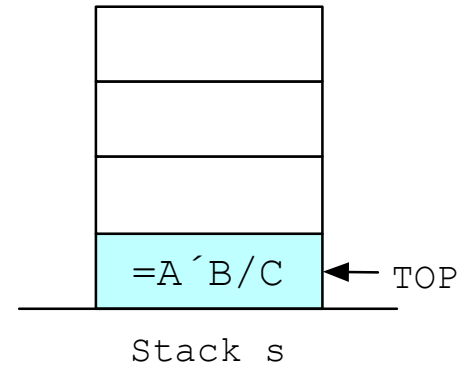
```
endToken = false  
currentToken = 'C'  
isOperator(currentToken) = false  
\ push(s, currentToken)
```



# Evaluasi Ekspresi Aritmatika

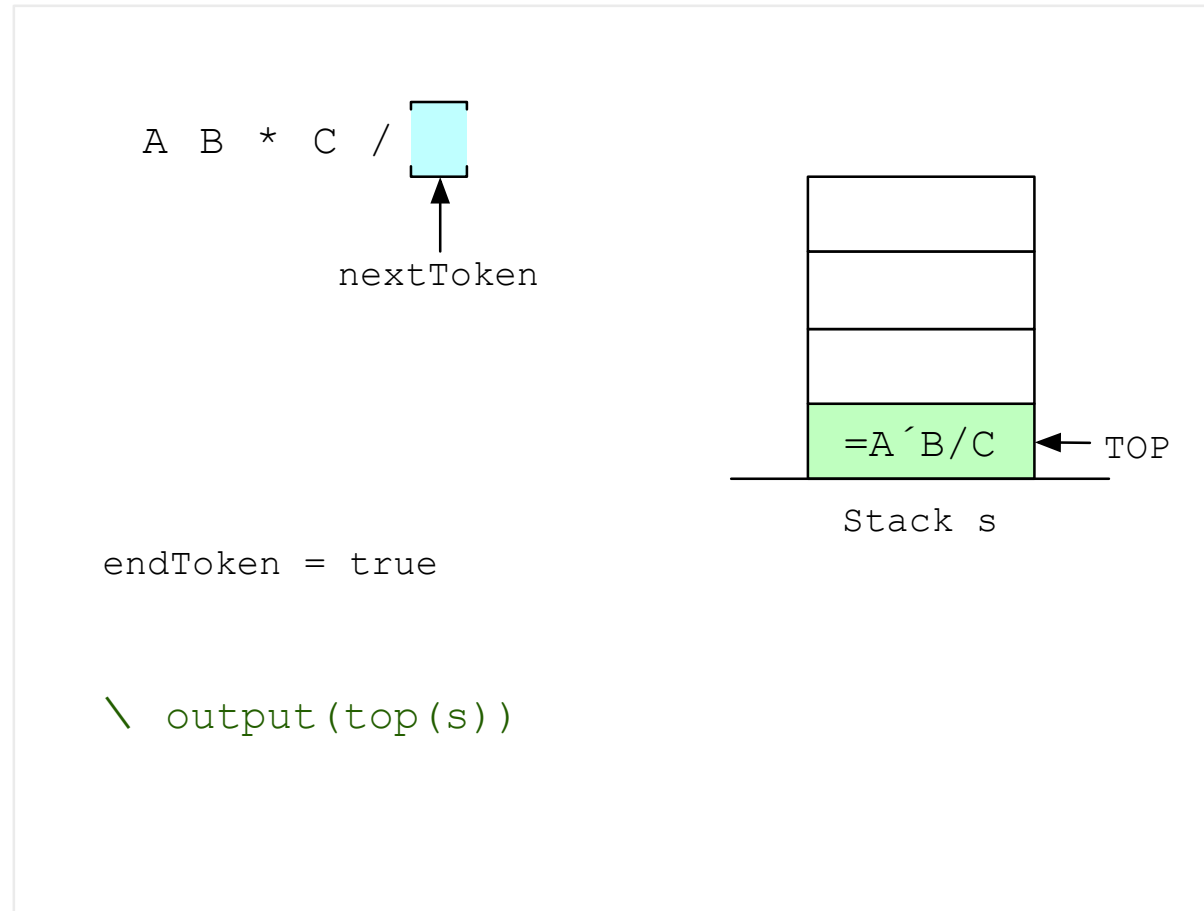
A B \* C /

↑  
nextToken



```
endToken = false
currentToken = '/'
isOperator(currentToken) = true
\ pop(s,op2)
  pop(s,op1)
  push(s,evaluate(op1,op2,currentToken))
```

# Evaluasi Ekspresi Aritmatika



# $ABC^{\wedge}/DE^{*}+AC^{*}-$

