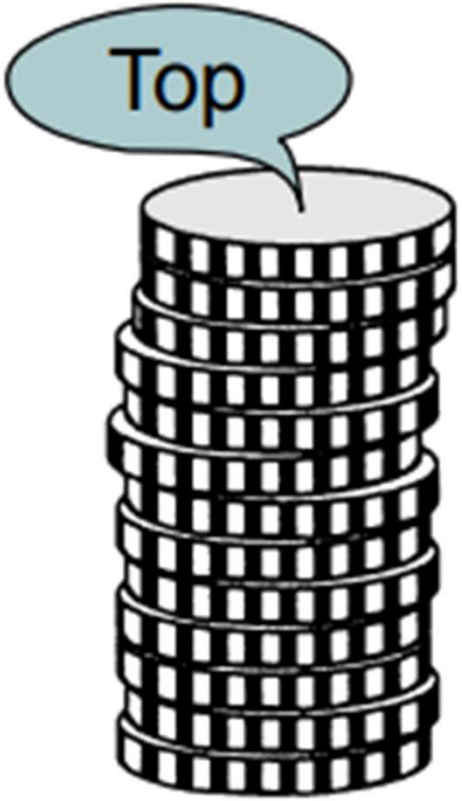
Fakultas Informatika
School of Computing
Telkom University

CDK2AAB4
STRUKTUR DATA

Stack

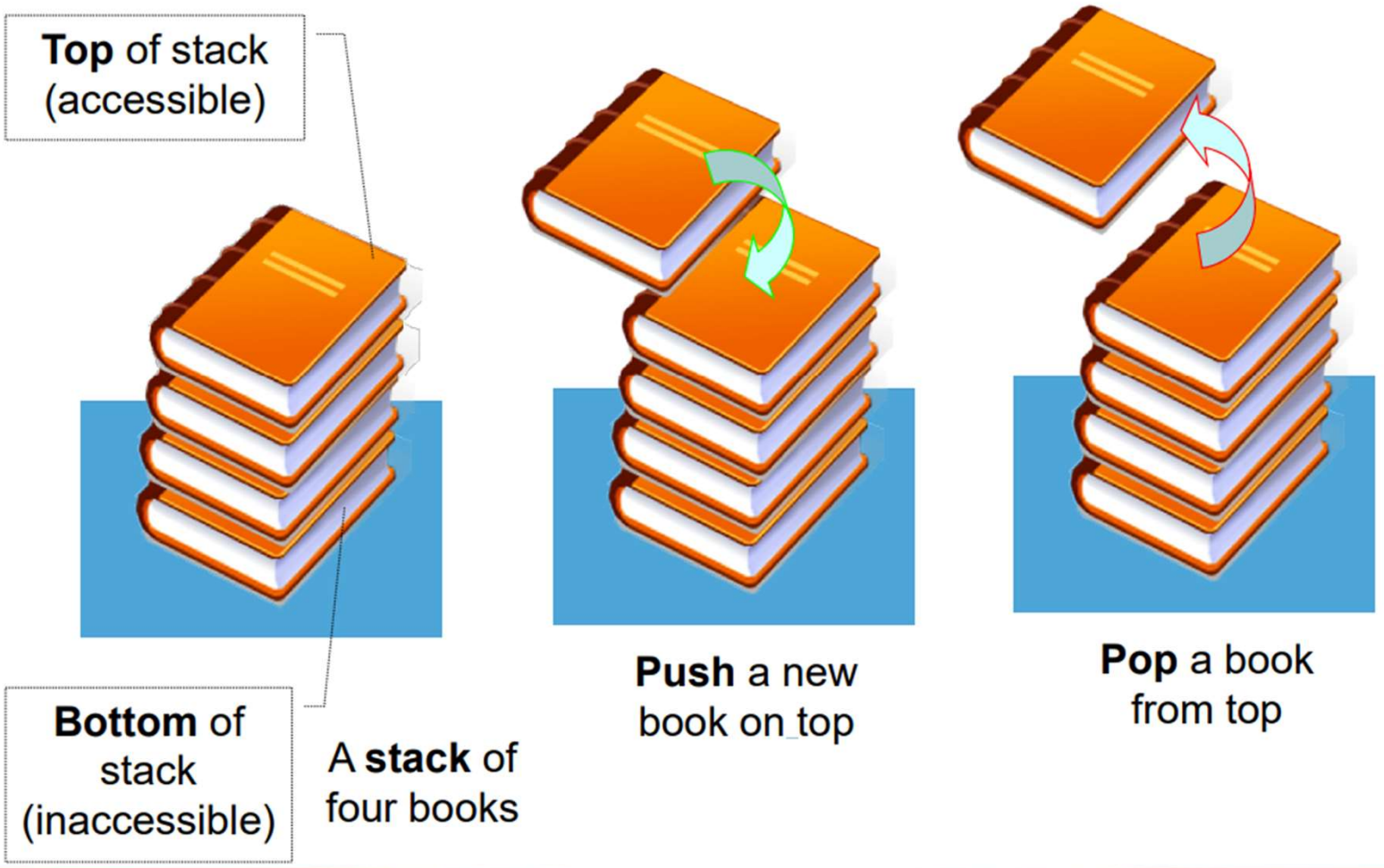
Stack Illustration

Stack of books



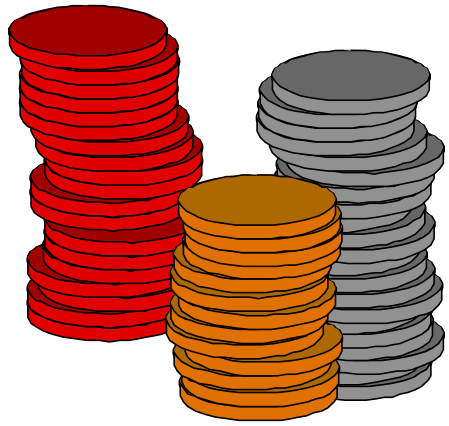
Stack of coins

Stack: Illustration



Stack

- An ordered list in which insertion and deletion are done at **one end**, called *top*.
  - The last element inserted is the first one to be deleted. Hence, it is called the **Last in First out (LIFO)** or First in Last out (FILO) list.



Primary Stack Operations

- **`push(el)`**—Put the element el on the top of the stack.
- **`pop()`**—Take the topmost element from the stack.

PUSH
(ADD A NEW ITEM
TO THE TOP)

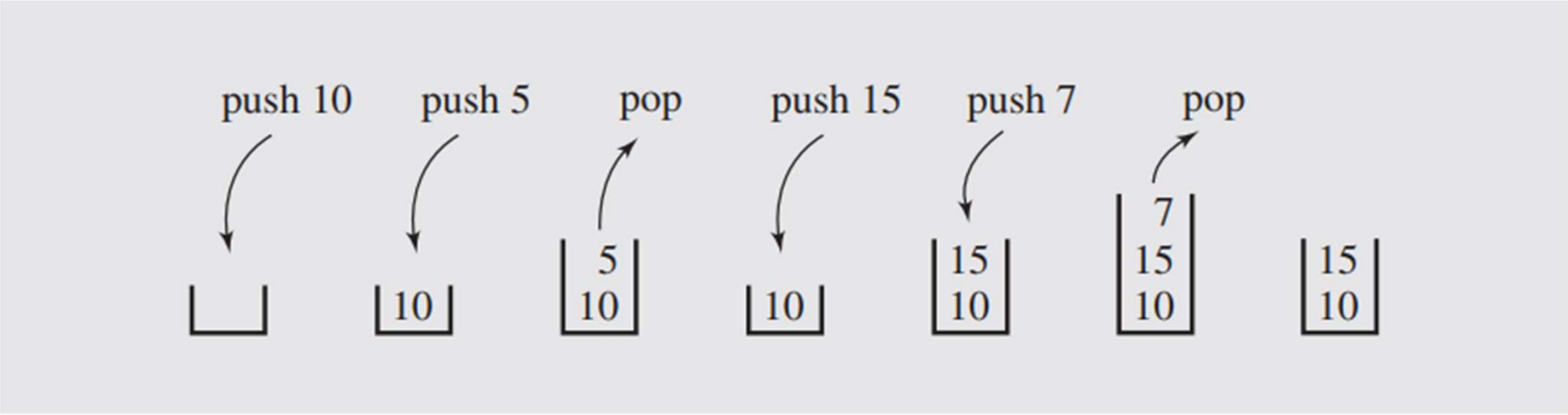
POP
(REMOVE THE TOPMOST
ITEM AND READ IT)

Auxiliary Stack Operations

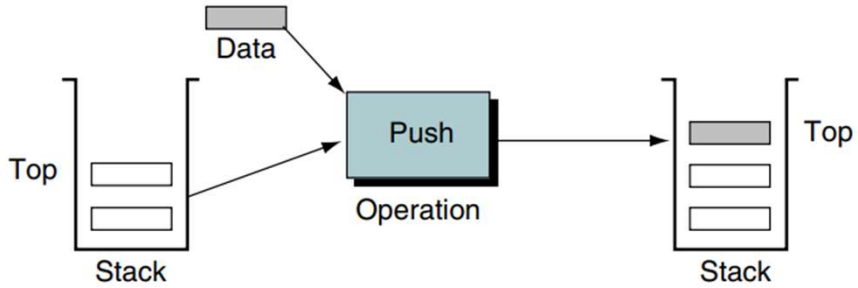
- **`isEmpty()`**—Check to see if the stack is empty.
- **`isFull()`**—Check to see if the stack is full.
- **`peek()`**—Return the topmost element in the stack without removing it.
- **`size()`**—Return the number of element in the stack.

Snapshots of the Stack

A series of operations executed on a stack.

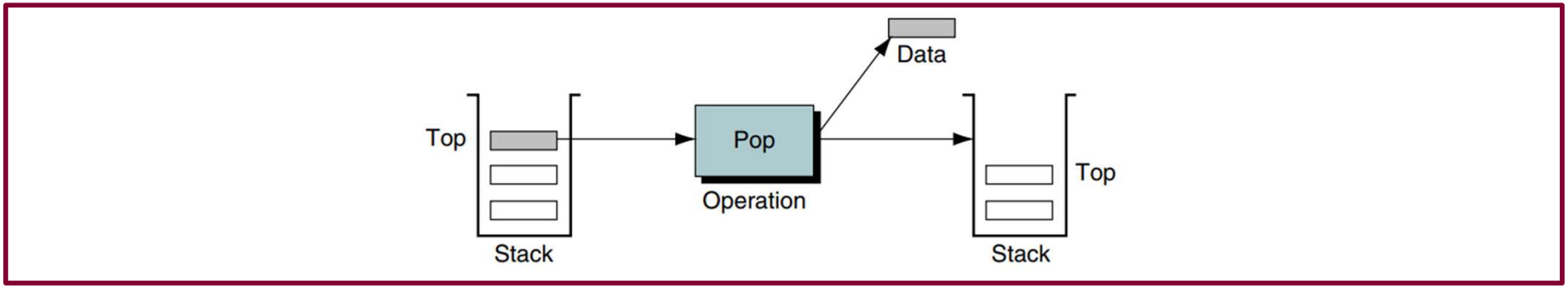


Potential Problem: Overflow



- **Push** adds an item at the top of the stack.
- After the push, the new item becomes the top.
- The only potential problem with this simple operation is that we must ensure that there is room for the new item.
- If there is not enough room, the stack is in an **overflow** state and the item cannot be added.

Potential Problem: Underflow



- When we **pop** a stack, we remove the item at the top of the stack and return it to the user.
- Because we have removed the top item, the next older item in the stack becomes the top.
- When the last item in the stack is deleted, the stack must be set to its empty state.
- If pop is called when the stack is empty, it is in an **underflow** state

Stack Applications

- Balancing of symbols
- Infix-to-postfix conversion
- Evaluation of postfix expression
- Implementing function calls (including recursion)
- Page-visited history in a Web browser [Back Buttons]
- Undo sequence in a text editor
- Matching Tags in HTML and XML

Stack Implementations

- Linked list implementation
- Array implementation

Linked List Implementation of Stacks

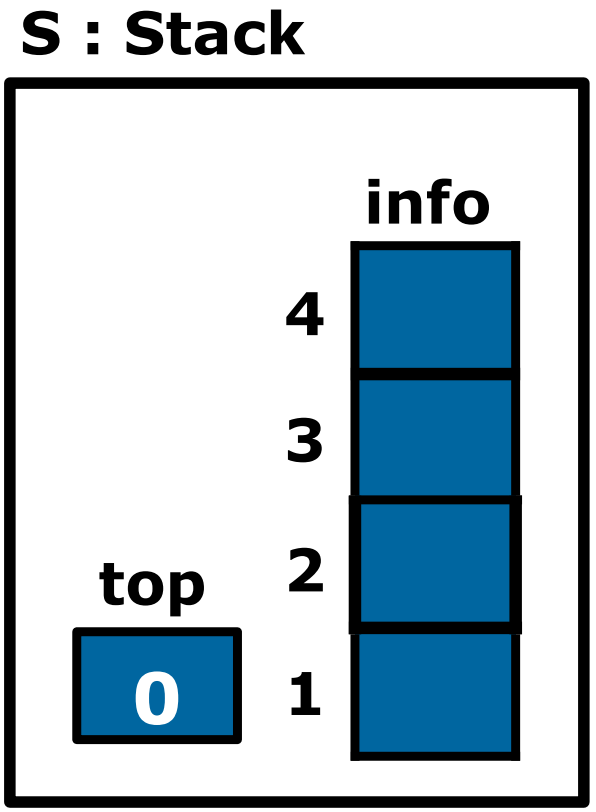
- The first implementation of a stack uses a singly linked list.
- We perform a push by inserting at the front of the list.
- We perform a pop by deleting the element at the front of the list.
- A peek operation merely examines the element at the front of the list, returning its value.

Array Implementation of Stacks

- An alternative implementation avoids links and is probably the more popular solution.
- If a stack's maximum size can be calculated before the program is written, an array implementation of a stack is more efficient than implementation using a linked list.
- In addition, an array stack is a more easily understood and natural picture of a stack.
- The fact that most modern machines have stack operations as part of the instruction set enforces the idea that the stack is probably the most fundamental data structure in computer science, after the array.

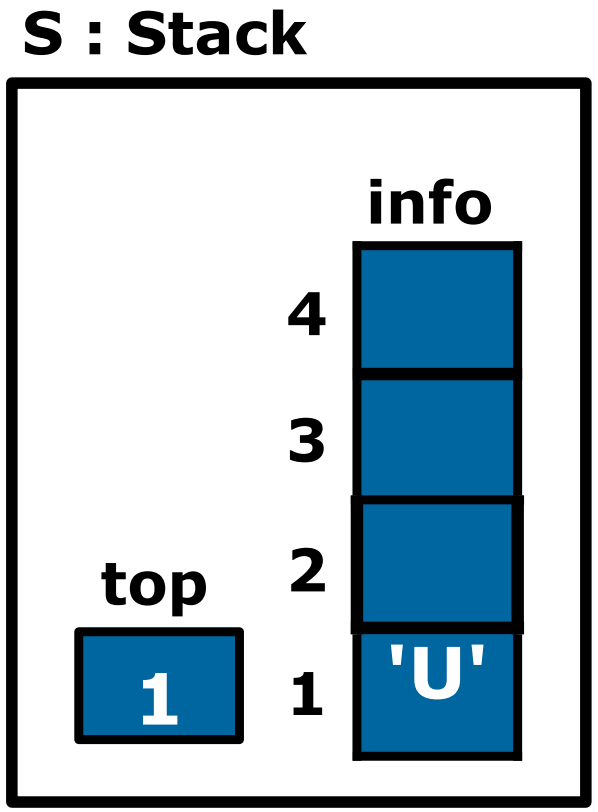
Array Representation of Stack

```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```



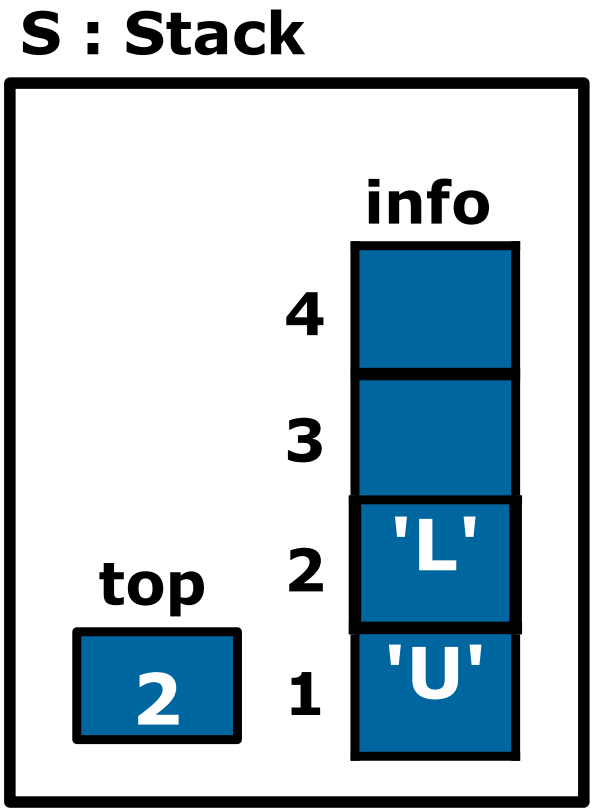
Array Representation of Stack

```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```



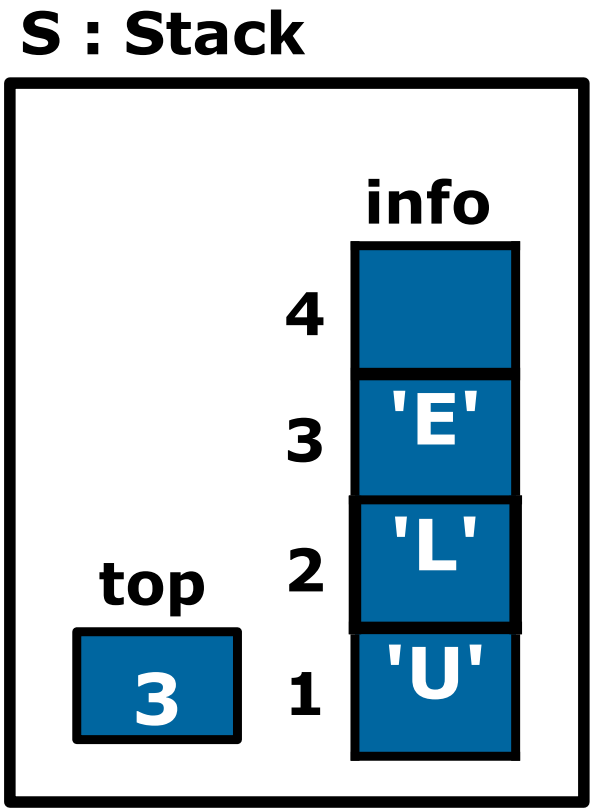
Array Representation of Stack

```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```



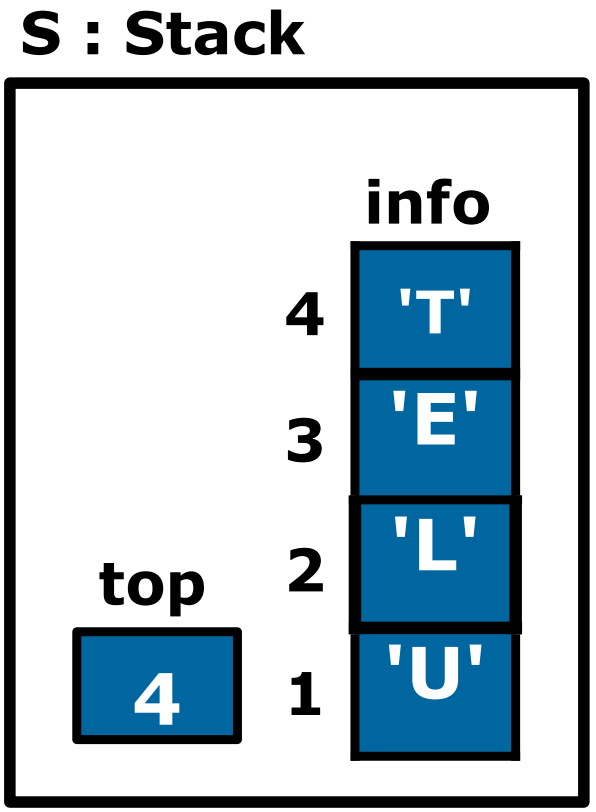
Array Representation of Stack

```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```



Array Representation of Stack

```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

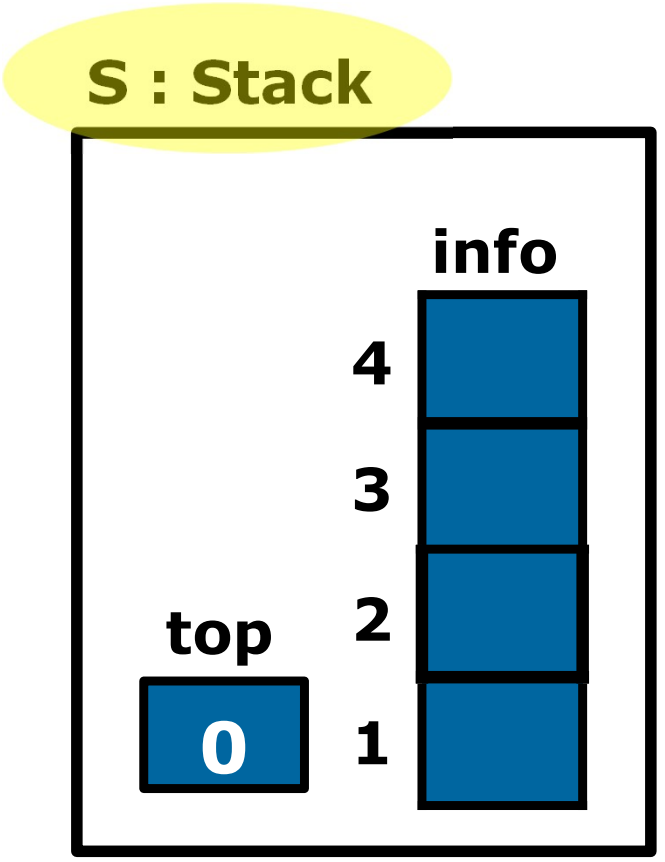


Stack Operations (Primitives)

- Put the element on the top of the stack.
- Take the topmost element from the stack.
- Check to see if the stack is empty.
- Check to see if the stack is full.
- Return the topmost element in the stack without removing it.
- Return the number of element in the stack.

```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

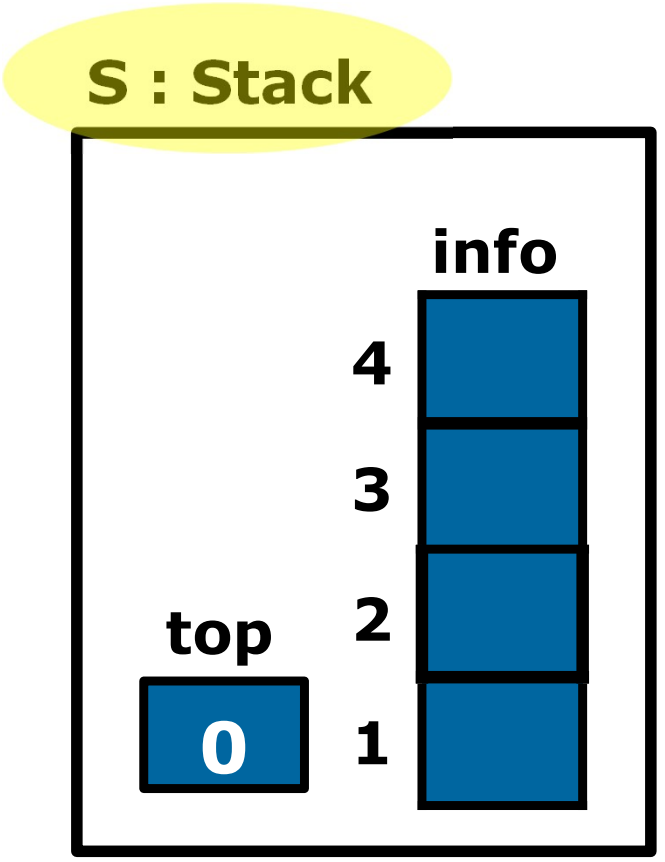
Stack Operations Illustration



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Stack Operations Illustration

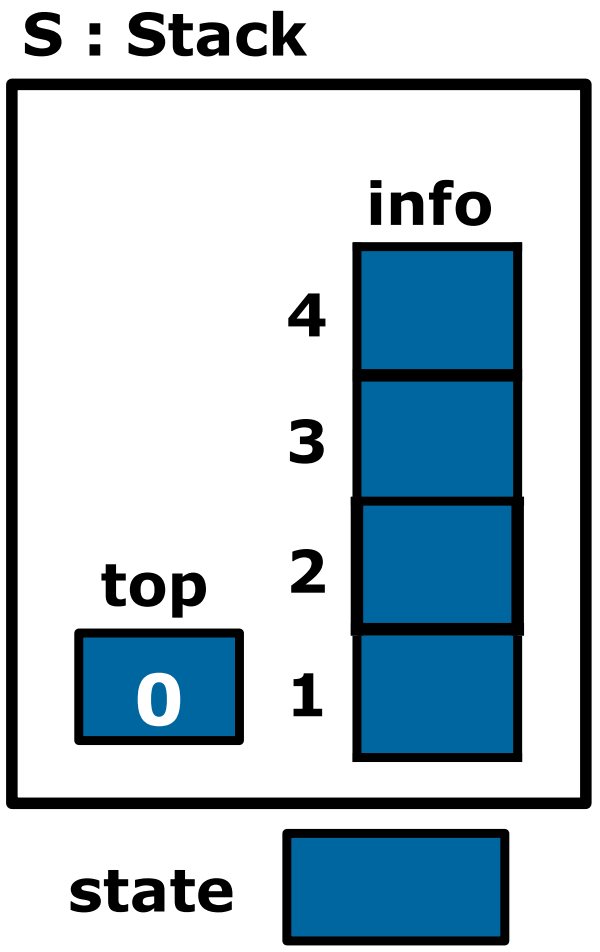
```
S = createStack()
```



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Stack Operations Illustration

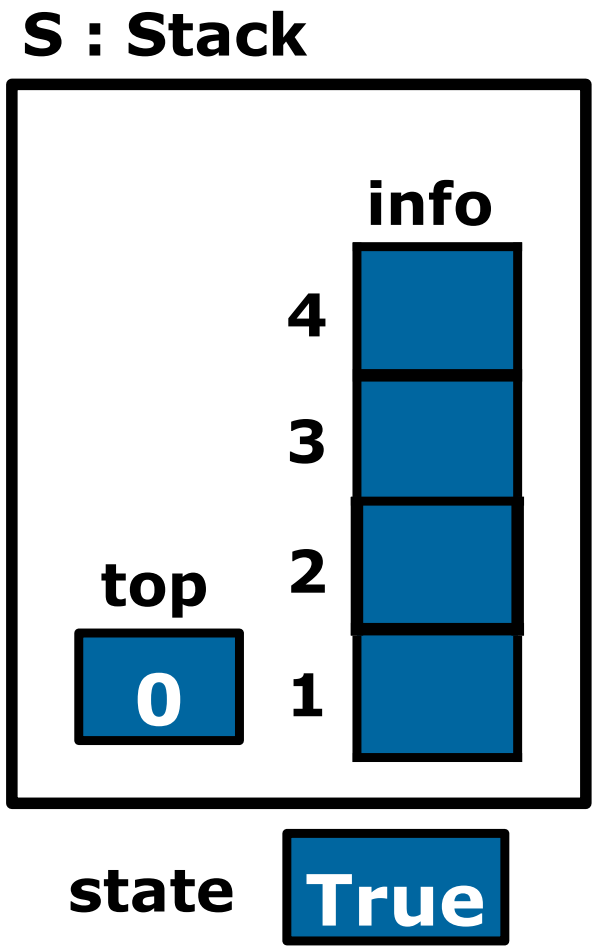
```
S = createStack()
state = isEmpty( S )
```



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Stack Operations Illustration

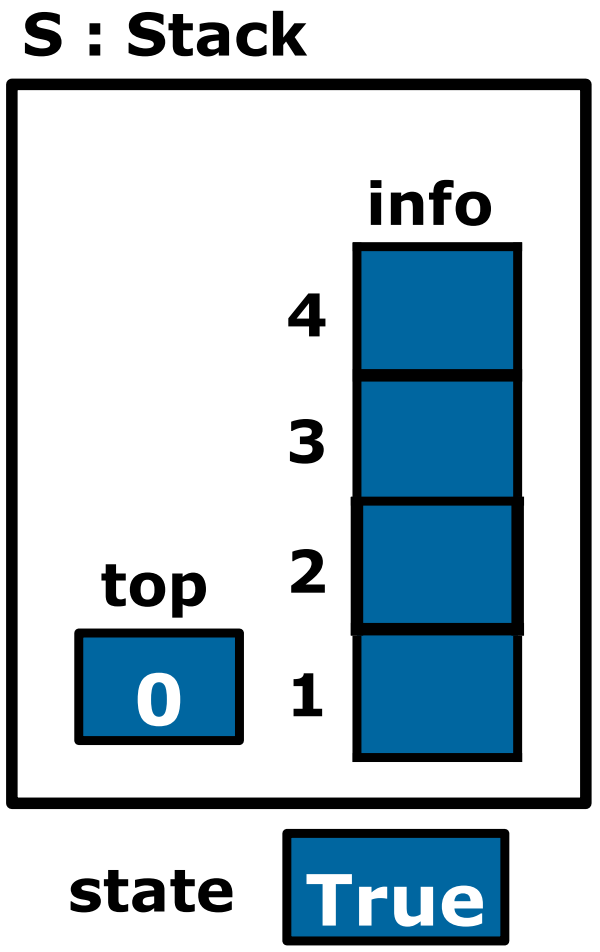
```
S = createStack()
state = isEmpty( S )
```



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Stack Operations Illustration

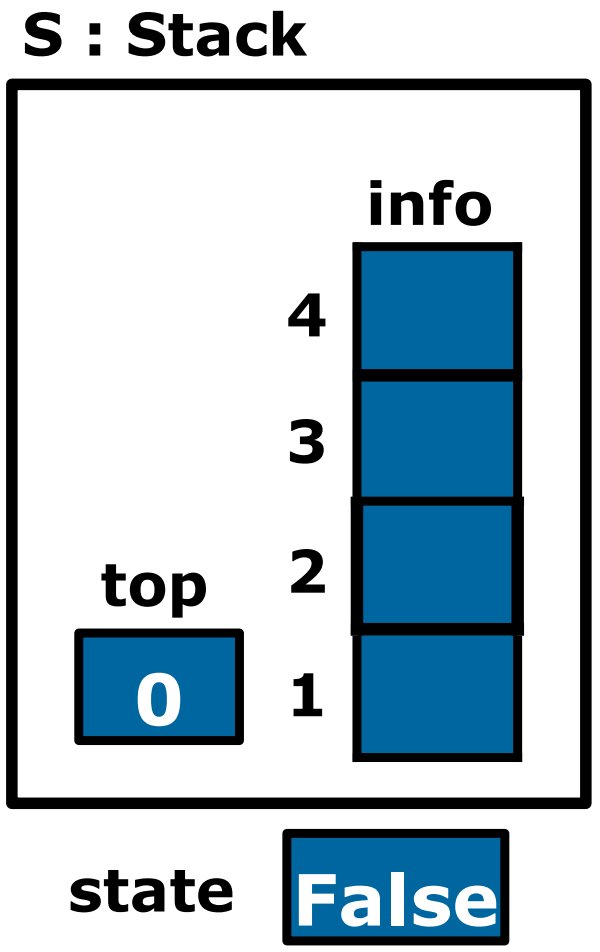
```
S = createStack()
state = isEmpty( S )
state = isFull( S )
```



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Stack Operations Illustration

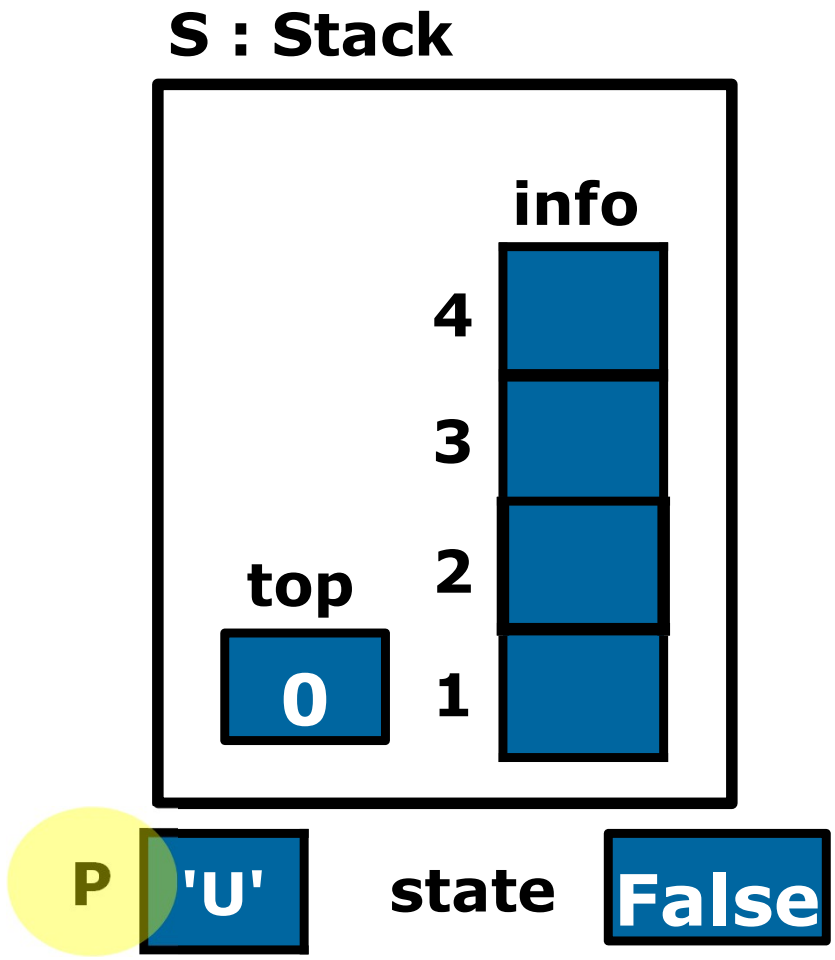
```
S = createStack()
state = isEmpty( S )
state = isFull( S )
```



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Stack Operations Illustration

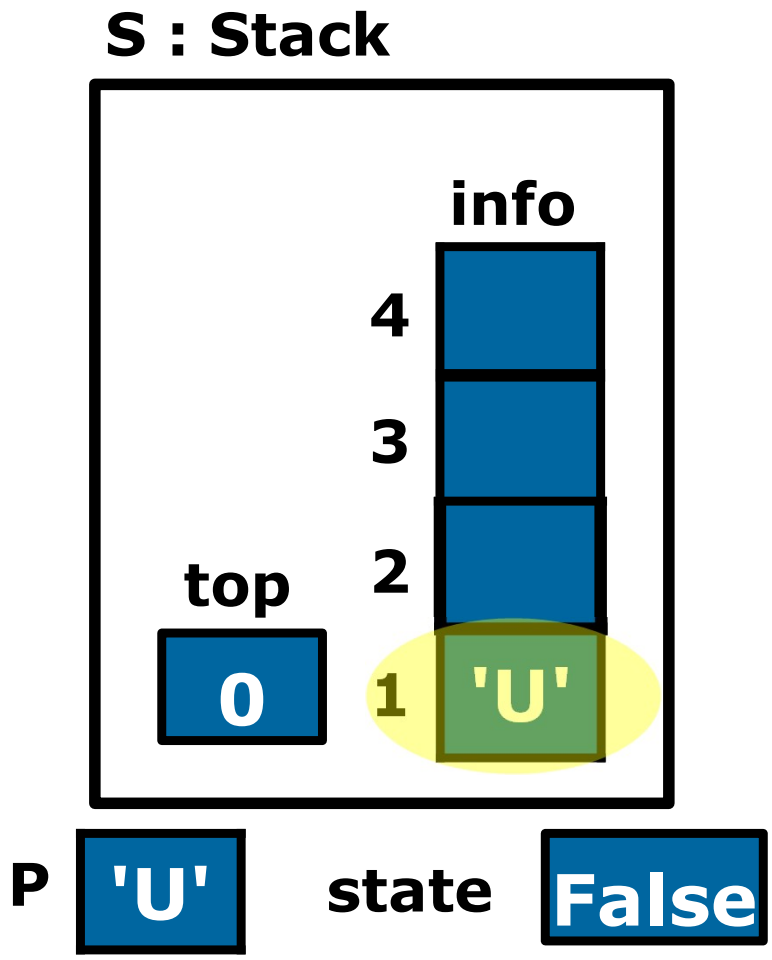
```
S = createStack()
state = isEmpty( S )
state = isFull( S )
push( S, P )
```



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Stack Operations Illustration

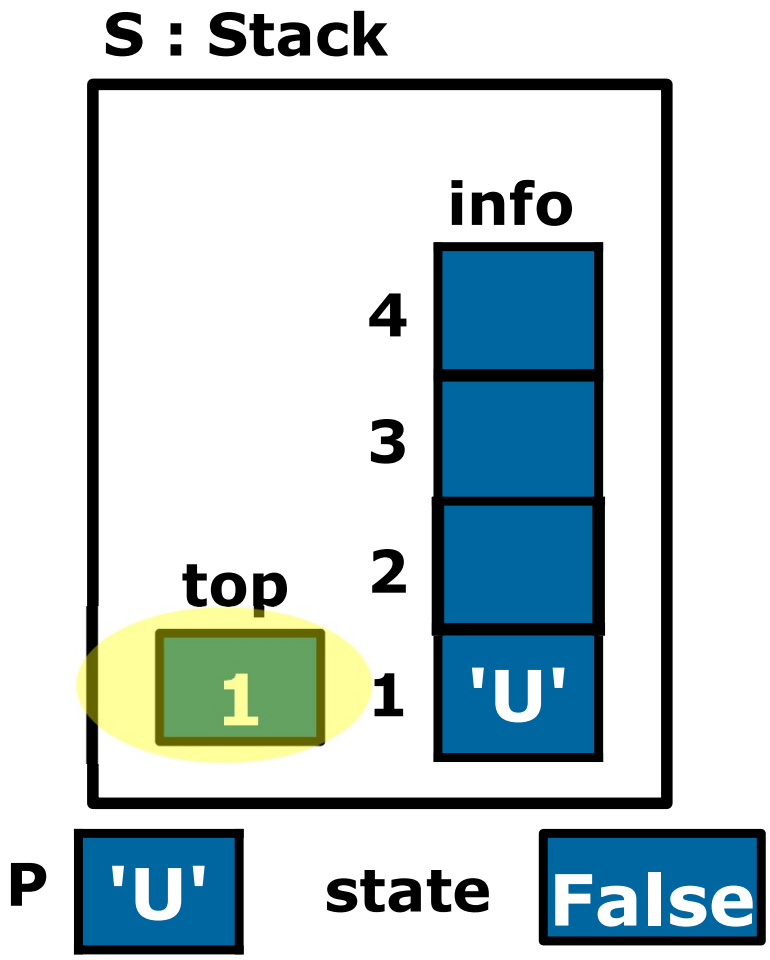
```
S = createStack()
state = isEmpty( S )
state = isFull( S )
push( S, P )
```



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Stack Operations Illustration

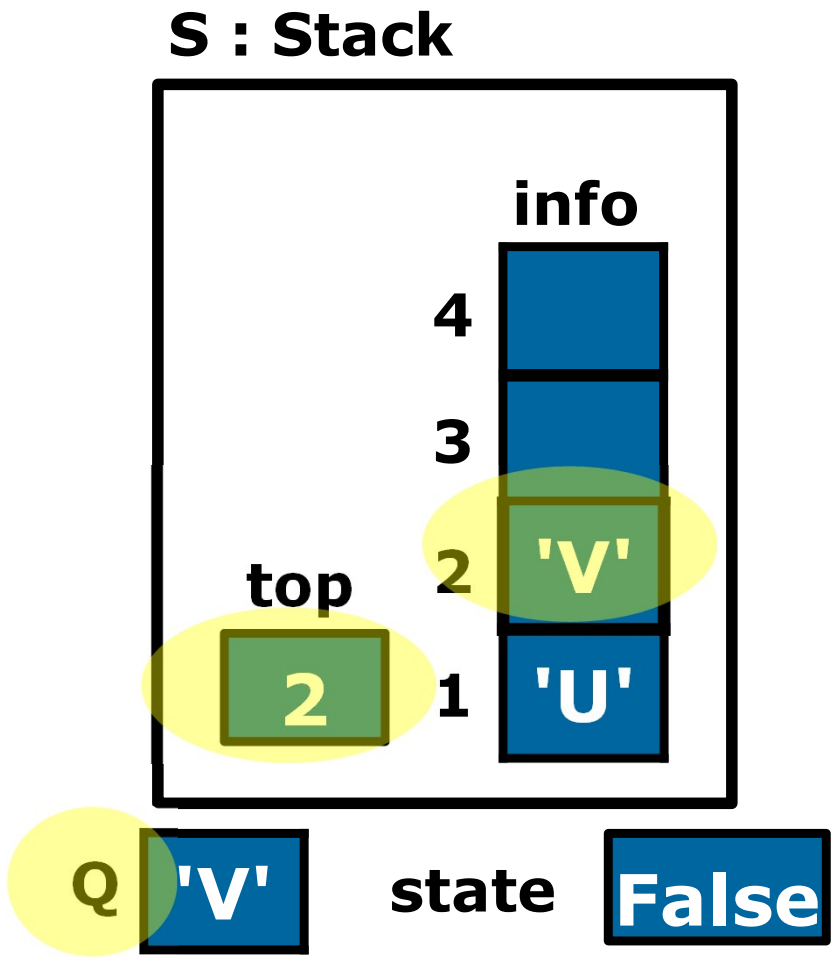
```
S = createStack()
state = isEmpty( S )
state = isFull( S )
push( S, P )
```



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Stack Operations Illustration

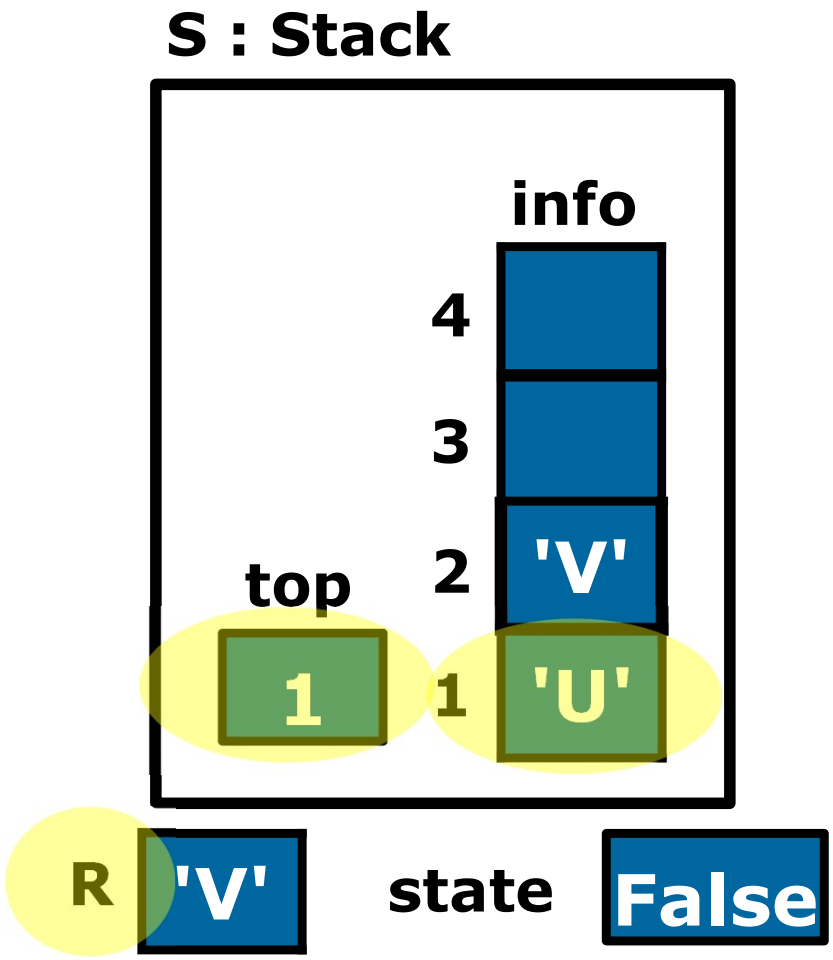
```
S = createStack()
state = isEmpty( S )
state = isFull( S )
push( S, P )
push( S, Q )
```



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Stack Operations Illustration

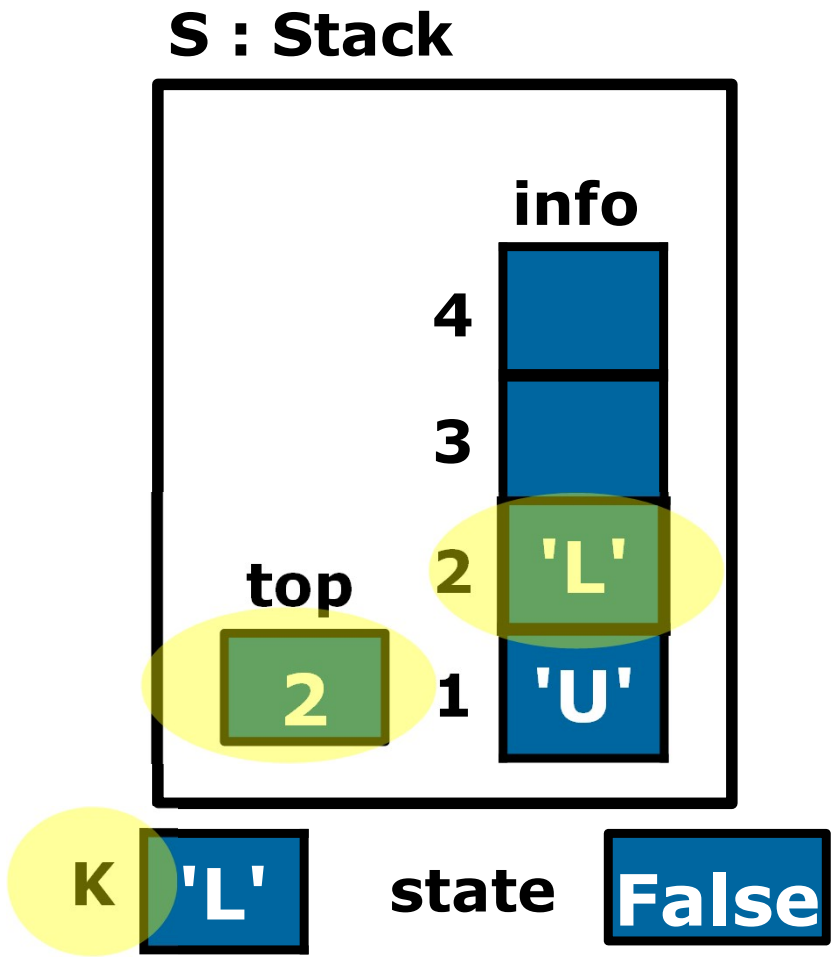
```
S = createStack()
state = isEmpty( S )
state = isFull( S )
push( S, P )
push( S, Q )
pop( S, R )
```



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Stack Operations Illustration

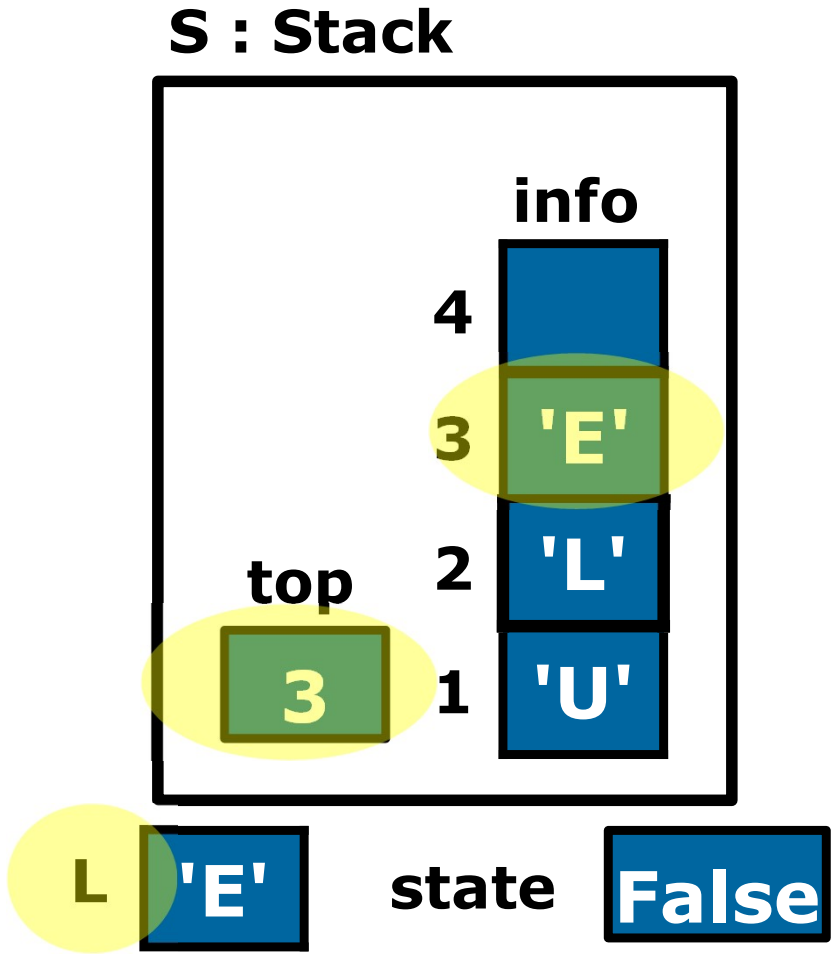
```
S = createStack()
state = isEmpty( S )
state = isFull( S )
push( S, P )
push( S, Q )
pop( S, R )
push( S, K )
```



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Stack Operations Illustration

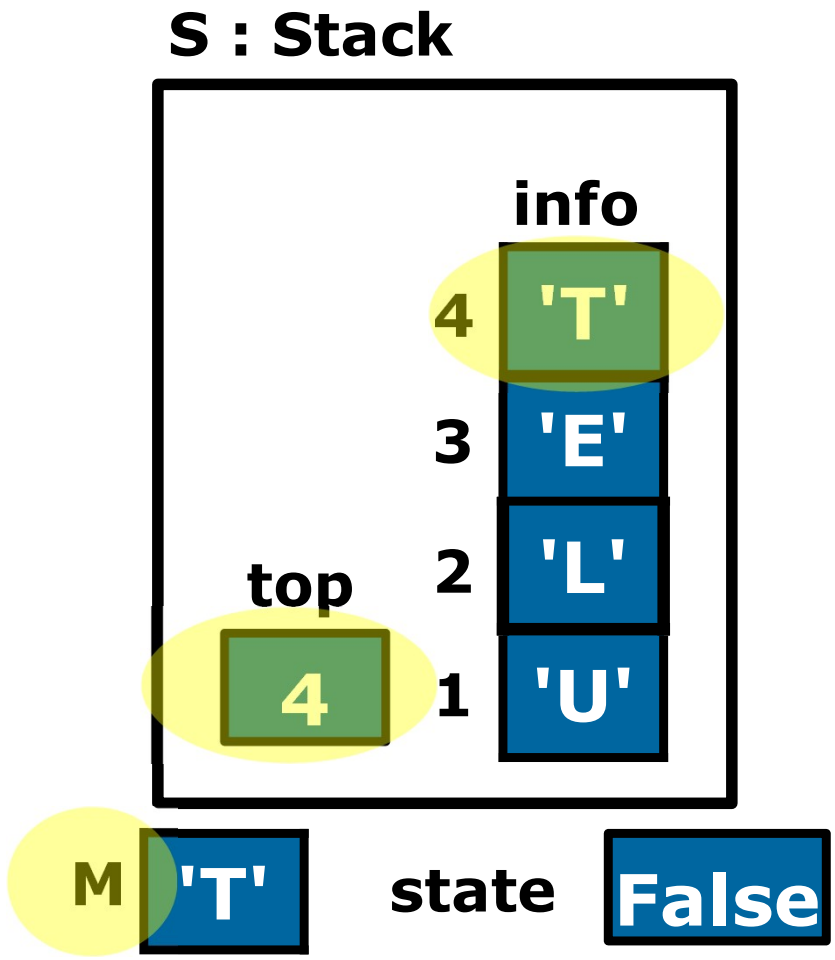
```
S = createStack()
state = isEmpty( S )
state = isFull( S )
push( S, P )
push( S, Q )
pop( S, R )
push( S, K )
push( S, L )
```



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Stack Operations Illustration

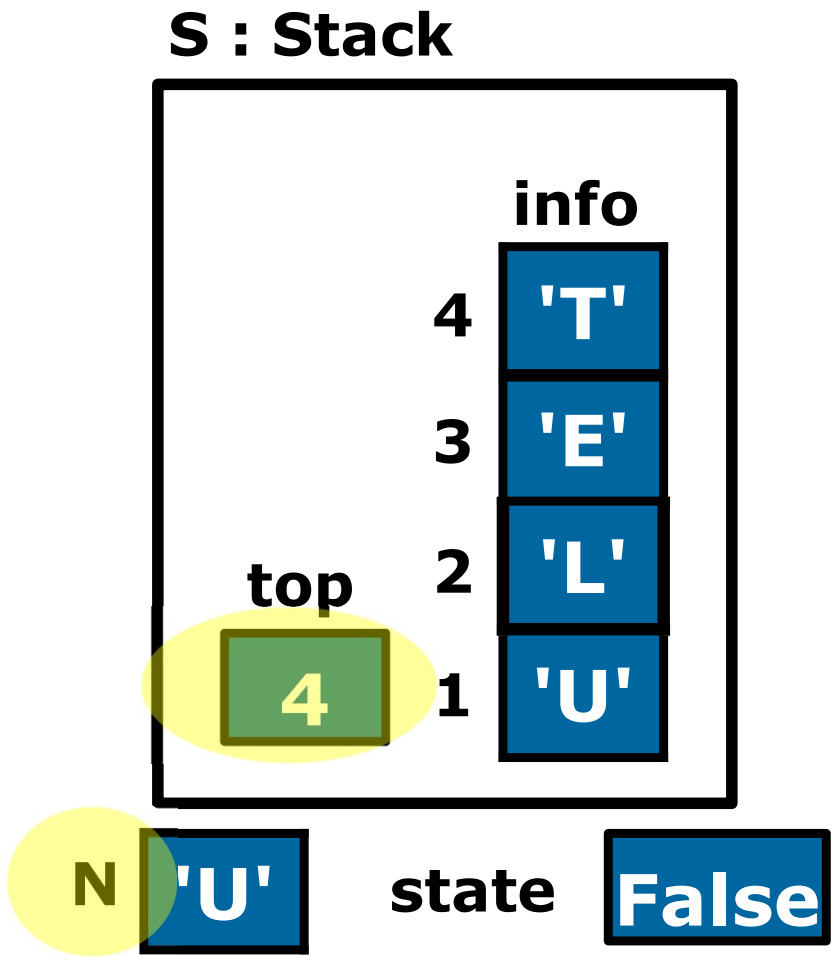
```
S = createStack()
state = isEmpty( S )
state = isFull( S )
push( S, P )
push( S, Q )
pop( S, R )
push( S, K )
push( S, L )
push( S, M )
```



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Stack Operations Illustration

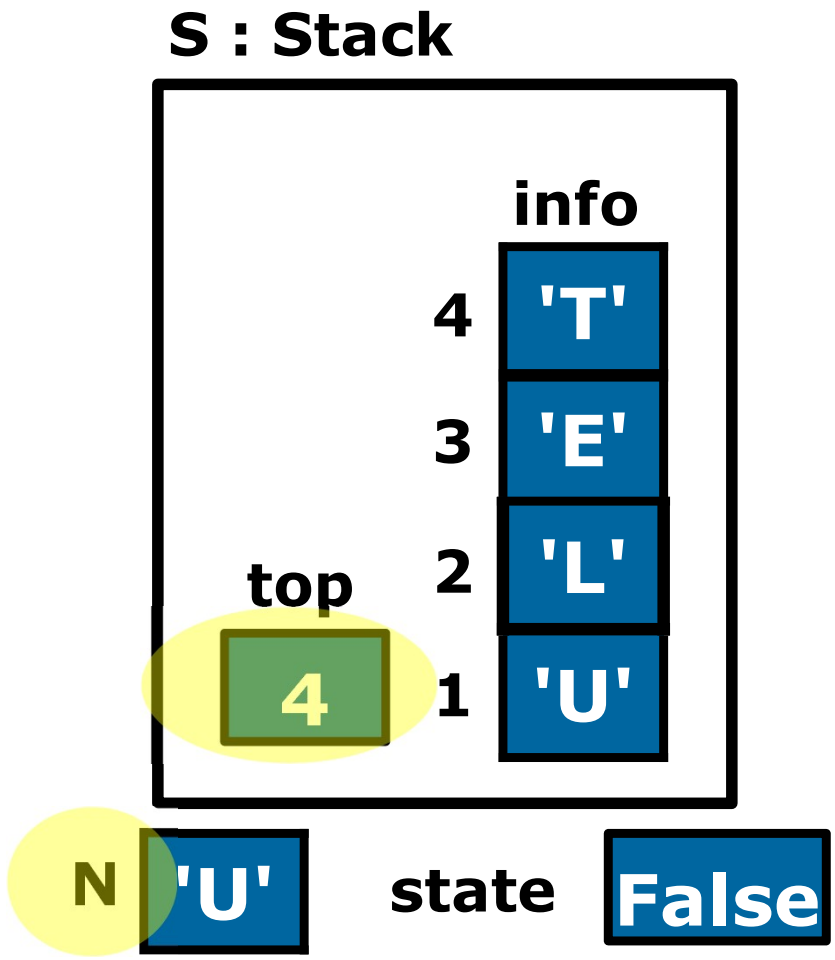
```
S = createStack()
state = isEmpty( S )
state = isFull( S )
push( S, P )
push( S, Q )
pop( S, R )
push( S, K )
push( S, L )
push( S, M )
push( S, N )
```



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Stack Operations Illustration

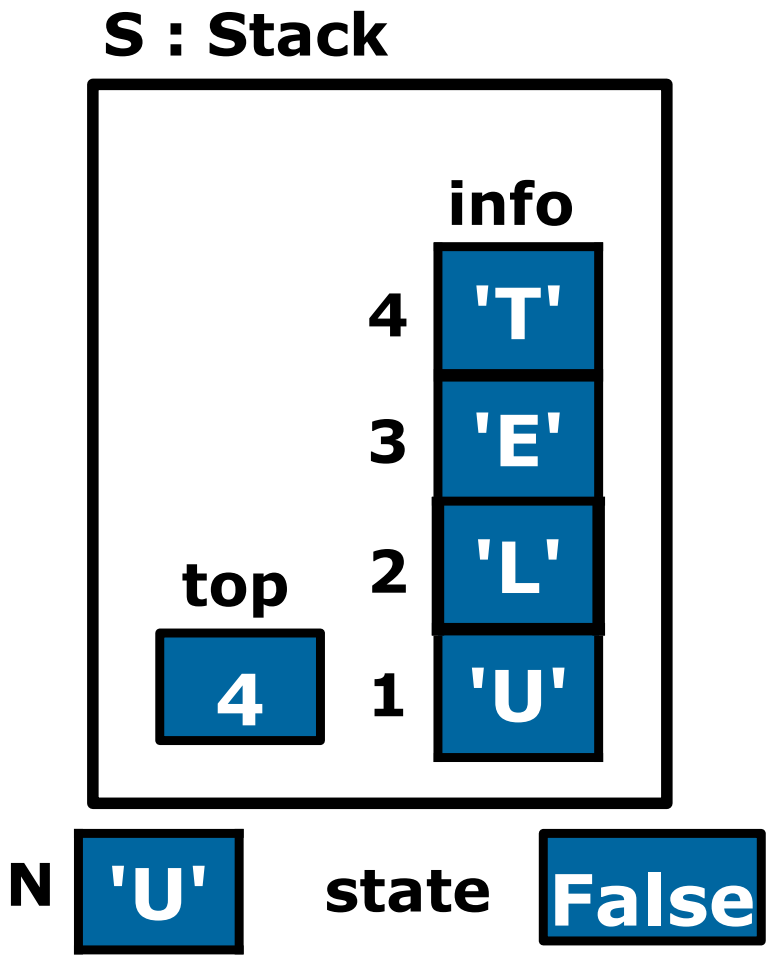
```
S = createStack()
state = isEmpty( S )
state = isFull( S )
push( S, P )
push( S, Q )
pop( S, R )
push( S, K )
push( S, L )
push( S, M )
push( S, N )        { fail }
```



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Stack Operations Illustration

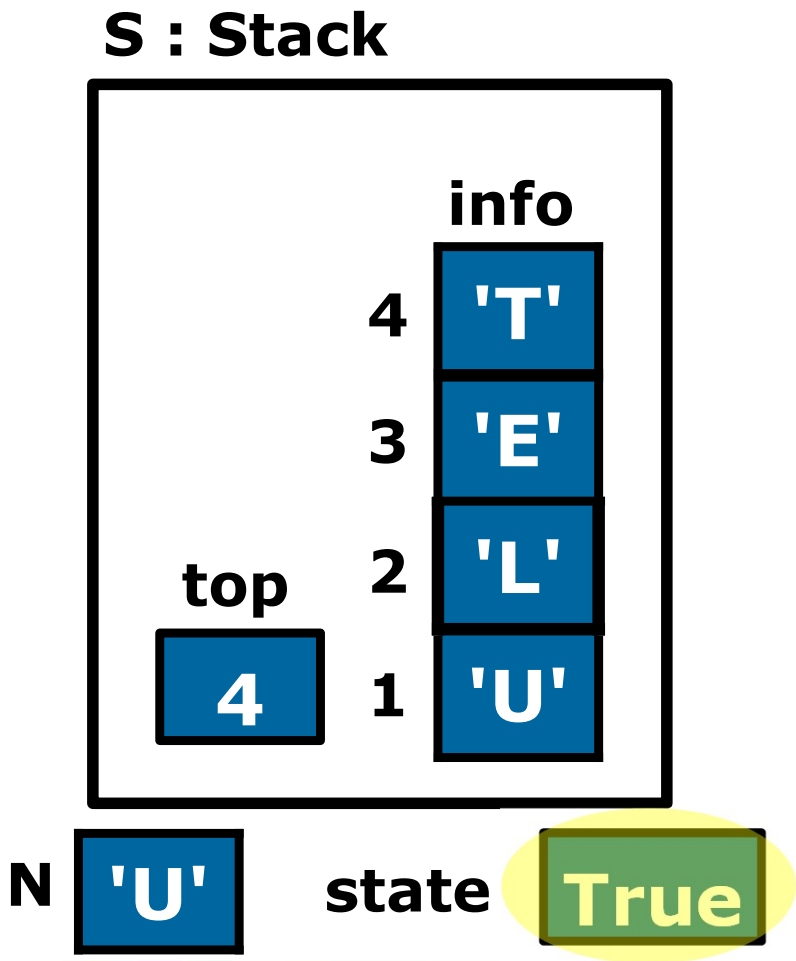
```
S = createStack()
state = isEmpty( S )
state = isFull( S )
push( S, P )
push( S, Q )
pop( S, R )
push( S, K )
push( S, L )
push( S, M )
push( S, N )      { fail }
state = isFull( S )
```



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Stack Operations Illustration

```
S = createStack()
state = isEmpty( S )
state = isFull( S )
push( S, P )
push( S, Q )
pop( S, R )
push( S, K )
push( S, L )
push( S, M )
push( S, N )      { fail }
state = isFull( S )
```



```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Implementation: `createStack`

```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Implementation: `isEmpty`

```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Implementation: `isFull`

```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Implementation: `push`

```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Implementation: pop

```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Implementation: peek

ıg. }

```
constant MAXSIZE : integer = 4
type Infotype : char
type Index : integer

type Stack: <
   info : array [1..MAXSIZE] of Infotype
   top : Index >
```

Implementation: `size`

Question?


TRAIN YOUR BRAIN

Train your Brain!

- Implementasikan fungsi/prosedur berikut dengan menggunakan primitif Stack!
  - `procedure swap( in/out S : Stack )`
  - `procedure popth( in/out S: Stack, in Idx: Index, in/out P : Infotype )`
  - `procedure printStack( in S : Stack )`
  - `function isPalindrome( in S : Stack ) → Boolean`
  - `Procedure pushSorted( in/out S : Stack, in P : Infotype )`

Stack Application

INFIX TO POSTFIX CONVERSION

Why use Postfix (or Prefix)

- INFIX notations are not as simple as they seem specially while evaluating them.
- To evaluate an infix expression, we need to consider operators' priority (precedence) and associative property
  - for example, expression 3 + 5 * 4 evaluate to **32**, i.e. (3 + 5) * 4, or **23**, i.e. 3 + (5 * 4)

Infix Expression is Hard to Parse

- Need operator priorities, tie breaker, and delimiters.
- This makes computer evaluation more difficult than is necessary.
- Postfix and prefix expression forms do not rely on operator priorities, a tie breaker, or delimiters.
- So, it is easier to evaluate expressions that are in these forms.

Examples of Infix to Postfix and Prefix

Infix	PostFix	Prefix
A+B	AB+	+AB
(A+B) * (C + D)	AB+CD+*	*+AB+CD
A-B/(C*D^E)	ABCDE^*/-	-A/B*C^DE

No brackets necessary

Example : Infix to Postfix

2*3/(2-1)+5*3

Rules:

- **Constant**, no push, "print" only
- **Operator**, push to stack (if op. has higher precedence than top, or pop & push if lower/same)
- **Delimiter**, push (opening) or pop & "print" (closing) until opening delim. found. Delimiter is not printed.

Expression	Stack	Output
2	Empty	2
*	*	2
3	*	23
/	/	23*
(	/(	23*
-	/(-	23*2
1	/(-	23*21
)	/	23*21-
+	+	23*21-/
5	+	23*21-/5
*	+*	23*21-/5
3	+*	23*21-/53
	Empty	23*21-/53*+

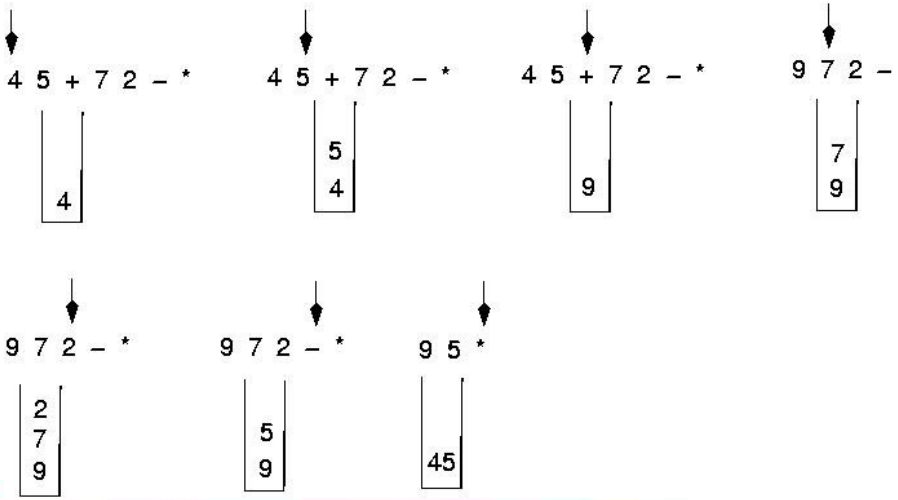
(bottom stack)

Example : Postfix Evaluation

45+72-*

Rules:

- **Constant**, push
- **Operator**, pop & evaluate, push evaluation result



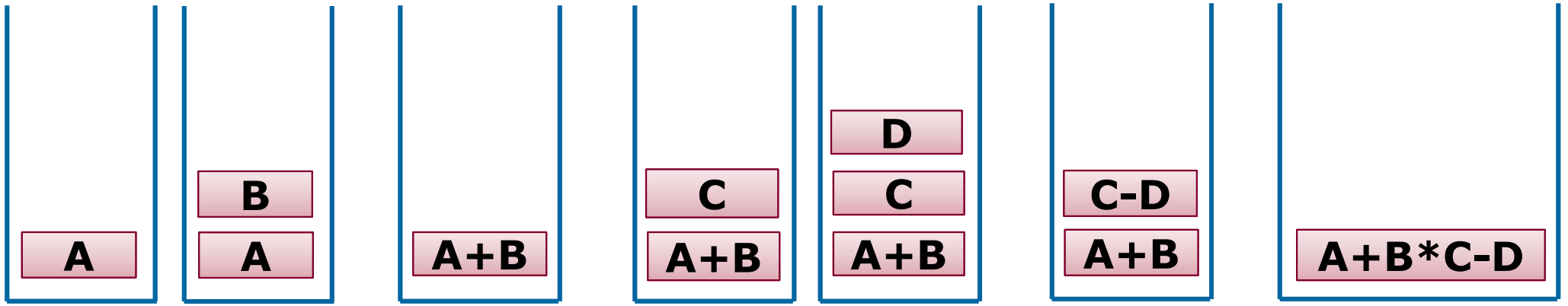
Expression
4
5
+
7
2
-
*

	Stack
(bottom stack)	4
	4 5
	9
	9 7
	9 7 2
	9 5
	45

Evaluation: Postfix to Infix

Evaluate the postfix expression: **AB+CD-***

Start scanning from left to right →



Push A (operand)

Push B (operand)

Operator + Pop 2 elements A+B Push A-B

Push C (operand)

Push D (operand)

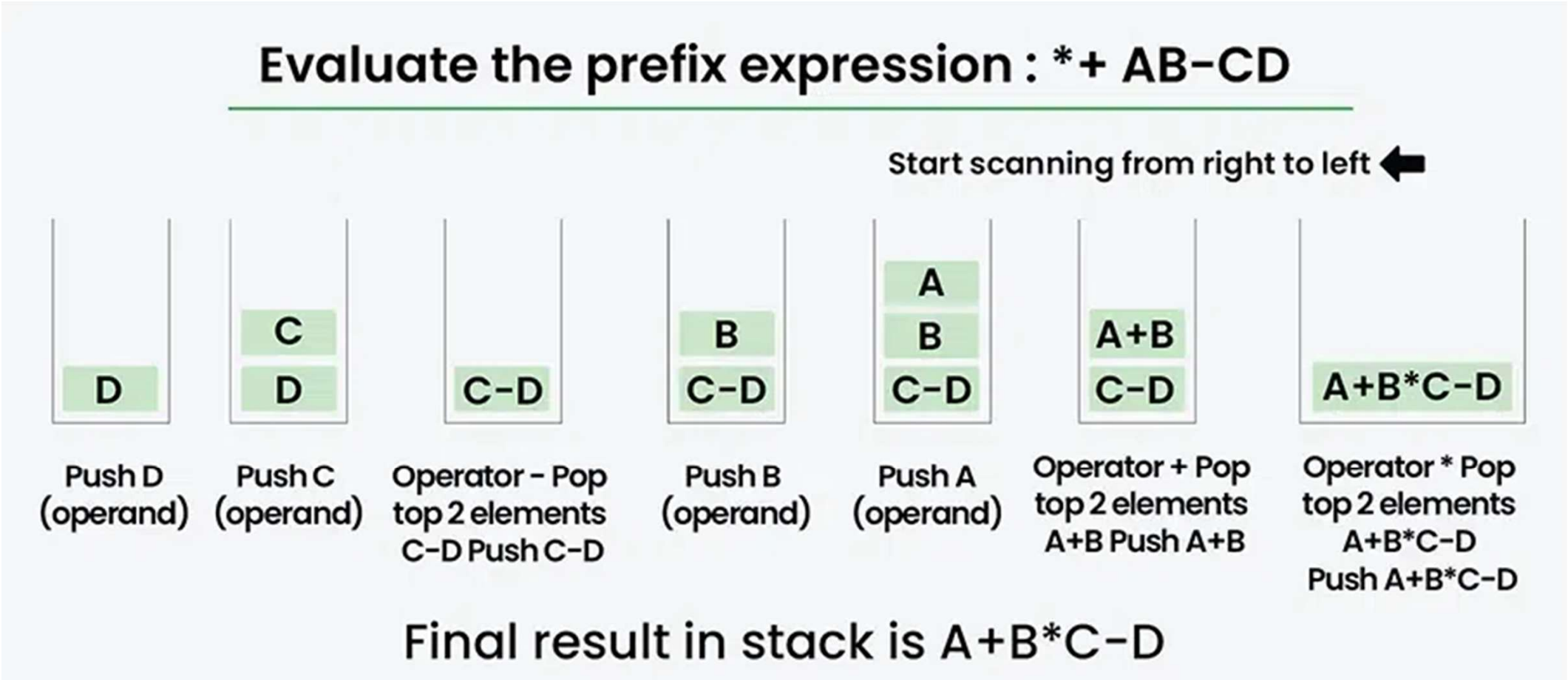
Operator – Pop 2 elements C-D Push C-D

Operator * Pop 2 elements A+B*C-D Push A+B*C-D

Infix to Prefix

1. Reverse the expression
   - Delimiter opening convert to closing and closing to opening.
2. Convert to Postfix
   - But if **operator**, push if operator has higher/**same** precedence with the top, or pop & push if lower.
3. Reverse the output

Evaluation Example: Prefix to Infix




TRAIN YOUR BRAIN

Train your Brain!

- Write an algorithm to convert an infix expression to postfix and infix to prefix
- Write an algorithm to evaluate a postfix and a prefix expression

Referensi

[1] Karumanchi, N. (2017). Data Structures And Algorithms Made Easy (5th ed.). CareerMonk Pub.

[2] Bhargava, A. Y. (2016). Grokking Algorithms. Manning Pub. Co.

[3] Weiss, M. A. (2014). Data Structures and Algorithm Analysis in C++ (4th ed.). Addison-Wesley Pub.

[4] Drozdek, A. (2013). Data Structures and Algorithms in C++ (4th ed.). Cengage Learning.

[5] Gilberg, R. F. & Forouzan, B. A. (2005). Data Structures- A Pseudocode Approach with C (2nd ed.). Thomson Learning, Inc.

[6] Lafore, R. (2003). Data Structures & Algorithms in Java (2nd ed.). Sams Pub.

Fakultas Informatika
School of Computing
Telkom University

THANK YOU