자료구조

Chap 5-1. Stack

2018년 1학기

컴퓨터과학과
민 경 하

Contents

5. Stack

1. Definition of stack
2. Data structure of stack
3. Operations of stack
4. Implementation of operations
5. Applications of stack
6. Evaluation of expressions

1. Definition of stack

- Definition of stack
  - 堆積
  - 彈倉

1. Definition of stack

- Stack
  - A list that records the arrival times of its elements (reverse order)

병원으로 가는 작은 엘리베이터가 있고, 그 앞에 5명이 기다리고 있다. 나는 그 5명 중에서 가장 먼저 접수하기를 원한다. 나는 몇 번째로 이 엘리베이터에 타야 할까?
(1) 첫 번째
(2) 두 번째
(3) 세 번째
(4) 네 번째
(5) 다섯 번째

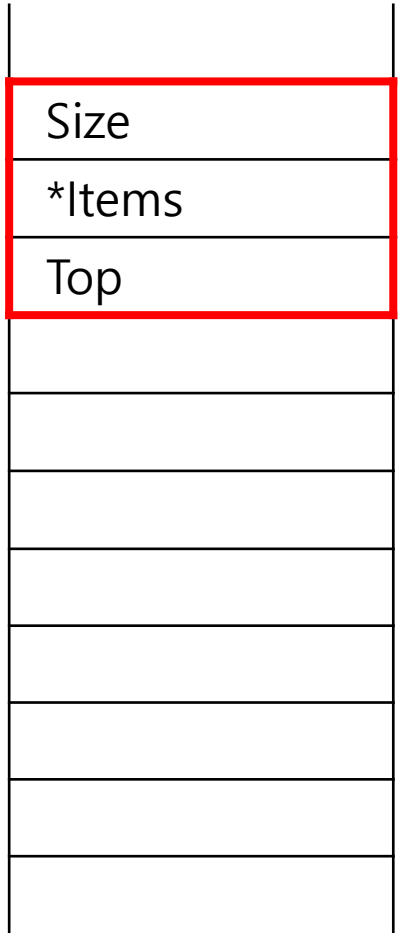
1. Definition of stack

- Stack
  - An ordered list in which insertions and deletions are made at one end called the *top*.
    - Insertion: *push*
    - Deletion: *pop*
  - LIFO (Last-In-First-Out)

2. Data structure of Stack

- Data structure for stack
  - Size
  - List of elements
  - TOP

```
class Stack  {
      int Size;
      DataType *Items;
      int TOP;
};
```



3. Operations of stack

① CreateStack
- Create a stack of size n

② IsEmpty
- Return True, if the stack is empty

③ IsFull
- Return True, if the stack is full

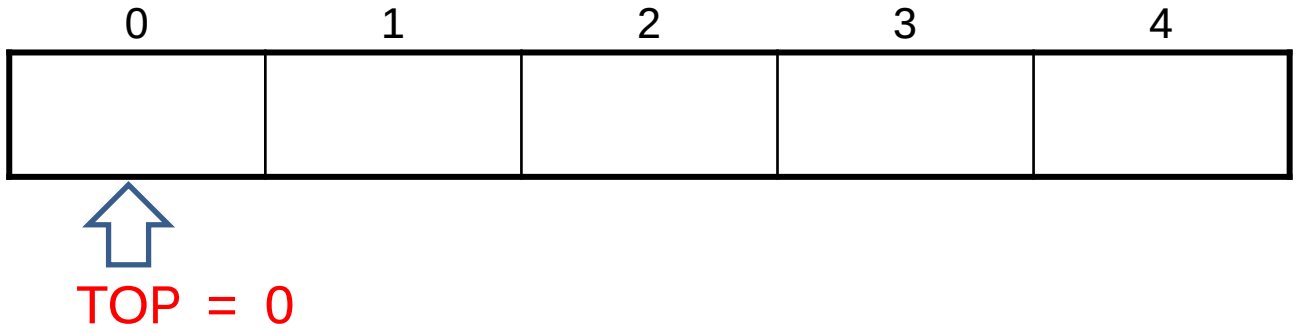
④ Push
- Insert a new element to a stack

⑤ Pop
- Delete an element from a stack

3. Operations of stack

④ Push

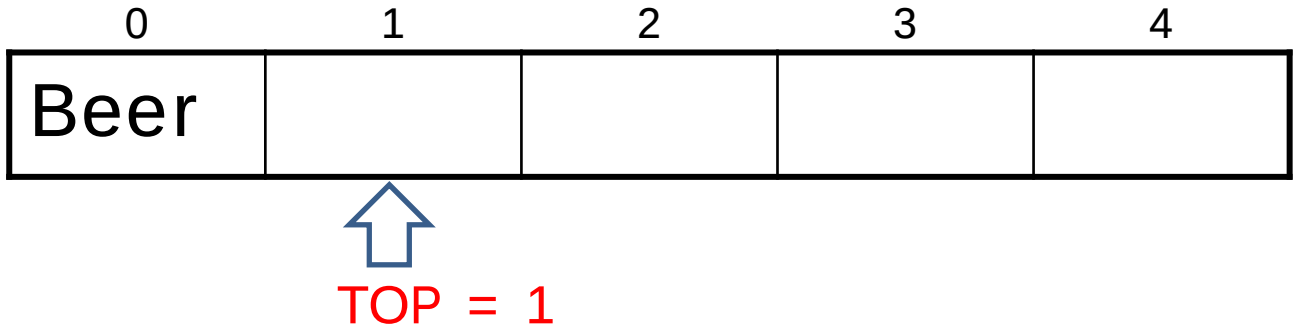
- Insert a new element to a stack
- Push can be executed only at TOP of a stack



3. Operations of stack

④ Push

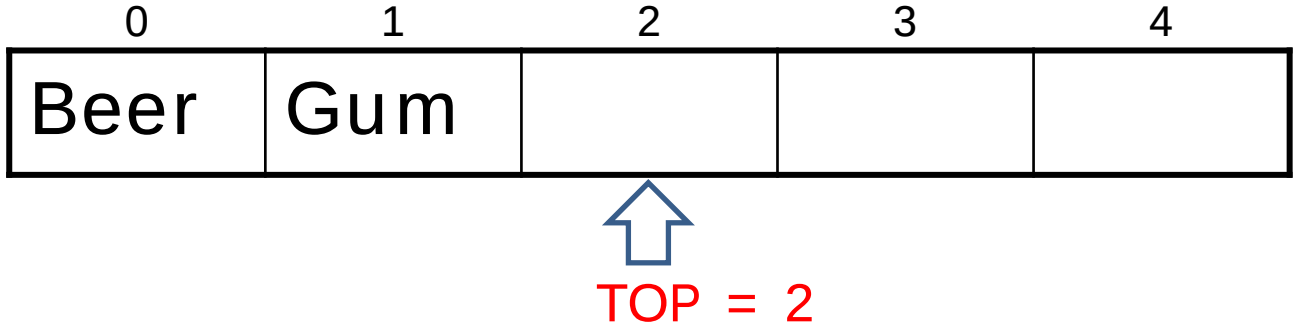
- Insert a new element to a stack
- Push can be executed only at TOP of a stack



3. Operations of stack

④ Push

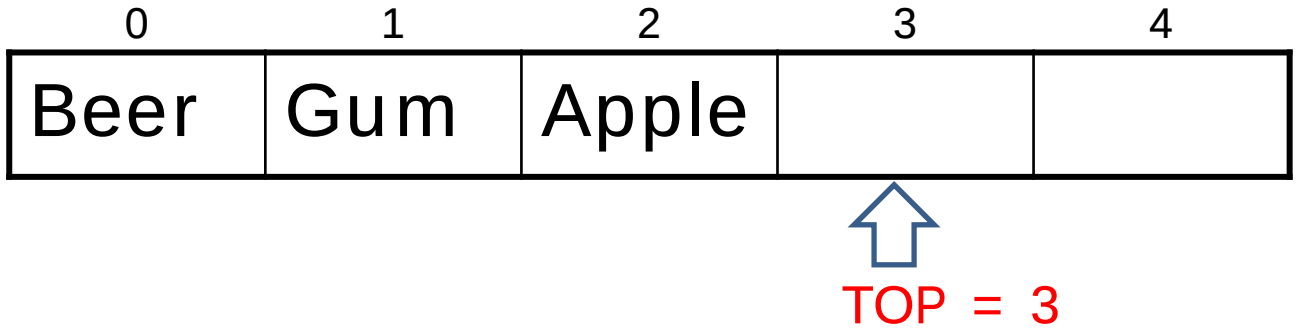
- Insert a new element to a stack
- Push can be executed only at TOP of a stack



3. Operations of stack

④ Push

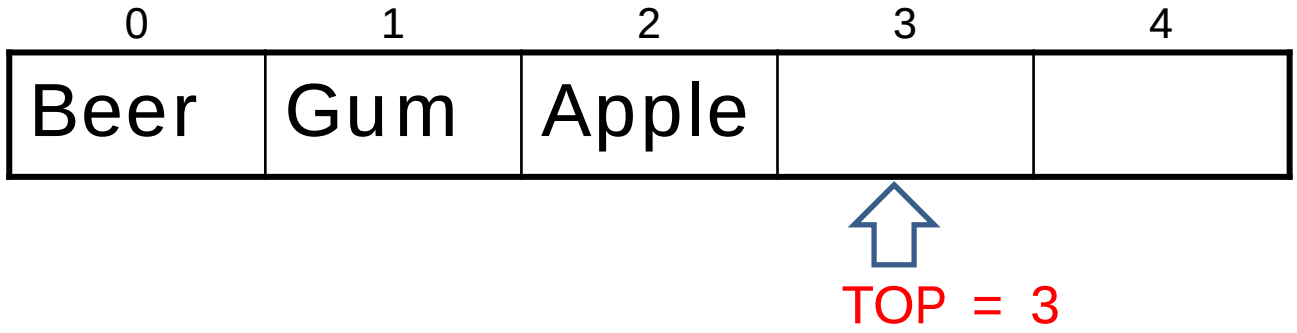
- Insert a new element to a stack
- Push can be executed only at TOP of a stack



3. Operations of stack

⑤ Pop

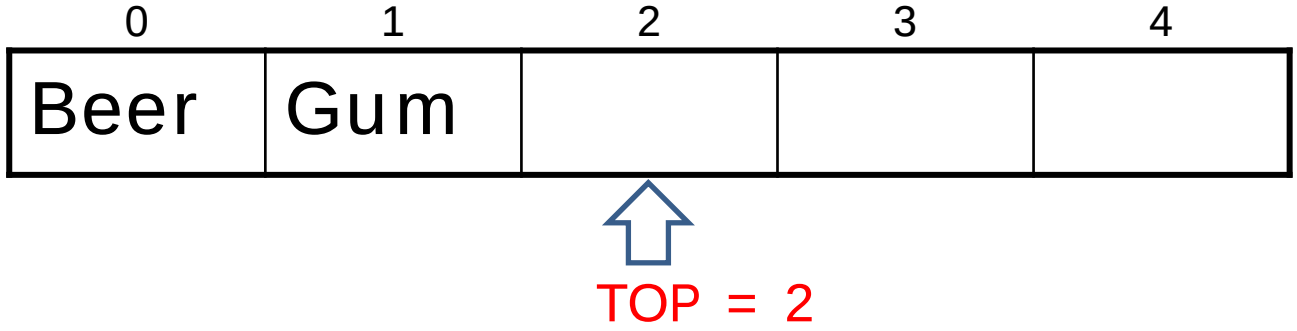
- Delete an element from a stack
- Pop can be executed only at TOP of a stack



3. Operations of stack

⑤ Pop

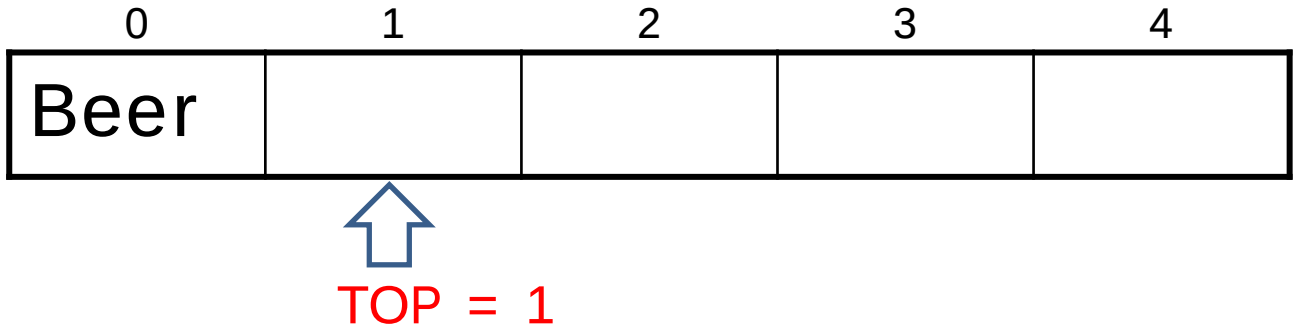
- Delete an element from a stack
- Pop can be executed only at TOP of a stack



3. Operations of stack

⑤ Pop

- Delete an element from a stack
- Pop can be executed only at TOP of a stack



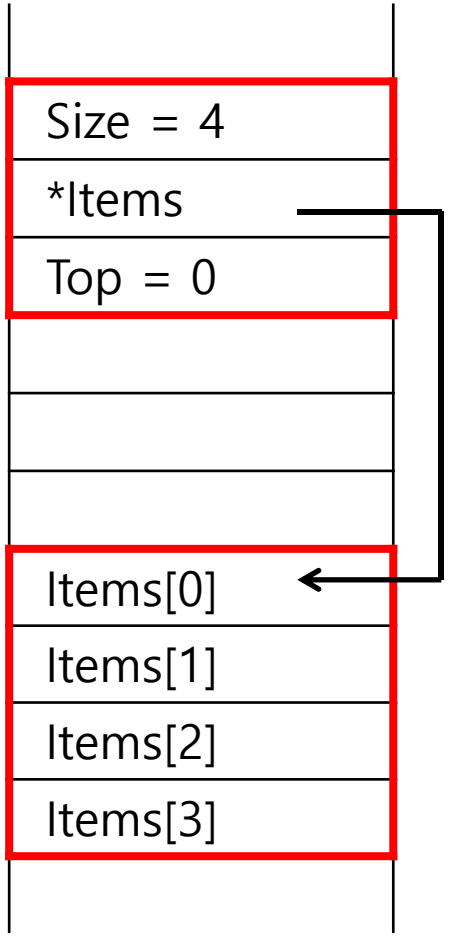
4. Implementation of operations

① CreateS (int n)

- Create a stack of size n

```
void Stack::Create ( int maxStackSize )
{
    Size = maxStackSize ;
    tItem = new Datatype[Size];
    TOP = 0;
}
```

```
void main ( ) {
    Stack myStack.Create ( 4 );
}
```



4. Implementation of operations

② is_Full ( )

- Check overflow
- Returns TRUE if the stack is FULL

```
int Stack::is_Full (  ) {
    return ( TOP == Size );
}
```

4. Implementation of operations

③ is_Empty ( )

- Check underflow
- Returns TRUE if the stack is EMPTY

```
int Stack::is_Empty (  ) {
    return ( TOP == 0 );
}
```

4. Implementation of operations

④ Push ( )

- Add a new element at the TOP of the stack

```
void Stack::push( Datatype DataItem ) {
    Items[TOP] = DataItem;
    TOP++;
}
```

```
main ( ) {
    myStack.push ( “Potato” );
}
```

4. Implementation of operations

④ Push ( )

- Degenerate case?
  - **Overvflow**
  - Cannot push an element to a FULL stack

```
void Stack::push( Datatype DataItem ) {
    if ( is_Full ( ) )
        printf ( "Pushing in Full Stack\n");

    Items[TOP] = DataItem;
    TOP++;
}
```

4. Implementation of operations

⑤ Pop ( )

- Remove the element at TOP of the stack
- Return the removed element

```
Datatype Stack::pop( ) {
    TOP--;
    return Items[TOP];
}
```

```
main ( ) {
    Data = Stack.pop ( );
}
```

4. Implementation of operations

⑤ Pop ( )

- Degenerate case?
  - **Underflow**
  - Cannot pop an element to an EMPTY stack

```
Datatype Stack::pop( ) {
    if ( is_Empty ( ) )
        printf ( "Poping from Empty Stack\n");

    TOP--;
    return Items[TOP];
}
```

4. Implementation of operations

① CreateStack → O(1)

② IsEmpty → (1)

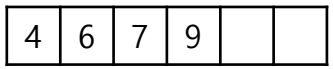
③ IsFull → O(1)

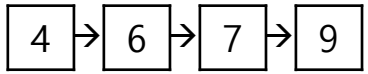
④ Push → O(1)

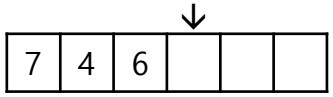
⑤ Pop → O(1)

Overall summary

Type	Data structure	Operations		
		search	**insert**	**delete**

Array (sorted)	`int size;`
	`int n;`
	`int *arr;`
	 
		linear search ( )
		
		binary search ( )
			1. find the location
			2. move to right (→)
			3. insert an element
			4. increase the count
				1. find the location
				2. move to left (←)
				3. reduce the count

Linked list (sorted)	`class node {`
	`  int element;`
	`  node *link;`
	`}`
	 
		linear search ( )	1. find the location
			2. build a new node
			3. change the links
				1. find the location
				2. change the links
				3. free a delete node

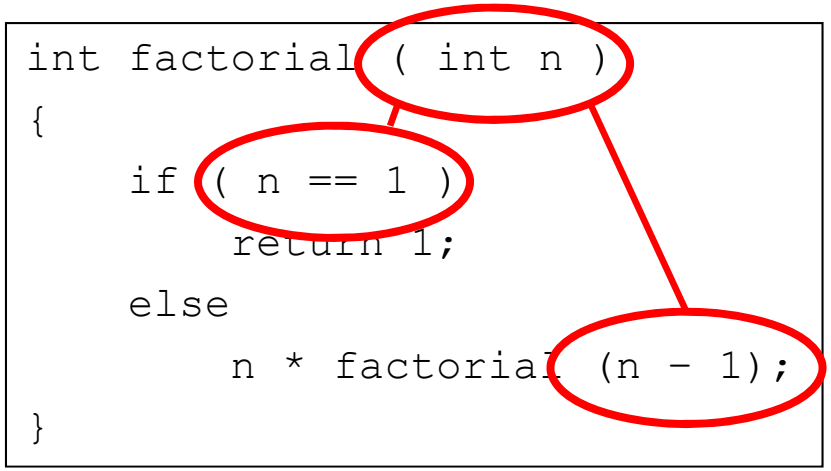
Stack	`Class stack {`
	`  int size;`
	`  int *Items;`
	`  int TOP;`
	`}`
	 
		No search operation	**Push**
			
			Items[TOP++] = item
				Pop
				
				return Items[--Top]

Queue				

5. Applications of stack

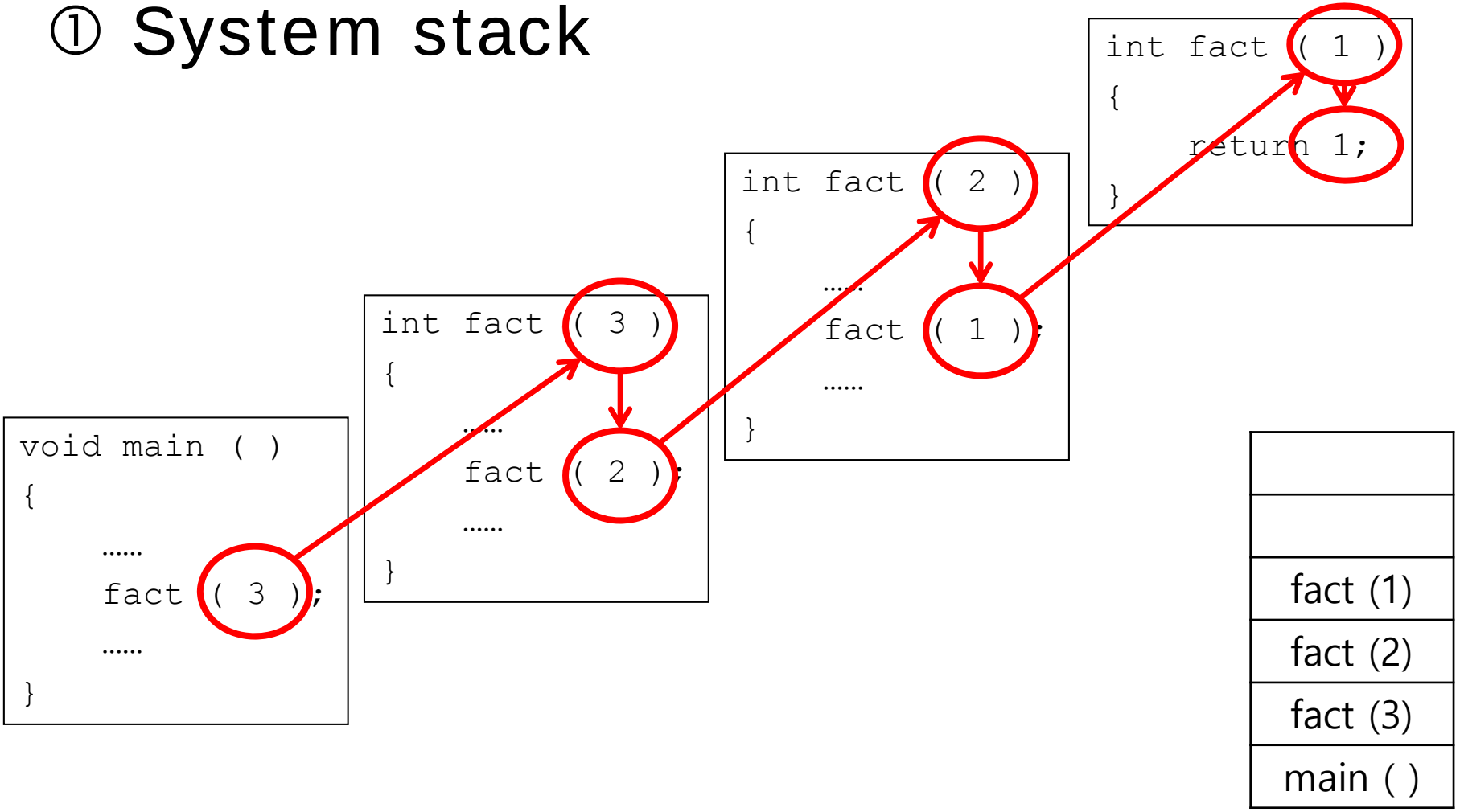
① System stack

- Function call
- Call stack
- Recursive call



5. Applications of stack

① System stack



5. Applications of stack

② Checking parenthesis

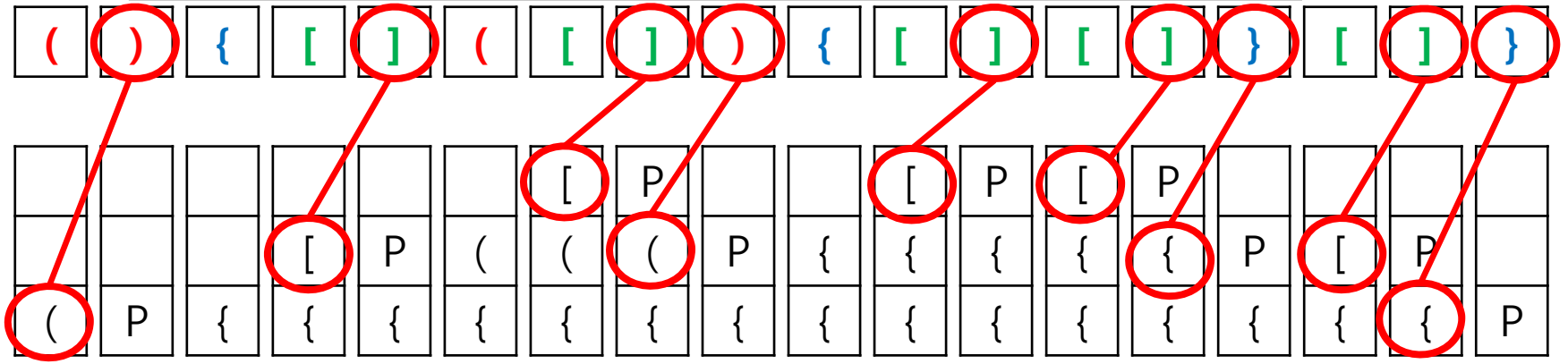
- Parenthesis: ( ) { } [ ]
- Parenthesis matching rule
  - Closing parenthesis follows opening parenthesis
  - Parenthesis of same type matches

```
int factorial ( int n )
{
    int fact[6];

    for ( int i = 2, fact[1] = 1; i <= 5; i++ ) {
        fact[i] = fact[i-1] * i;
    }
    return fact[5];
}
```

5. Applications of stack

② Checking parenthesis

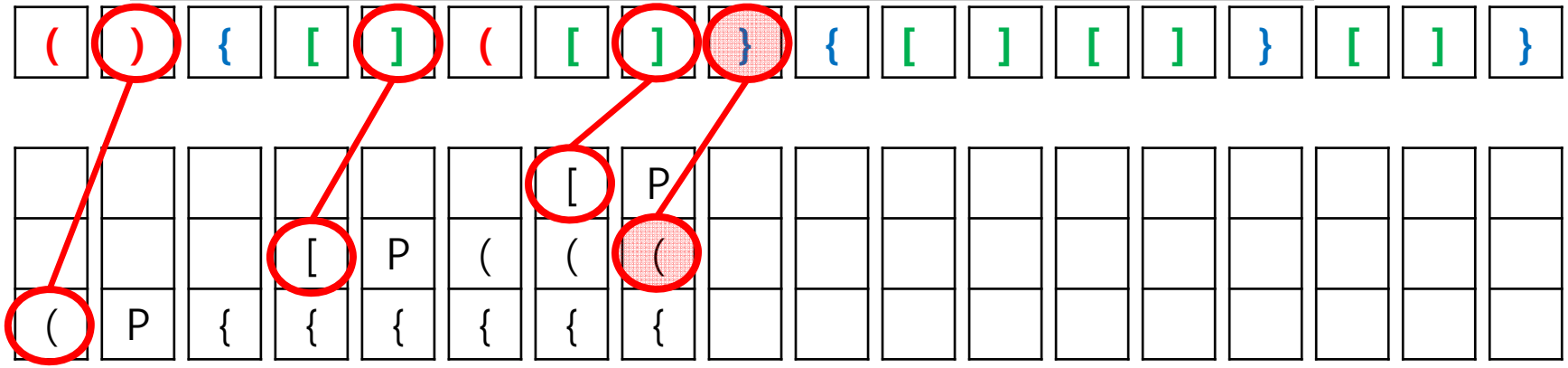
```
int factorial ( int n )
{
    int fact[6];

    for ( int i = 2, fact[1] = 1; i <= 5; i++ ) {
        fact[i] = fact[i-1] * i;
    }
    return fact[5];
}
```



5. Applications of stack

② Checking parenthesis

```
int factorial ( int n )
{
    int fact[6];

    for ( int i = 2, fact[1] = 1; i <= 5; i++ ) {
        fact[i] = fact[i-1] * i;
    }
    return fact[5];
}
```



5. Applications of stack

③ Postfix expression (notation)

- Infix notation
  - a + b * c
  - Problem of infix notation → precedence
  - * has higher precedence than +
  - To override the precedence, we need ( )
  - (a + b)*c != a + b * c

- Postfix notation
  - a + b → a b +
  - b * c → b c *

a + b * c → a b c * +

(a + b) * c → a b + c *

5. Applications of stack

③ Postfix expression (notation)

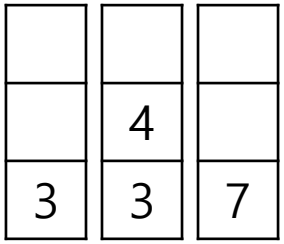
- Postfix evaluation
  - 3 + 4 * 5 → 345*+
  - (3 + 4) * 5 → 34+5*

- Evaluation rule
  - Operand → Push it to a stack;
  - Operator → Pop two times;
    Evaluate it;
    Push to the stack;

5. Applications of stack

③ Postfix expression (notation)

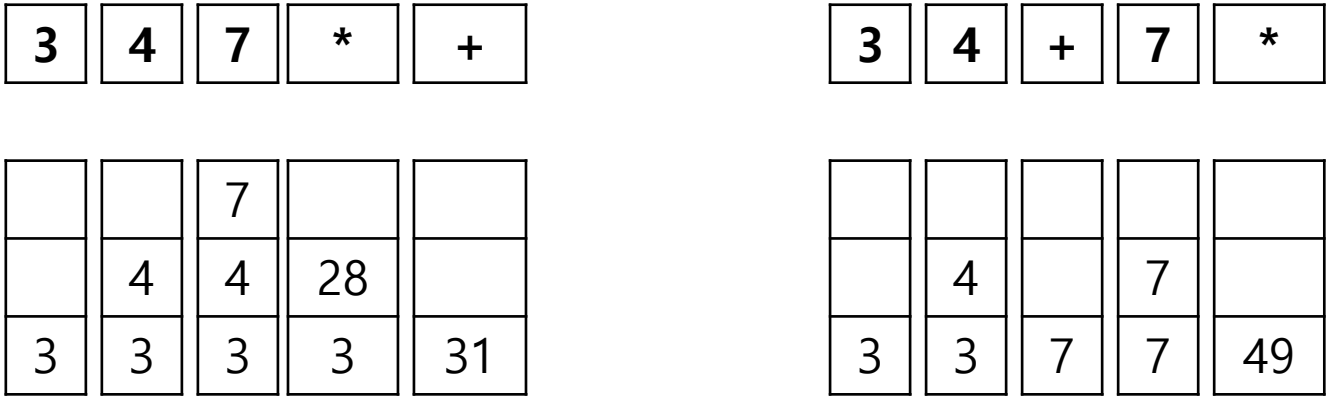
- Example
  - 3 + 4 → 34+



5. Applications of stack

③ Postfix expression (notation)

- Example
  - 3 + 4 * 7 → 347*+
  - (3 + 4) * 7 → 34+7*



5. Applications of stack

④ Maze problem

1	1	1	1	1	1	1	1
0	0	1	1	0	0	0	1
1	0	1	0	0	1	1	1
1	0	1	1	0	1	1	1
1	0	0	0	0	0	0	1
1	1	1	1	1	1	0	1
1	1	1	1	0	0	0	0
1	1	1	1	1	1	1	1

5. Applications of stack

④ Maze problem

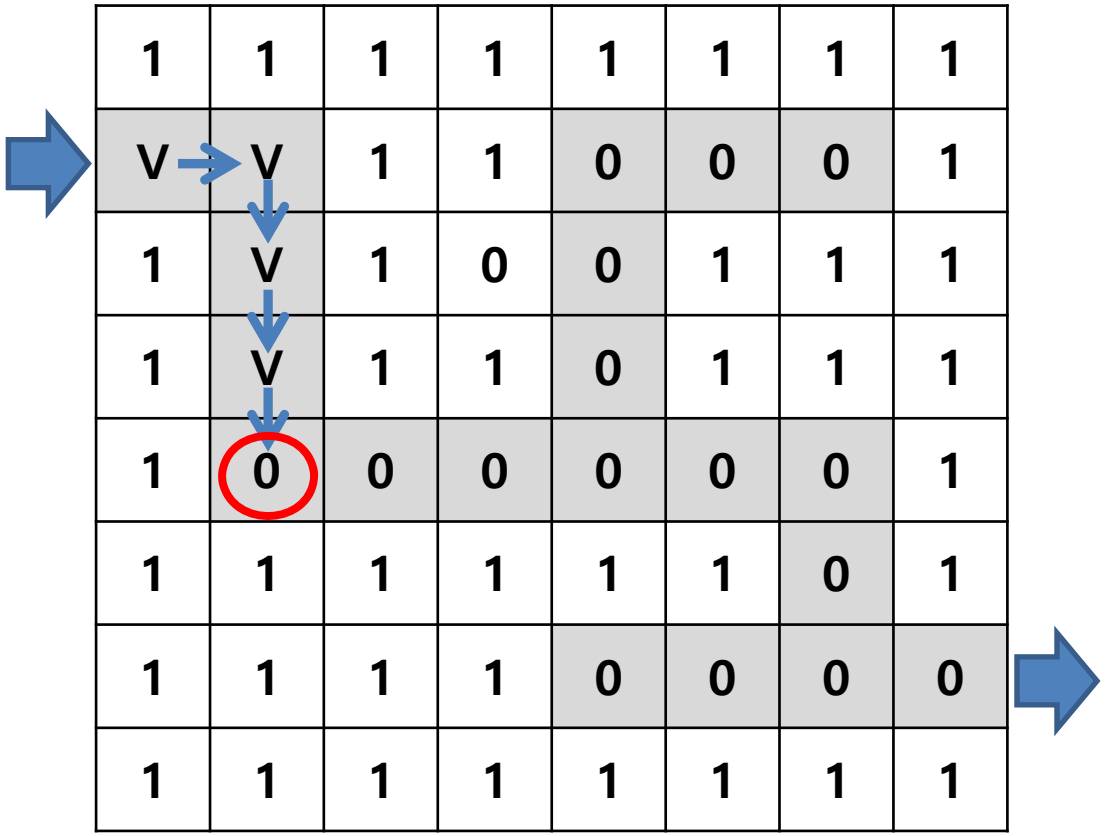
- If you can move
  - Push your position to a stack and move;
- If you cannot move
  - Pop and move to the popped position;

- Mark the cells as
  - V → already visited
  - X → Cannot go further

5. Applications of stack

④ Maze problem

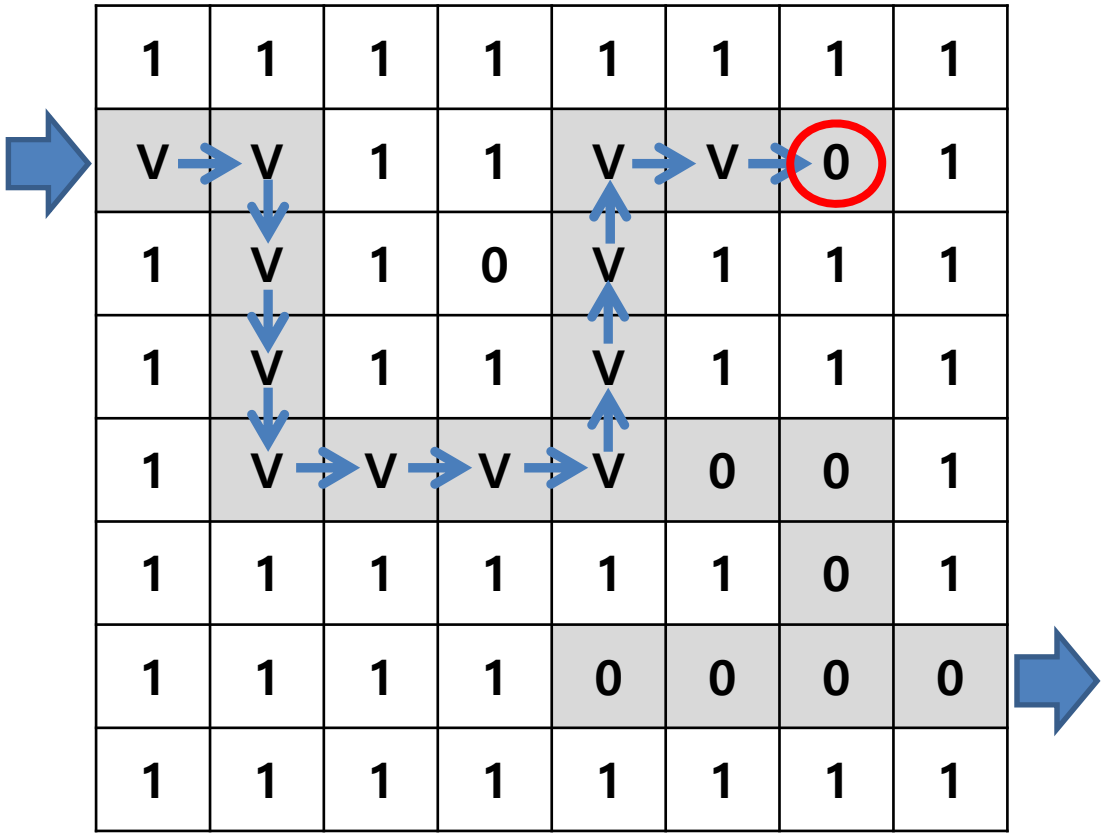
(1,3)
(1,2)
(1,1)
(0,1)



5. Applications of stack

④ Maze problem

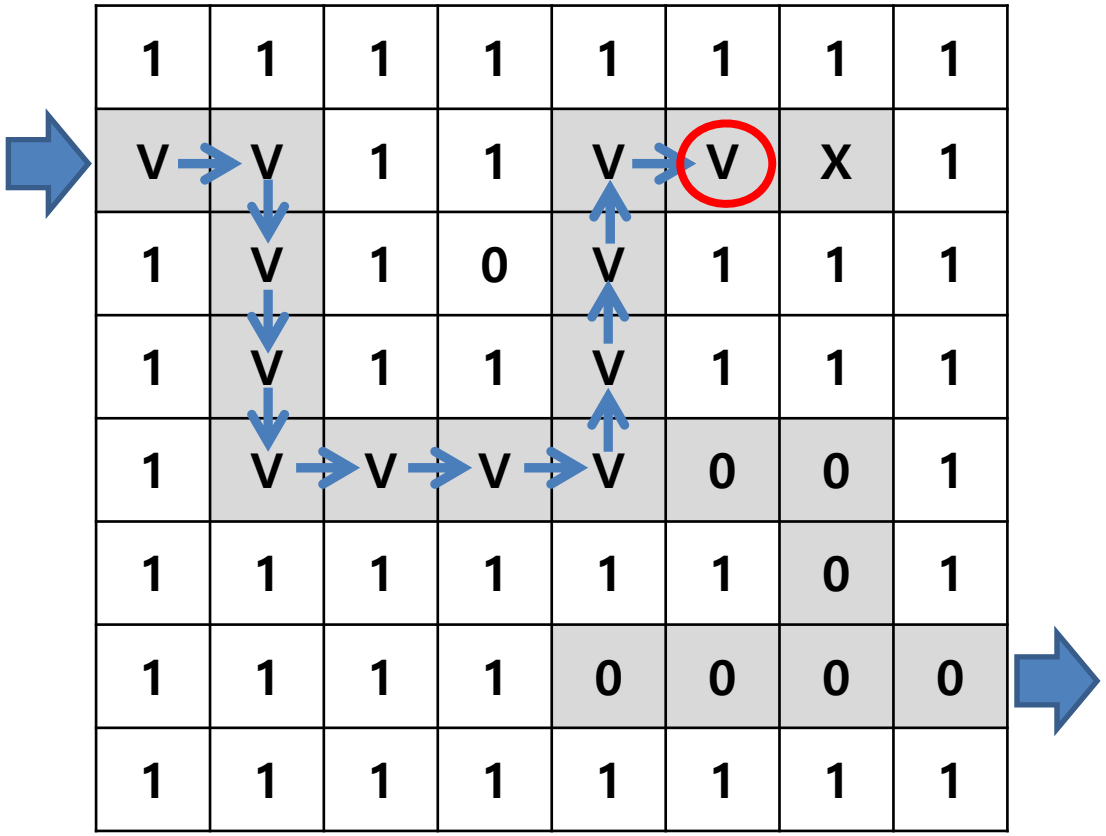
(5,1)
(4,1)
(4,2)
(4,3)
(4,4)
(3,4)
(2,4)
(1,4)
.....
(1,1)
(0,1)



5. Applications of stack

④ Maze problem

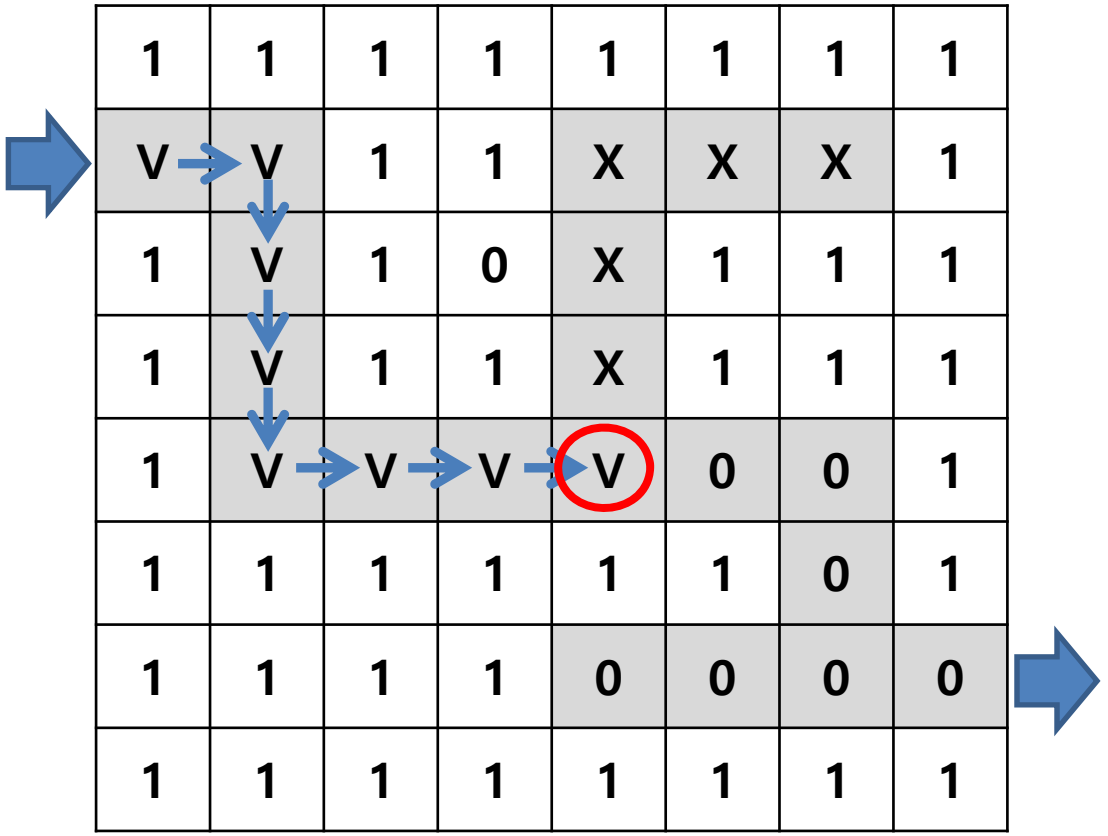
(4,1)
(4,2)
(4,3)
(4,4)
(3,4)
(2,4)
(1,4)
.....
(1,1)
(0,1)



5. Applications of stack

④ Maze problem

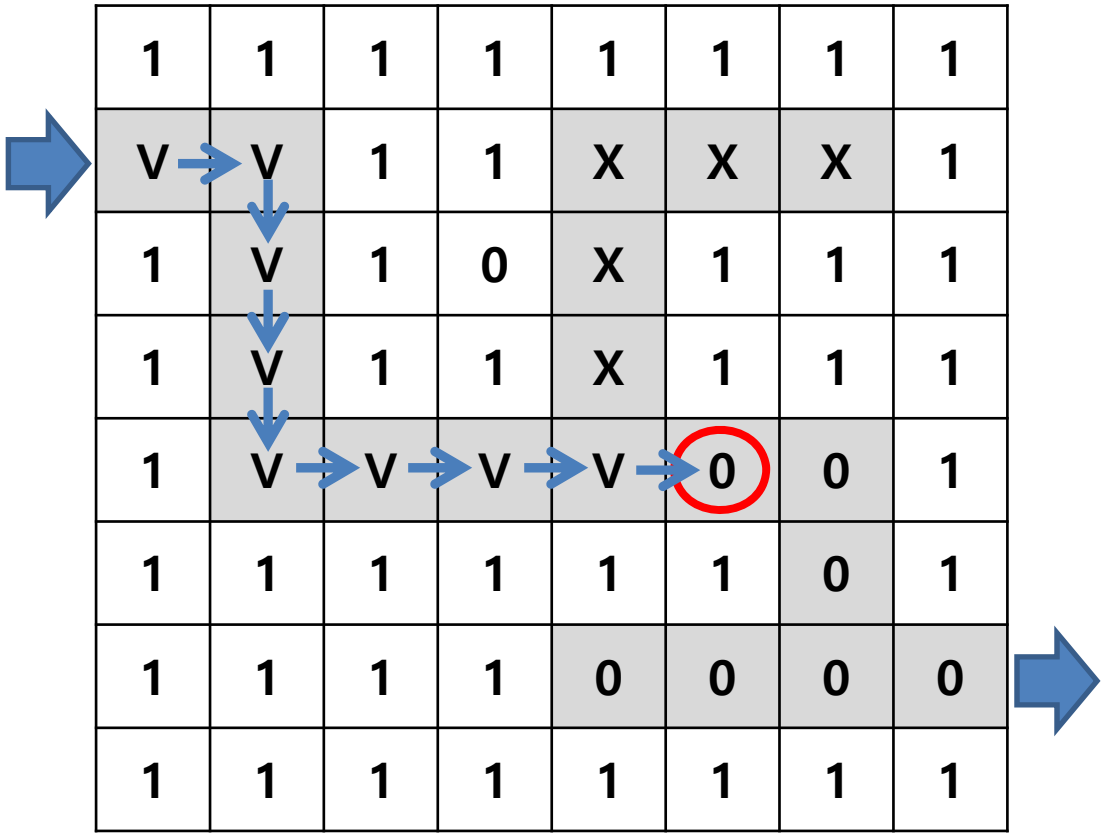
(3,4)
(2,4)
(1,4)
.....
(1,1)
(0,1)



5. Applications of stack

④ Maze problem

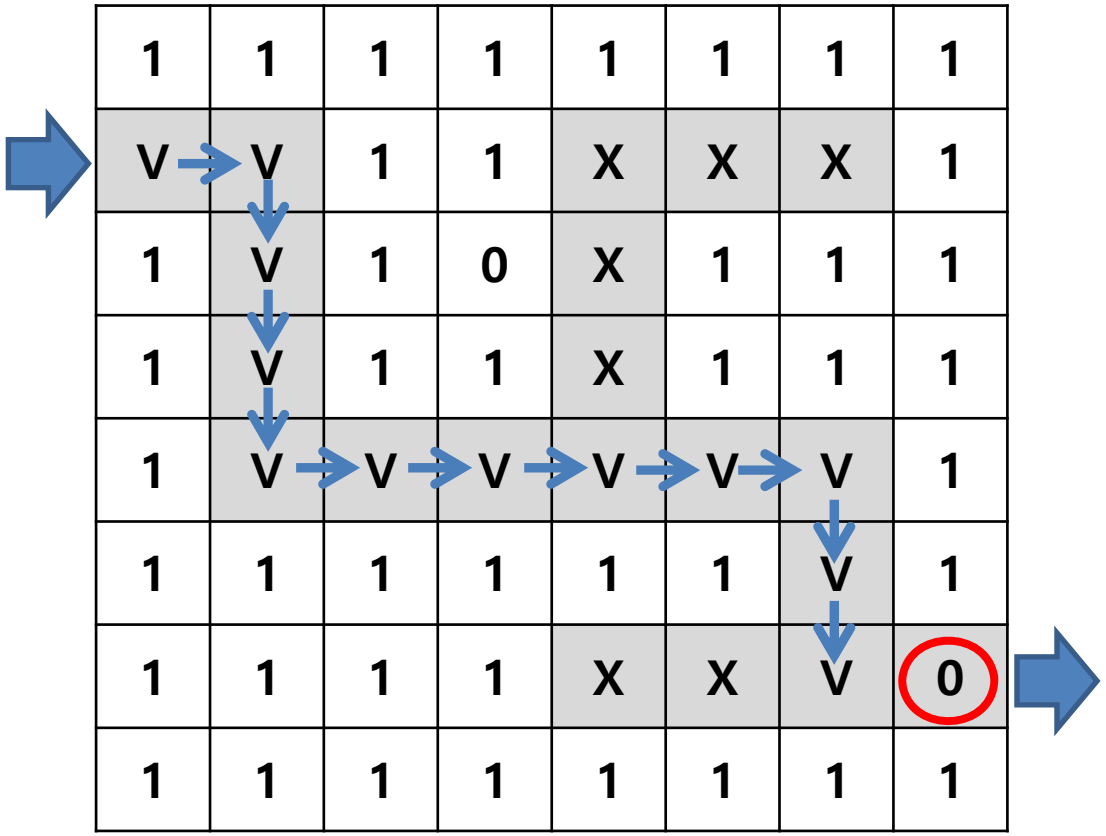
(4,4)
(3,4)
(2,4)
(1,4)
.....
(1,1)
(0,1)



5. Applications of stack

④ Maze problem

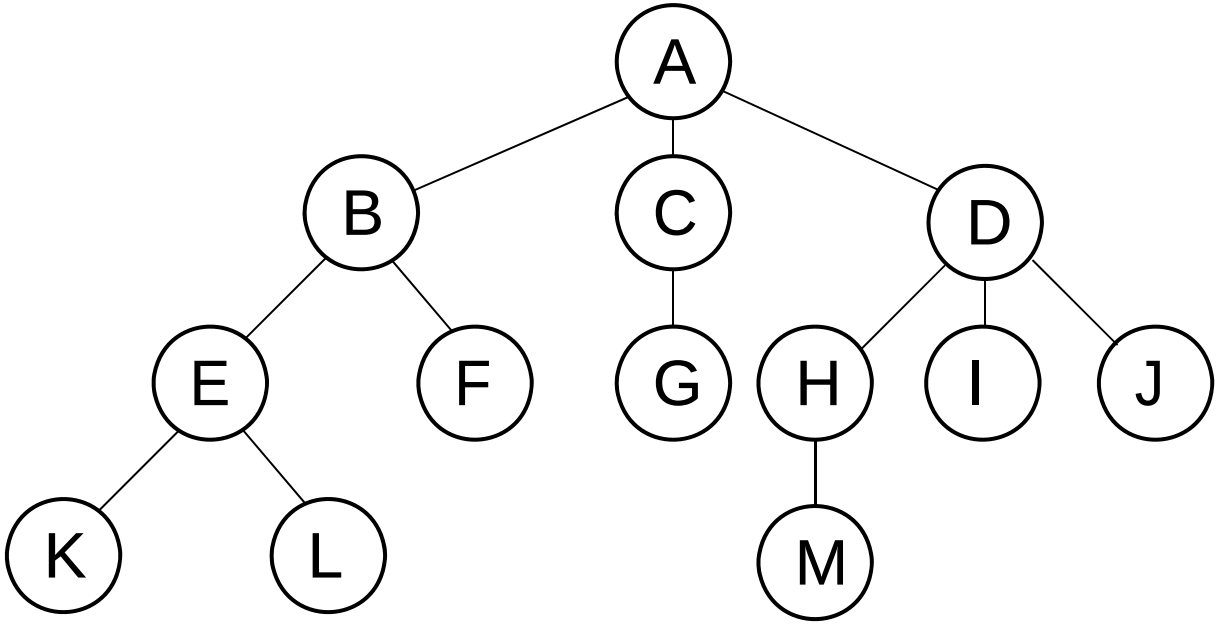
(6,6)
(6,5)
(6,4)
(5,4)
(4,4)
(3,4)
(2,4)
(1,4)
.....
(1,1)
(0,1)



5. Applications of stack

⑤ Graph (tree) search

– Depth first search



6. Evaluation of Expressions

(1) Expressions

(2) Evaluation of expressions

(3) Types of expressions

(4) Evaluation strategy

(5) Translating into postfix

(6) Evaluating a postfix expression

(1) Expressions

- Expression: a formula composed of operators and operands

```
3+4*6-7/2
```

```
x = a/b -c + d/g - a*c;
```

```
((rear+1 == front) || ((rear == MAX_SIZE)
&& !front))
```

  - Operands: 3, 4, x, a, b, front, MAX_SIZE, ...
  - Operators: +, –, *, /, (,), ==, &&, !, ...

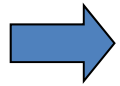
(1) Expressions

- Types of operators
  - **Unary** operator
    - An operator that has one operand
    - OPERATOR OPERAND or OPERAND OPERATOR
    - Example: **–**4, !front, x++
    - **OPERATOR (OPERAND) → OPERAND**
  - **Binary** operator
    - An operator that has two operands
    - OPERAND1 OPERATOR OPERAND2
    - Example: 4 – 5, index != 3, a && b
    - **OPERATOR (OPERAND1, OPERAND2) → OPERAND**

(1) Expressions

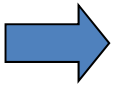
- Evaluation of expressions
  - Finding a single value that is equivalent to the given expression through performing the operations of the expression

```
3+4*6-7/2
```



```
23.5
```

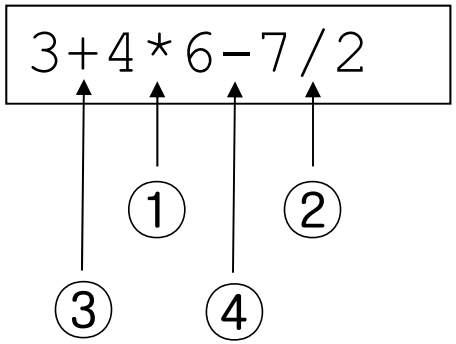
```
((rear+1 == front) || ((rear == MAX_SIZE)
&& !front))
```



```
1
```

(2) Evaluation of expression

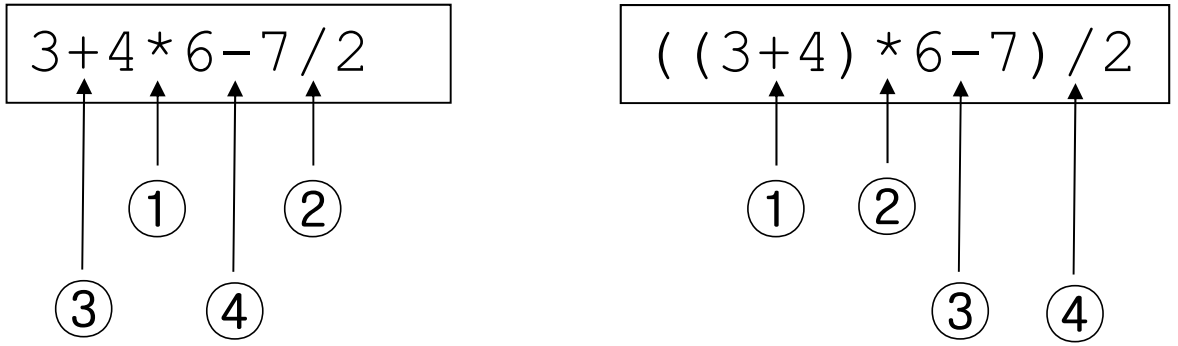
- Problems of evaluation (1)
  - Figure out the order in which the operations are performed → **precedence**



- Precedence in C: Figure 3.12 in Page 130

(2) Evaluation of expression

- Problems of evaluation (2)
  - Overriding the precedence?
    - Use of **parenthesis**
    - The **innermost parenthesis** is evaluated first



(2) Evaluation of expression

- Problems of evaluation (3)
  - The order of evaluations of the operators of same precedence → **associativity** (Figure 3.12 in P130)
    - Left-to-right
    - Right-to-left

(3) Types of expressions

- Three notations of expressions
  - Infix notation
    - OPERAND1 OPERATOR OPERAND2

```
3 + 4
```

  - Prefix notation
    - OPERATOR OPERAND1 OPERAND2

```
+ 3 4
```

  - Postfix notation
    - OPERAND1 OPERAND2 OPERATOR

```
3 4 +
```

(3) Types of expressions

- Usage of parenthesis
  - Infix notation (required)

```
3 + 4 * 5
```

```
(3 + 4) * 5
```

  - Prefix notation (Not required)

```
+ 3 * 4 5
```

```
* + 3 4 5
```

  - Postfix notation (Not required)

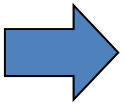
```
3 4 5 * +
```

```
3 4 + 5 *
```

(3) Types of expressions

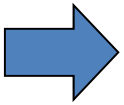
- Examples of postfix notation

```
3 + 4 * 8 – 7 / 2
```



```
3 4 8 * + 7 2 / –
```

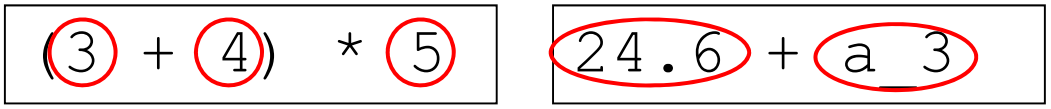
```
((3 + 4) * 8 – 7) / 2
```



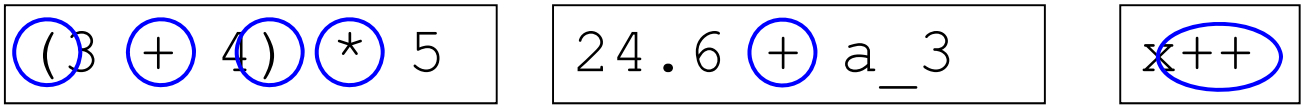
```
3 4 + 8 * 7 – 2 /
```

(4) Evaluation strategy

- Token
  - A unit that can be distinguished from other units
    - Operands

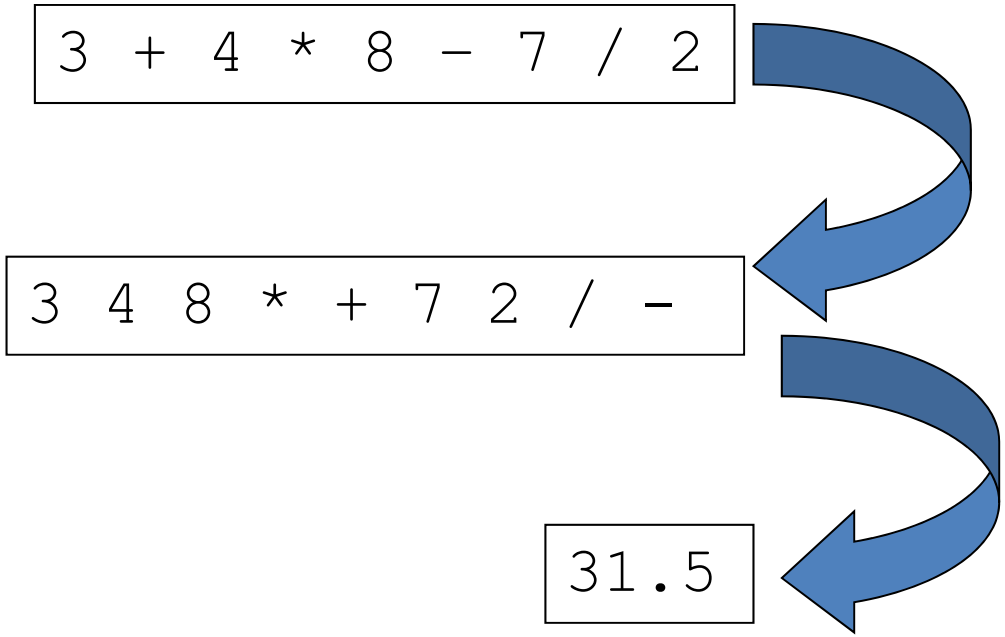


    - Operators



(4) Evaluation strategy

- How to evaluate an expression?



[1] Translating into postfix notation

[2] Evaluating a postfix expression

(5) Translating into postfix

- How to translate an infix notation into a postfix notation?

  ➔ Use a **stack**

  – A simple case: expression without parenthesis

  – More complex case: expression with parenthesis

(5) Translating into postfix

1. Algorithm for simple case:

```
1.  Decompose the expression as a sum of
    tokens
2.  Scan the expression from the first token
   1. If the current token is an OPERAND,
      then print OPERAND;
   2. If the current token is an OPERATOR
      1. While ( stack[TOP] ≥ OPERATOR )
            print POP ( );
      2. PUSH OPERATOR;
3.  If the current token is EOS
   1. While ( !stack.is_empty ( ) )
      print POP ( );
```

(5) Translating into postfix

- Example1: `3 + 4 * 8 – 7 / 2`

Token	Stack			TOP	Output
	[0]	[1]	[2]		
3					
+					
4					
*					
8					
–					
7					
/					
2					
EOS					

(5) Translating into postfix

- Example1: `3 + 4 * 8 – 7 / 2`

Token	Stack			TOP	Output
	[0]	[1]	[2]		
3				-1	3
+					
4					
*					
8					
–					
7					
/					
2					
EOS					

(5) Translating into postfix

- Example1: `3 + 4 * 8 – 7 / 2`

Token	Stack			TOP	Output
	[0]	[1]	[2]		
3				-1	3
+	+			0	3
4					
*					
8					
–					
7					
/					
2					
EOS					

(5) Translating into postfix

- Example1: `3 + 4 * 8 – 7 / 2`

Token	Stack			TOP	Output
	[0]	[1]	[2]		
3				-1	3
+	+			0	3
4	+			0	3 4
*					
8					
–					
7					
/					
2					
EOS					

(5) Translating into postfix

- Example1: `3 + 4 * 8 – 7 / 2`

Token	Stack			TOP	Output
	[0]	[1]	[2]		
3				-1	3
+	+			0	3
4	+			0	3 4
*	+	*		1	3 4
8					
–					
7					
/					
2					
EOS					

(5) Translating into postfix

- Example1: `3 + 4 * 8 – 7 / 2`

Token	Stack			TOP	Output
	[0]	[1]	[2]		
3				-1	3
+	+			0	3
4	+			0	3 4
*	+	*		1	3 4
8	+	*		1	3 4 8
–					
7					
/					
2					
EOS					

(5) Translating into postfix

- Example1: `3 + 4 * 8 – 7 / 2`

Token	Stack			TOP	Output
	[0]	[1]	[2]		
3				-1	3
+	+			0	3
4	+			0	3 4
*	+	*		1	3 4
8	+	*		1	3 4 8
–				0	3 4 8 * +
7					
/					
2					
EOS					

(5) Translating into postfix

- Example1: `3 + 4 * 8 – 7 / 2`

Token	Stack			TOP	Output
	[0]	[1]	[2]		
3				-1	3
+	+			0	3
4	+			0	3 4
*	+	*		1	3 4
8	+	*		1	3 4 8
–	–			0	3 4 8 * +
7					
/					
2					
EOS					

(5) Translating into postfix

- Example1: `3 + 4 * 8 – 7 / 2`

Token	Stack			TOP	Output
	[0]	[1]	[2]		
3				-1	3
+	+			0	3
4	+			0	3 4
*	+	*		1	3 4
8	+	*		1	3 4 8
–	–			0	3 4 8 * +
7	–			0	3 4 8 * + 7
/					
2					
EOS					

(5) Translating into postfix

- Example1: `3 + 4 * 8 – 7 / 2`

Token	Stack			TOP	Output
	[0]	[1]	[2]		
3				-1	3
+	+			0	3
4	+			0	3 4
*	+	*		1	3 4
8	+	*		1	3 4 8
–	–			0	3 4 8 * +
7	–			0	3 4 8 * + 7
/	–	/		1	3 4 8 * + 7
2					
EOS					

(5) Translating into postfix

- Example1: `3 + 4 * 8 – 7 / 2`

Token	Stack			TOP	Output
	[0]	[1]	[2]		
3				-1	3
+	+			0	3
4	+			0	3 4
*	+	*		1	3 4
8	+	*		1	3 4 8
–	–			0	3 4 8 * +
7	–			0	3 4 8 * + 7
/	–	/		1	3 4 8 * + 7
2	–	/		1	3 4 8 * + 7 2
EOS					

(5) Translating into postfix

- Example1: `3 + 4 * 8 – 7 / 2`

Token	Stack			TOP	Output
	[0]	[1]	[2]		
3				-1	3
+	+			0	3
4	+			0	3 4
*	+	*		1	3 4
8	+	*		1	3 4 8
–	–			0	3 4 8 * +
7	–			0	3 4 8 * + 7
/	–	/		1	3 4 8 * + 7
2	–	/		1	3 4 8 * + 7 2
EOS				-1	3 4 8 * + 7 2 / –

(5) Translating into postfix

2. Algorithm for more complex case

- Expression with parenthesis
- Innermost parenthesis is processed first.

```
2. If the current token is an OPERATOR
   1. If ( token == Right parenthesis )
      1.1 While ( stack[TOP] != Left parenthesis )
          Print POP ( );
      1.2 POP ( );
   2. While ( stack[TOP] ≥ OPERATOR )
         print POP ( );
   3. PUSH OPERATOR;
```

(5) Translating into postfix

- Example2: `((3 + 4) * 8 – 7) / 2`

Token	Stack					TOP	Output
	[0]	[1]	[2]	[3]	[4]		
(							
(							
3							
+							
4							
)							
*							
8							
–							
7							

(5) Translating into postfix

- Example2: `((3 + 4) * 8 – 7) / 2`

Token	Stack					TOP	Output
	[0]	[1]	[2]	[3]	[4]		
(	(					0	
(							
3							
+							
4							
)							
*							
8							
–							
7							

(5) Translating into postfix

- Example2: `((3 + 4) * 8 – 7) / 2`

Token	Stack					TOP	Output
	[0]	[1]	[2]	[3]	[4]		
(	(					0	
(	(	(				1	
3							
+							
4							
)							
*							
8							
–							
7							

(5) Translating into postfix

- Example2: `((3 + 4) * 8 – 7) / 2`

Token	Stack					TOP	Output
	[0]	[1]	[2]	[3]	[4]		
(	(					0	
(	(	(				1	
3	(	(				1	3
+							
4							
)							
*							
8							
–							
7							

(5) Translating into postfix

- Example2: `((3 + 4) * 8 – 7) / 2`

Token	Stack					TOP	Output
	[0]	[1]	[2]	[3]	[4]		
(	(					0	
(	(	(				1	
3	(	(				1	3
+	(	(	+			2	3
4							
)							
*							
8							
–							
7							

(5) Translating into postfix

- Example2: `((3 + 4) * 8 – 7) / 2`

Token	Stack					TOP	Output
	[0]	[1]	[2]	[3]	[4]		
(	(					0	
(	(	(				1	
3	(	(				1	3
+	(	(	+			2	3
4	(	(	+			2	3 4
)							
*							
8							
–							
7							

(5) Translating into postfix

- Example2: `((3 + 4) * 8 – 7) / 2`

Token	Stack					TOP	Output
	[0]	[1]	[2]	[3]	[4]		
(	(					0	
(	(	(				1	
3	(	(				1	3
+	(	(	+			2	3
4	(	(	+			2	3 4
)	(					0	3 4 +
*							
8							
–							
7							

(5) Translating into postfix

- Example2: `((3 + 4) * 8 – 7) / 2`

Token	Stack					TOP	Output
	[0]	[1]	[2]	[3]	[4]		
(	(					0	
(	(	(				1	
3	(	(				1	3
+	(	(	+			2	3
4	(	(	+			2	3 4
)	(					0	3 4 +
*	(	*				1	3 4 +
8							
–							
7							

(5) Translating into postfix

- Example2: `((3 + 4) * 8 – 7) / 2`

Token	Stack					TOP	Output
	[0]	[1]	[2]	[3]	[4]		
(	(					0	
(	(	(				1	
3	(	(				1	3
+	(	(	+			2	3
4	(	(	+			2	3 4
)	(					0	3 4 +
*	(	*				1	3 4 +
8	(	*				1	3 4 + 8
–							
7							

(5) Translating into postfix

- Example2: `((3 + 4) * 8 – 7) / 2`

Token	Stack					TOP	Output
	[0]	[1]	[2]	[3]	[4]		
(	(					0	
(	(	(				1	
3	(	(				1	3
+	(	(	+			2	3
4	(	(	+			2	3 4
)	(					0	3 4 +
*	(	*				1	3 4 +
8	(	*				1	3 4 + 8
–	(	–				1	3 4 + 8 *
7							

(5) Translating into postfix

- Example2: `((3 + 4) * 8 – 7) / 2`

Token	Stack					TOP	Output
	[0]	[1]	[2]	[3]	[4]		
(	(					0	
(	(	(				1	
3	(	(				1	3
+	(	(	+			2	3
4	(	(	+			2	3 4
)	(					0	3 4 +
*	(	*				1	3 4 +
8	(	*				1	3 4 + 8
–	(	–				1	3 4 + 8 *
7	(	–				1	3 4 + 8 * 7

(5) Translating into postfix

- Example2: `((3 + 4) * 8 – 7) / 2`

Token	Stack					TOP	Output
	[0]	[1]	[2]	[3]	[4]		
)						-1	3 4 + 8 * 7 –
/							
2							
EOS							

(5) Translating into postfix

- Example2: `((3 + 4) * 8 – 7) / 2`

Token	Stack					TOP	Output
	[0]	[1]	[2]	[3]	[4]		
)						-1	3 4 + 8 * 7 –
/	/					0	3 4 + 8 * 7 –
2							
EOS							

(5) Translating into postfix

- Example2: `((3 + 4) * 8 – 7) / 2`

Token	Stack					TOP	Output
	[0]	[1]	[2]	[3]	[4]		
)						-1	3 4 + 8 * 7 –
/	/					0	3 4 + 8 * 7 –
2	/					0	3 4 + 8 * 7 – 2
EOS							

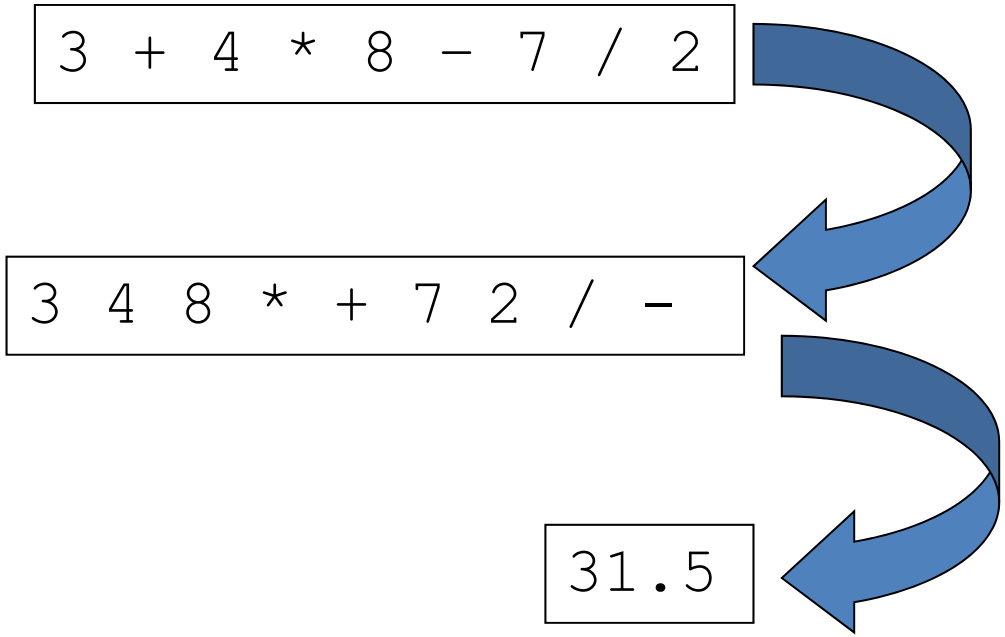
(5) Translating into postfix

- Example2: `((3 + 4) * 8 – 7) / 2`

Token	Stack					TOP	Output
	[0]	[1]	[2]	[3]	[4]		
)						-1	3 4 + 8 * 7 –
/	/					0	3 4 + 8 * 7 –
2	/					0	3 4 + 8 * 7 – 2
EOS						-1	3 4 + 8 * 7 – 2 /

(6) Evaluating a postfix expression

- How to evaluate an expression?



[1] Translating into postfix notation

[2] Evaluating a postfix expression

(6) Evaluating a postfix expression

- Evaluation of postfix expression ➔stack!!

1. Decompose the expression as a sum of tokens
2. Scan the expression from the first token
    1. If the current token is an operand, then PUSH it
    2. If the current token is an operator
        1. If it is a binary operator, POP two elements and PUSH the result
        2. If it is a unary operator, POP one element and PUSH the result
3. Repeat this until all the tokens are processed

(6) Evaluating a postfix expression

- The simplest example
  - Operands: a single digit integer
    - 0 ~ 9
  - Operators: arithmetic without parenthesis
    - +, –, *, /, %
  - Program 3.13 @P133

(6) Evaluating a postfix expression

- Example1: `3 4 8 * + 7 2 / -`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3							
4							
8							
*							
+							
7							
2							
/							
-							

(6) Evaluating a postfix expression

- Example1: `3 4 8 * + 7 2 / -`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4							
8							
*							
+							
7							
2							
/							
-							

(6) Evaluating a postfix expression

- Example1: `3 4 8 * + 7 2 / -`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
8							
*							
+							
7							
2							
/							
-							

(6) Evaluating a postfix expression

- Example1: `3 4 8 * + 7 2 / -`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
8	3	4	8	2			
*							
+							
7							
2							
/							
-							

(6) Evaluating a postfix expression

- Example1: `3 4 8 * + 7 2 / -`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
8	3	4	8	2			
*	3			1	4	8	32
+							
7							
2							
/							
-							

(6) Evaluating a postfix expression

- Example1: `3 4 8 * + 7 2 / -`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
8	3	4	8	2			
*	3	32		1	4	8	32
+							
7							
2							
/							
-							

(6) Evaluating a postfix expression

- Example1: `3 4 8 * + 7 2 / -`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
8	3	4	8	2			
*	3	32		1	4	8	32
+				0	3	32	35
7							
2							
/							
-							

(6) Evaluating a postfix expression

- Example1: `3 4 8 * + 7 2 / -`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
8	3	4	8	2			
*	3	32		1	4	8	32
+	35			0	3	32	35
7							
2							
/							
-							

(6) Evaluating a postfix expression

- Example1: `3 4 8 * + 7 2 / -`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
8	3	4	8	2			
*	3	32		1	4	8	32
+	35			0	3	32	35
7	35	7		1			
2							
/							
-							

(6) Evaluating a postfix expression

- Example1: `3 4 8 * + 7 2 / -`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
8	3	4	8	2			
*	3	32		1	4	8	32
+	35			0	3	32	35
7	35	7		1			
2	35	7	2	2			
/							
-							

(6) Evaluating a postfix expression

- Example1: `3 4 8 * + 7 2 / -`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
8	3	4	8	2			
*	3	32		1	4	8	32
+	35			0	3	32	35
7	35	7		1			
2	35	7	2	2			
/	35			1	7	2	3.5
-							

(6) Evaluating a postfix expression

- Example1: `3 4 8 * + 7 2 / -`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
8	3	4	8	2			
*	3	32		1	4	8	32
+	35			0	3	32	35
7	35	7		1			
2	35	7	2	2			
/	35	3.5		1	7	2	3.5
-							

(6) Evaluating a postfix expression

- Example1: `3 4 8 * + 7 2 / -`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
8	3	4	8	2			
*	3	32		1	4	8	32
+	35			0	3	32	35
7	35	7		1			
2	35	7	2	2			
/	35	3.5		1	7	2	3.5
-				0	35	3.5	31.5

(6) Evaluating a postfix expression

- Example1: `3 4 8 * + 7 2 / -`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
8	3	4	8	2			
*	3	32		1	4	8	32
+	35			0	3	32	35
7	35	7		1			
2	35	7	2	2			
/	35	3.5		1	7	2	3.5
-	31.5			0	35	3.5	31.5

(6) Evaluating a postfix expression

- Example2: `3 4 + 8 * 7 - 2 /`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3							
4							
+							
8							
*							
7							
-							
2							
/							

(6) Evaluating a postfix expression

- Example2: `3 4 + 8 * 7 - 2 /`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4							
+							
8							
*							
7							
-							
2							
/							

(6) Evaluating a postfix expression

- Example2: `3 4 + 8 * 7 - 2 /`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
+							
8							
*							
7							
-							
2							
/							

(6) Evaluating a postfix expression

- Example2: `3 4 + 8 * 7 - 2 /`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
+				0	3	4	7
8							
*							
7							
-							
2							
/							

(6) Evaluating a postfix expression

- Example2: `3 4 + 8 * 7 - 2 /`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
+	7			0	3	4	7
8							
*							
7							
-							
2							
/							

(6) Evaluating a postfix expression

- Example2: `3 4 + 8 * 7 - 2 /`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
+	7			0	3	4	7
8	7	8		1			
*							
7							
-							
2							
/							

(6) Evaluating a postfix expression

- Example2: `3 4 + 8 * 7 - 2 /`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
+	7			0	3	4	7
8	7	8		1			
*				0	7	8	56
7							
-							
2							
/							

(6) Evaluating a postfix expression

- Example2: `3 4 + 8 * 7 - 2 /`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
+	7			0	3	4	7
8	7	8		1			
*	56			0	7	8	56
7							
-							
2							
/							

(6) Evaluating a postfix expression

- Example2: `3 4 + 8 * 7 - 2 /`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
+	7			0	3	4	7
8	7	8		1			
*	56			0	7	8	56
7	56	7		1			
-							
2							
/							

(6) Evaluating a postfix expression

- Example2: `3 4 + 8 * 7 - 2 /`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
+	7			0	3	4	7
8	7	8		1			
*	56			0	7	8	56
7	56	7		1			
-				0	56	7	49
2							
/							

(6) Evaluating a postfix expression

- Example2: `3 4 + 8 * 7 - 2 /`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
+	7			0	3	4	7
8	7	8		1			
*	56			0	7	8	56
7	56	7		1			
-	49			0	56	7	49
2							
/							

(6) Evaluating a postfix expression

- Example2: `3 4 + 8 * 7 - 2 /`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
+	7			0	3	4	7
8	7	8		1			
*	56			0	7	8	56
7	56	7		1			
-	49			0	56	7	49
2	49	2		1			
/							

(6) Evaluating a postfix expression

- Example2: `3 4 + 8 * 7 - 2 /`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
+	7			0	3	4	7
8	7	8		1			
*	56			0	7	8	56
7	56	7		1			
-	49			0	56	7	49
2	49	2		1			
/				0	49	2	24.5

(6) Evaluating a postfix expression

- Example2: `3 4 + 8 * 7 - 2 /`

Token	Stack			Top	Op1	Op2	Result
	[0]	[1]	[2]				
3	3			0			
4	3	4		1			
+	7			0	3	4	7
8	7	8		1			
*	56			0	7	8	56
7	56	7		1			
-	49			0	56	7	49
2	49	2		1			
/	24.5			0	49	2	24.5

Contents