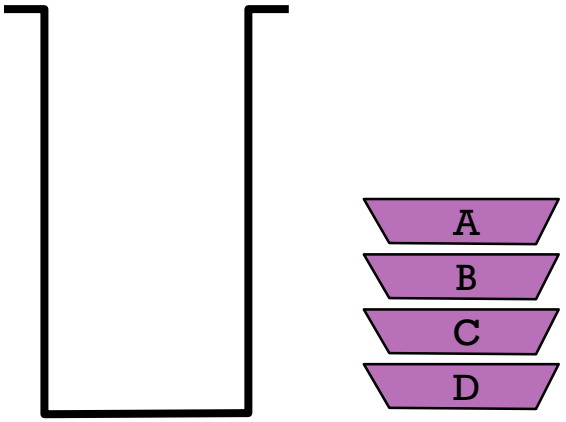
Stack

What is Stack?

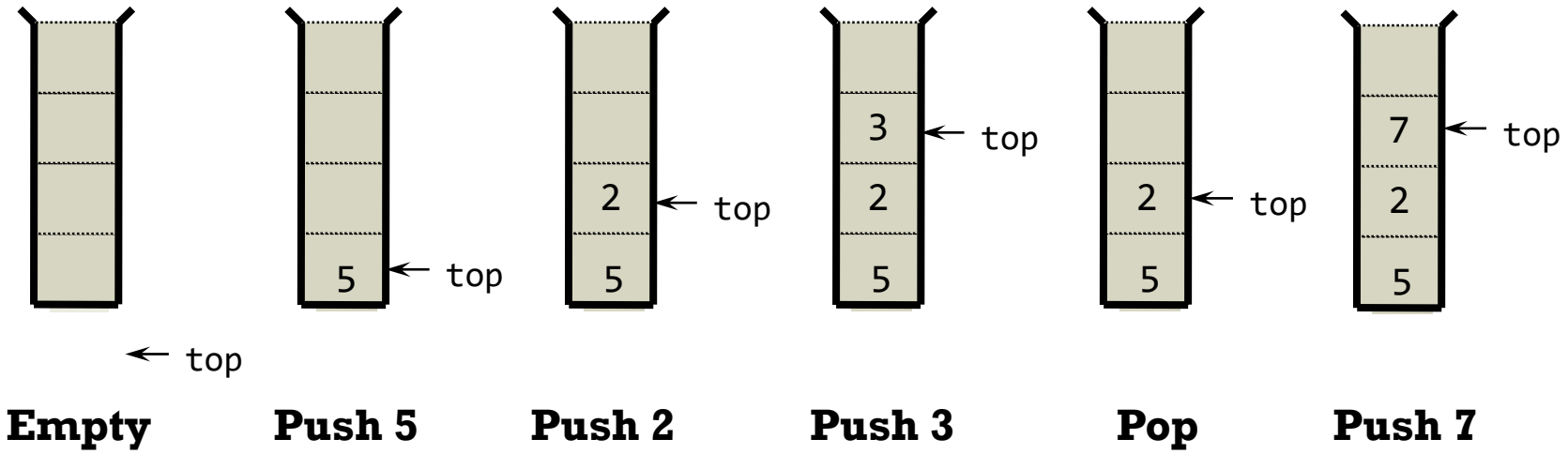
- Definition
  - A collection of elements that are inserted and removed according to the **last-in first-out (LIFO) principle**.
    - The last element will be the first element to be removed.
    - Input and output are only possible at the top on the stack.



What is Stack?

- Terminology
  - **Top**: The top of stack (default = -1)
  - **Push**: Insert an item on the top.
  - **Pop**: Remove the item on the top.

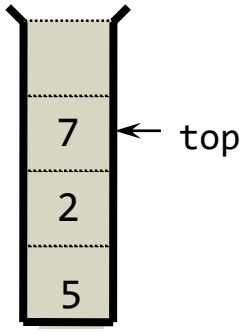
- How does the stack work?



What is Stack?

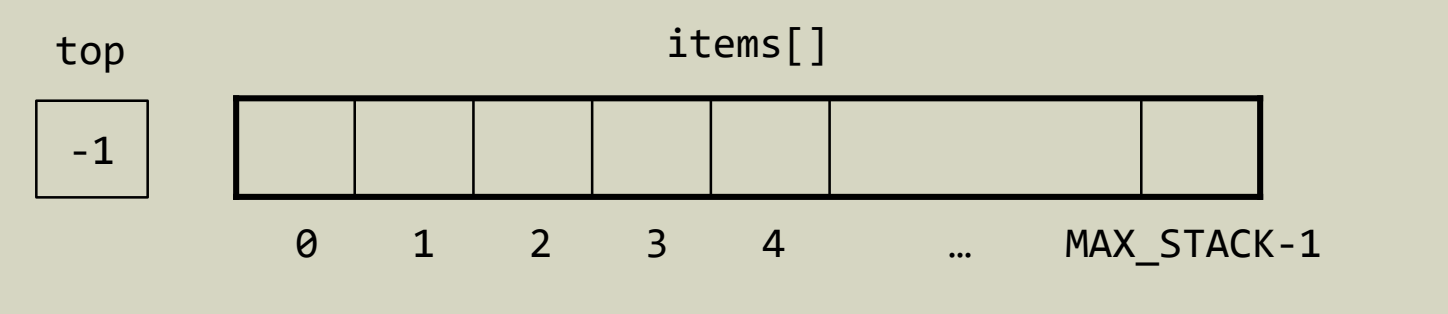
- Operations
  - **InitStack**: Make stack empty.
  - **IsFull**: Check whether stack is full.
  - **IsEmpty**: Check whether stack is empty.
  - **Peek**: Read the item at the top.
  - **Push**: Insert an item at the top.
  - **Pop**: Remove the item at the top.

Q: Can we access items other than at the top?
A: By definition, No.



Array-based Implementation

- Stack representation




```
#define MAX_STACK    100

typedef enum { false, true } bool;
typedef int Data;

typedef struct {
    Data items[MAX_STACK];
    int top;
} Stack;
```

Array-based Implementation

- Operations

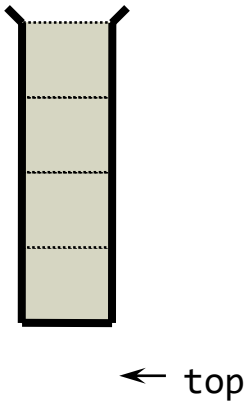
```
// Make stack empty.
void InitStack(Stack *pstack);
// Check whether stack is full.
bool IsFull(Stack *pstack);
// check whether stack is empty.
bool IsEmpty(Stack *pstack);

// Read the item at the top.
Data Peek(Stack *pstack);
// Insert an item at the top.
void Push(Stack *pstack, Data item);
// Remove the item at the top.
void Pop(Stack *pstack);
```

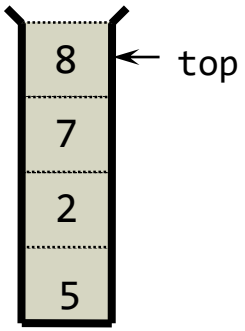
Array-based Implementation

- Initialize and IsFull operations

```
// Make stack empty.
void InitStack(Stack *pstack)
{
    pstack->top = -1;
}
```



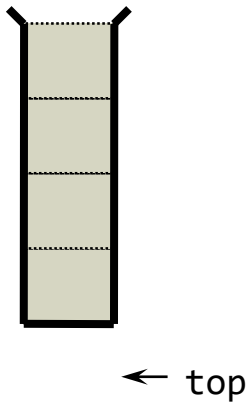

```
// Check whether stack is full.
bool IsFull(Stack *pstack)
{
    return pstack->top == MAX_STACK - 1;
}
```



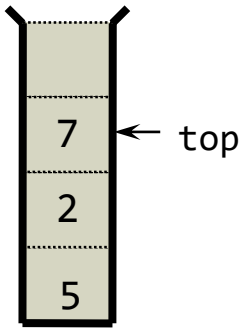
Array-based Implementation

- IsEmpty and Peek operations

```
// check whether stack is empty
bool IsEmpty(Stack *pstack)
{
    return pstack->top == -1;
}
```



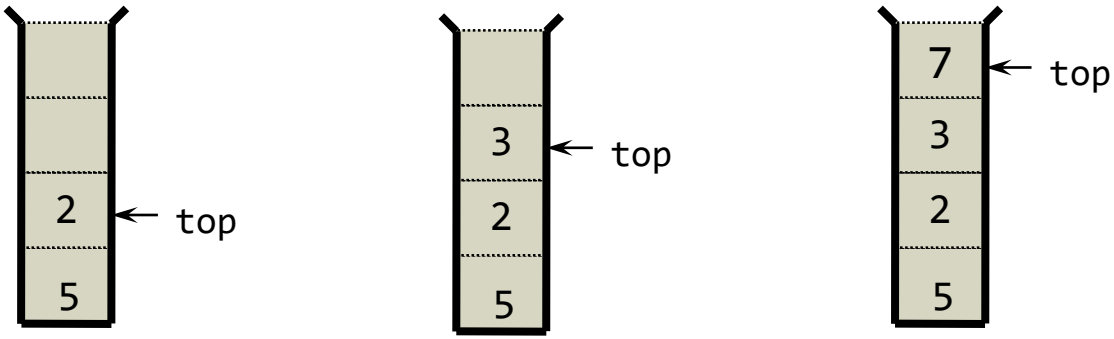

```
// Read the item at the top.
Data Peek(Stack *pstack)
{
    if (IsEmpty(pstack))
        exit(1); //error: empty stack
    return pstack->items[pstack->top];
}
```



Array-based Implementation

- Push operation

```
// Insert an item at the top.
void Push(Stack *pstack, Data item)
{
    if (IsFull(pstack))
        exit(1); //error: stack full
    pstack->items[++(pstack->top)] = item;
}
```

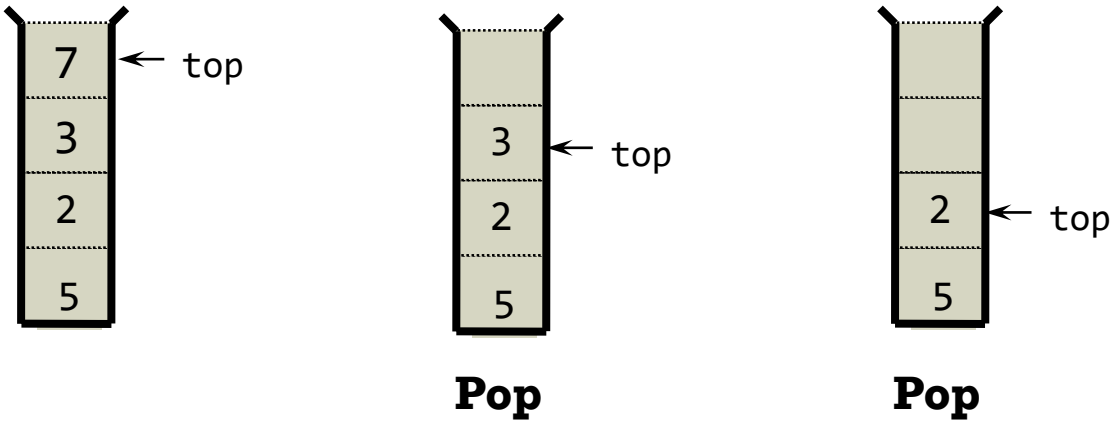


Push 3 **Push 7**

Array-based Implementation

- Pop operation

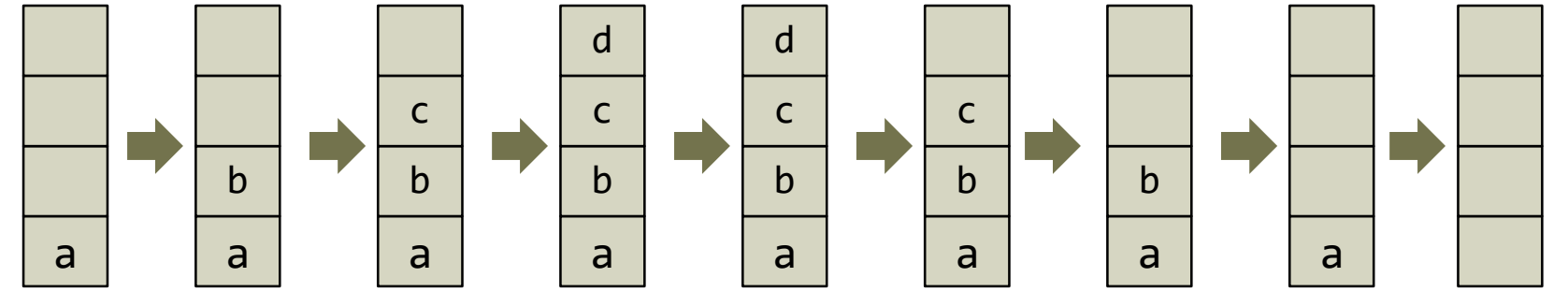
```
// Remove the item at the top.
void Pop(Stack *pstack)
{
    if (IsEmpty(pstack))
        exit(1); //error: empty stack
    --(pstack->top);
}
```



Print a Reverse String

- Print a string in the reverse order.
  - E.g., abcd → dcba

- How to do this with a stack?
  - Push all characters into stack.
  - Read the top, print it and pop until stack is empty.



Print a Reverse String

- Implementation

```
void ReversePrint(char* s, int len)
{
    Stack stack;
    char ch;

    InitStack(&stack);// Make a stack empty.
    for (int i = 0; i < len; i++)    // Push characters.
        Push(&stack, s[i]);

    while (!IsEmpty(&stack))// Pop characters.
    {
        ch = Peek(&stack);
        printf("%c", ch);
        Pop(&stack);
    }
}
```

Parenthesis Matching

- Problem
  - Check if each opening symbol has a corresponding closing symbol and the pairs of parentheses are nested properly.

First open waits until last close

(( ) (( )) ( ))

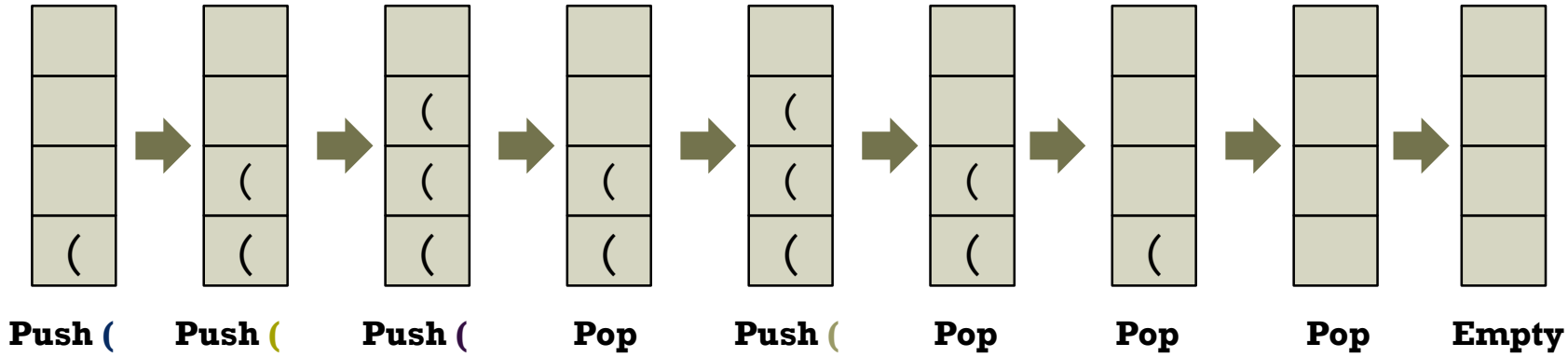
Most recent open matches first close

- Example
  - Balanced: (()()()()), ((((())))), (()((())()))
  - Unbalanced: (((((((()), ())), (()()(()

Parenthesis Matching

- How to do this with a stack?
  - Push all open symbols into stack.
  - Whenever finding the close symbol, pop the open symbol.
    - If the stack is empty, it is not balanced.
  - After reading all parentheses, check the status of the stack.
    - If the stack is empty, it is balanced.
    - Otherwise, it is not balanced.

- Example: ((( ) ( )))



Parenthesis Matching

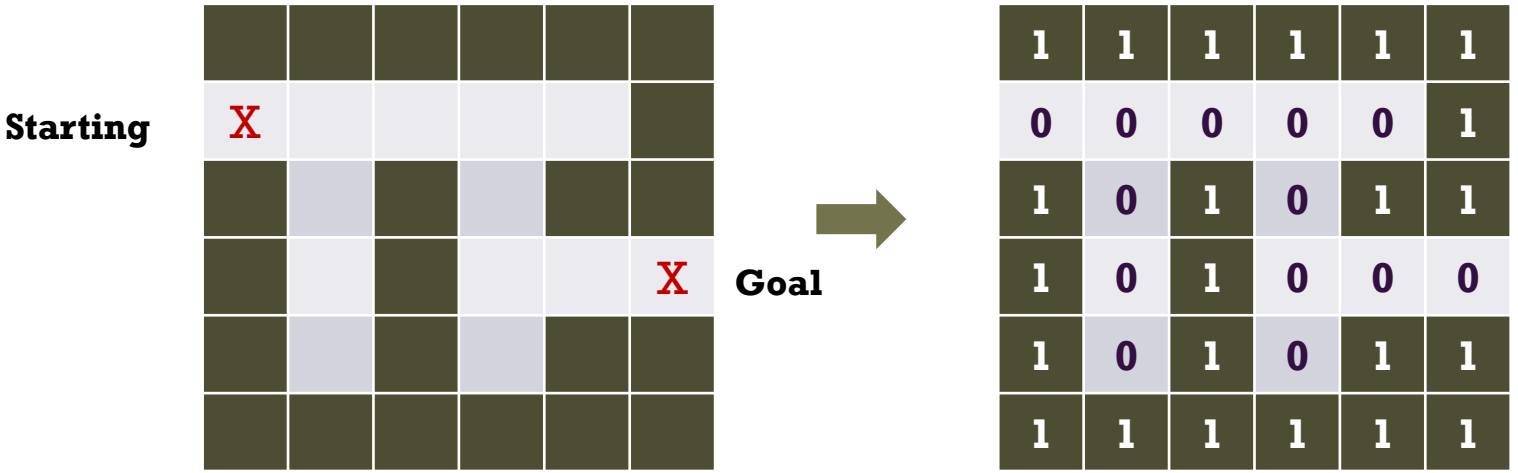
- Implementation

```
bool IsParanBalanced(char* exp, int len)
{
    Stack stack;
    InitStack(&stack);           // Make a stack empty.
    for (int i = 0; i < len; i++) {
        if (exp[i] == '(')       // Check open symbol
            Push(&stack, exp[i]);
        else if (exp[i] == ')') { // Check close symbol
            if (IsEmpty(&stack))
                return false;    // Unbalanced case
            else
                Pop(&stack);
        }
    }
    if (IsEmpty(&stack))
        return true;     // Balanced case
    else
        return false;    // Unbalanced case
}
```

Maze Solving

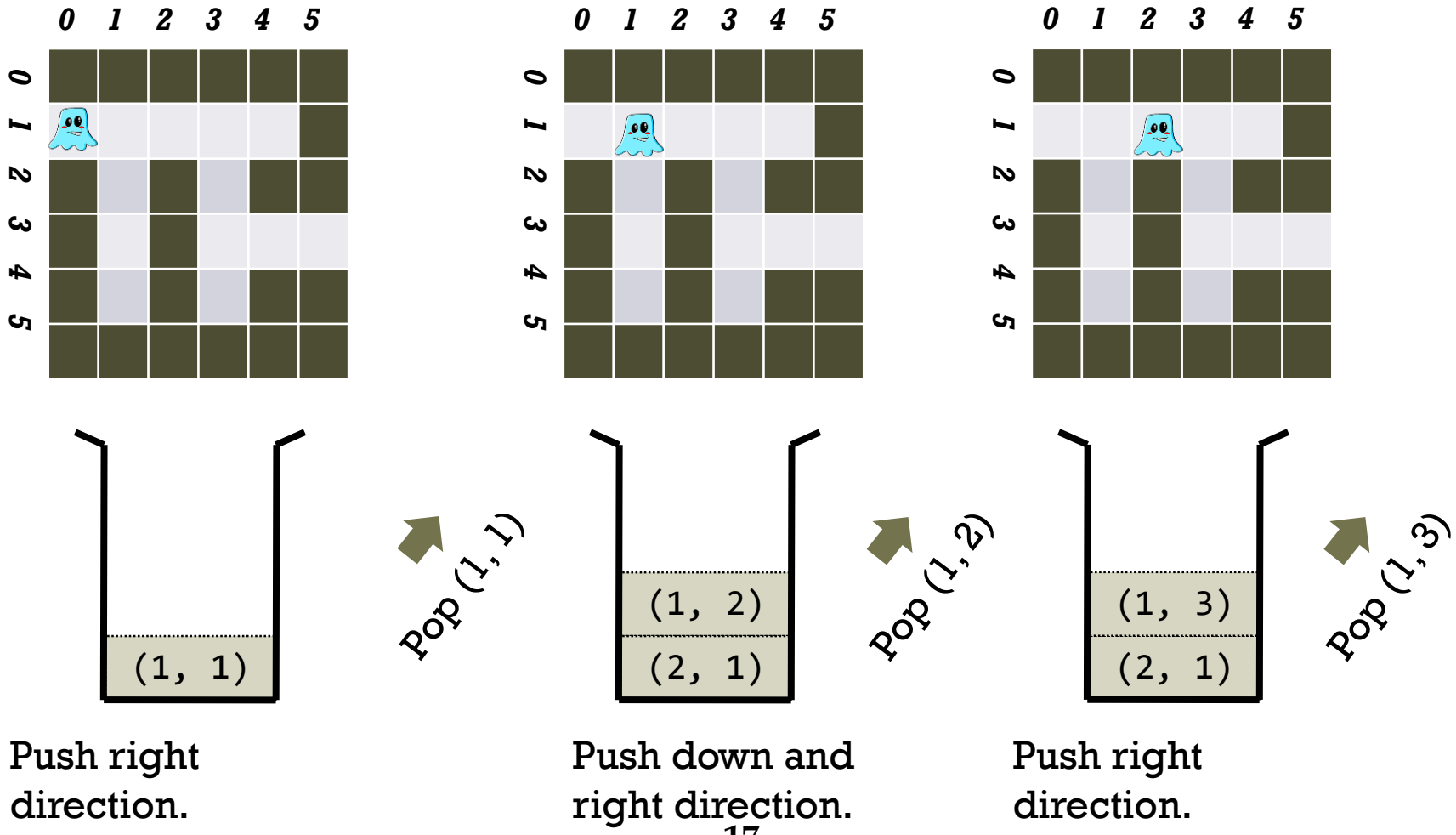
- Problem
  - Find a path from the starting positon to the goal point position.
  - At any moment, you can only move one step in one of four directions (up, down, left, and right).

- Representation
  - The maze is represented by a 2D binary array.
    - 0: path, 1: block



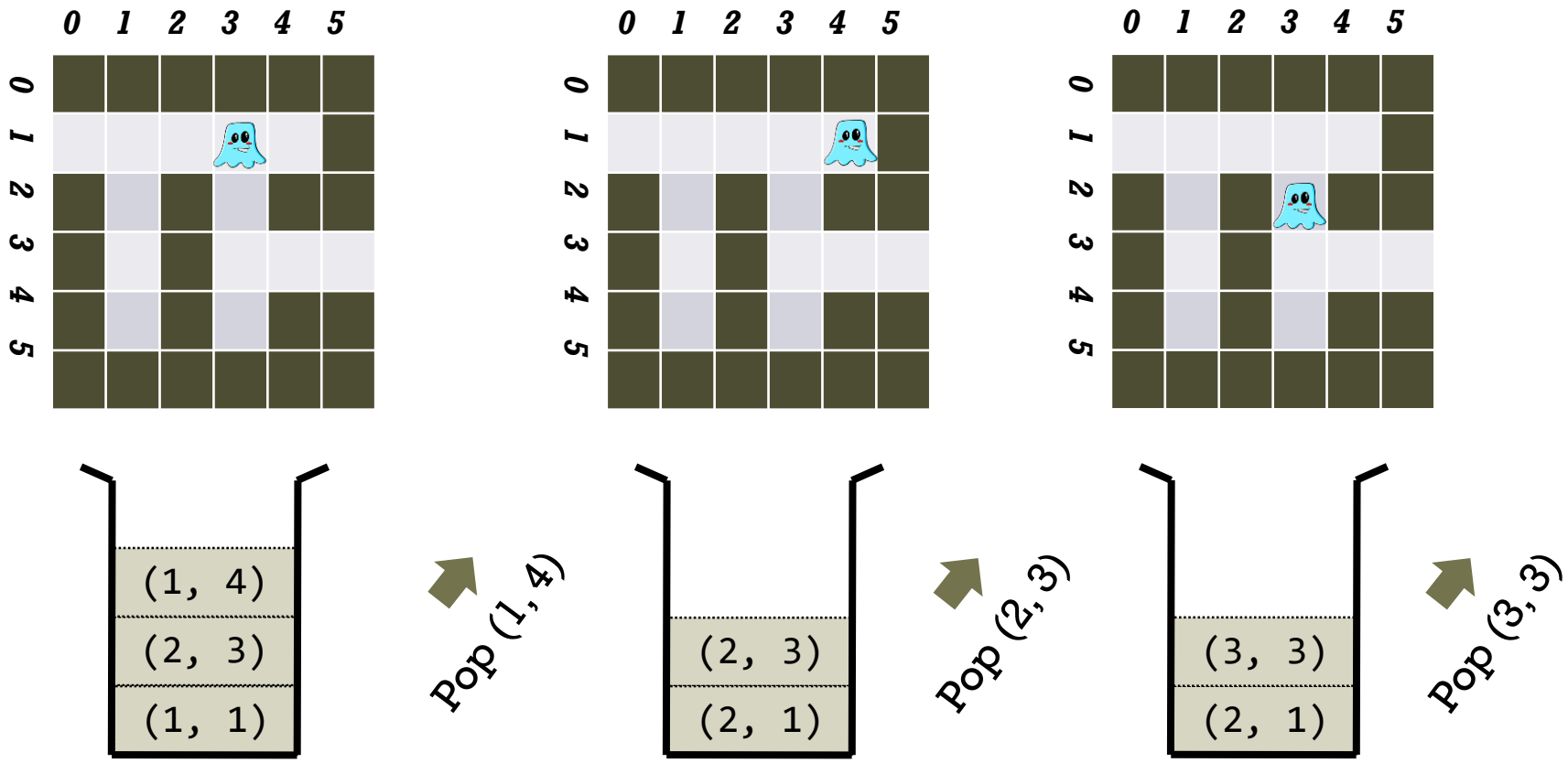
Maze Solving

- How to do this with a stack?



Maze Solving

- How to do this with a stack?

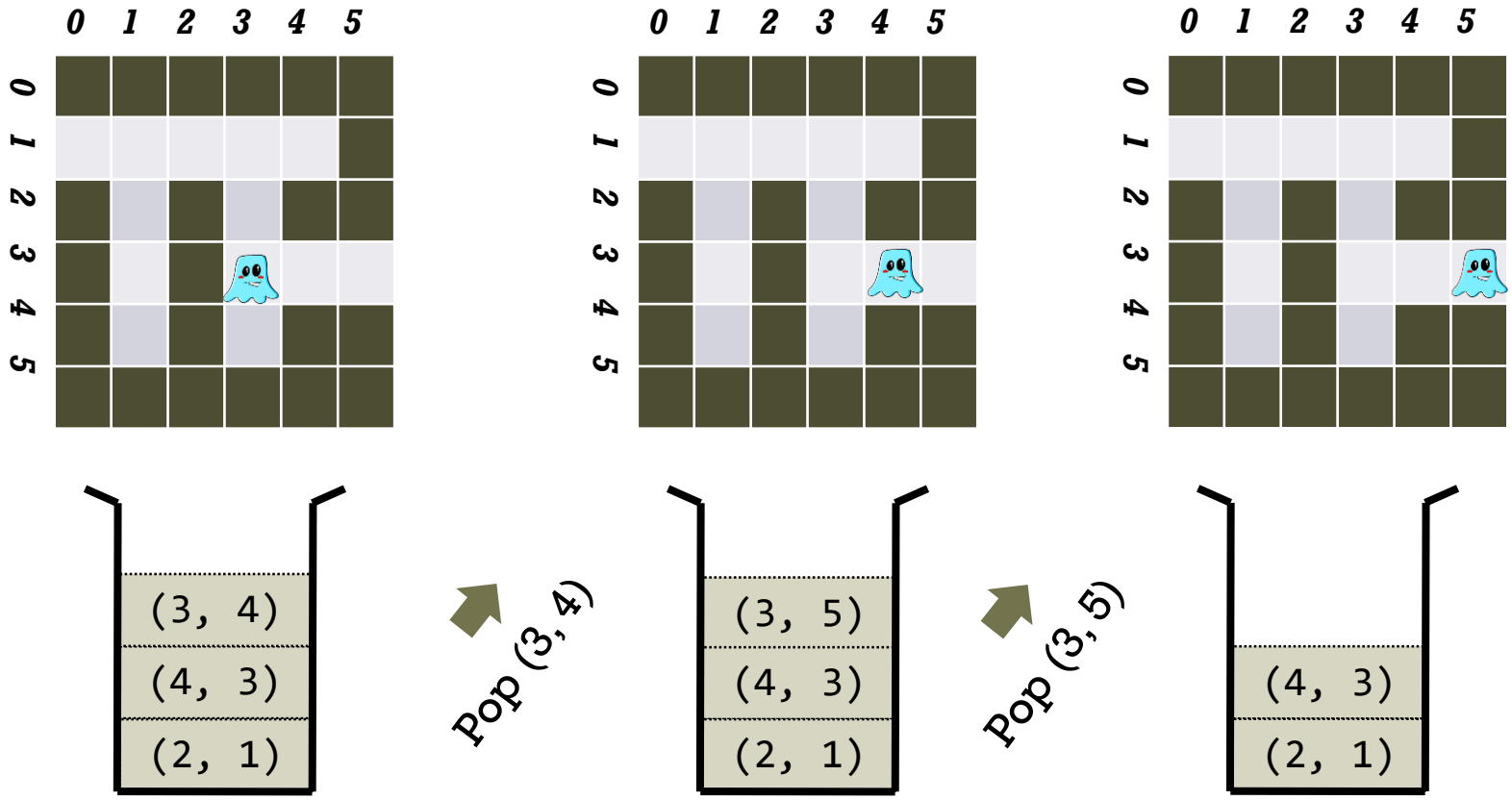


Push down and right direction.

Push down direction.

Maze Solving

- How to do this with a stack?



Push down and right direction.

Push right direction.

Maze Solving

- Overall process

1. When extending the path, push a new position on the stack.

1.1. At the current position, extend the path one step by trying to go three directions (up, down, or right).

2. Pop a position to move the current position.

3. If none of the neighboring positions is available, pop a position on the stack.

3.1. If the stack is empty, it indicates failure.

Repeat steps **1~3** until the goal has been reached.

Evaluation of Expression

- How to evaluate the expression?

```
3 + 4 * ( 5 / 2 ) + ( 7 + 9 * 3 )
```

- Evaluation by human
  - Assign to each operator a priority.
  - Use parenthesis and evaluate inner-most ones.

```
( ( 3 + ( 4 * ( 5 / 2 ) ) ) + ( 7 + ( 9 * 3 ) ) )
```

- How to evaluate by computer?
  - How to evaluate the operators with parenthesis
  - How to determine the precedence of operators

Infix, Postfix, and Prefix

- Infix notation: X + Y
  - Operators are written in-between their operands.
  - Need extra information to make the order of evaluation of the operators clear.

- Postfix notation: X Y +
  - Operators are written after their operands.
  - The order of evaluation of operators is always left-to-right.
  - Unnecessary to use parenthesis and precedence of operators.

- Prefix notation: + X Y
  - Operators are written before their operands.
  - As for postfix, operators are evaluated left-to-right.

Infix vs. Postfix

- Example

Infix	Postfix	Prefix
A * B + C / D	A B * C D / +	+ * A B / C D
A * (B + C) / D	A B C + * D /	/ * A + B C D
A * (B + C / D)	A B C D / + *	* A + B / C D
A * B / C – D	A B * C / D -	- / * A B C D

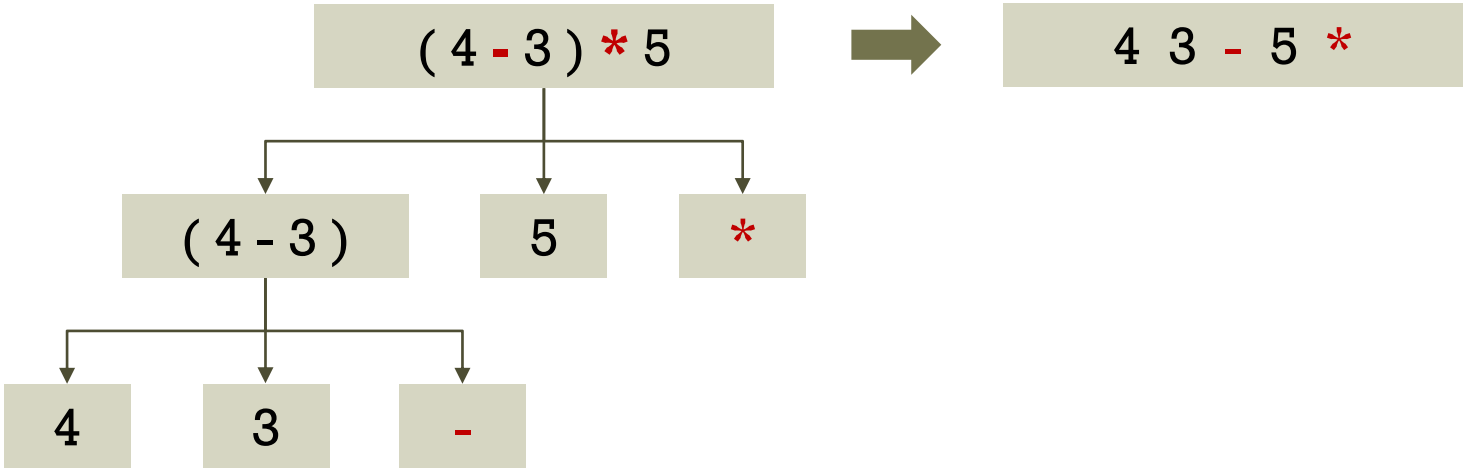
- Why is postfix useful?
  - No parenthesis is needed
  - No precedence of operators is needed

Converting Infix to Postfix

- Applying the conversion rule in a recursive way

```
A * B   =>  AB*
```

- Example



How to Evaluate Postfix Notation

- Overall process

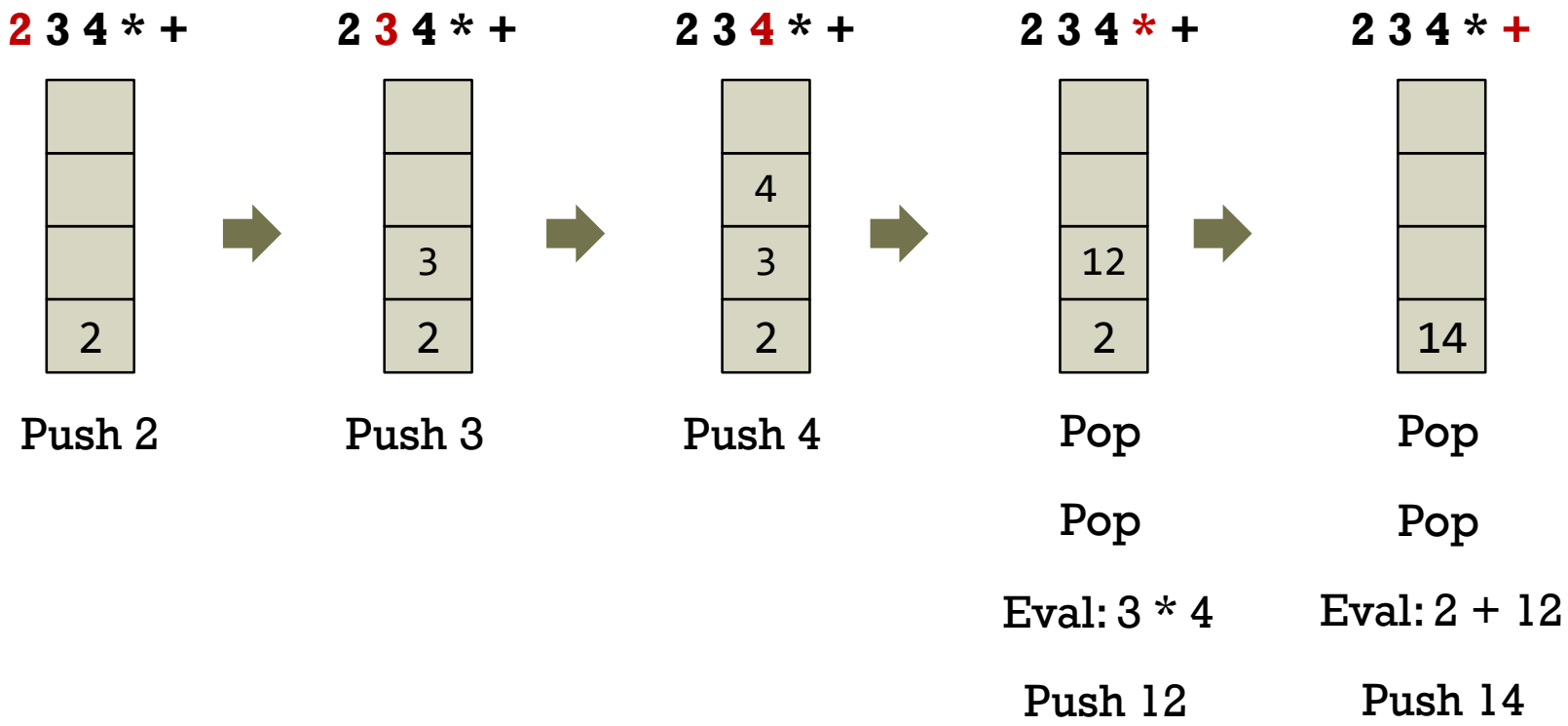
  1. Push operands on the stack until finding an operator.

  2. If the operator is found, pop two operands and evaluate the operator. Then, push the result on the stack.

  Repeat steps **1~2** until reading all characters in postfix notation.

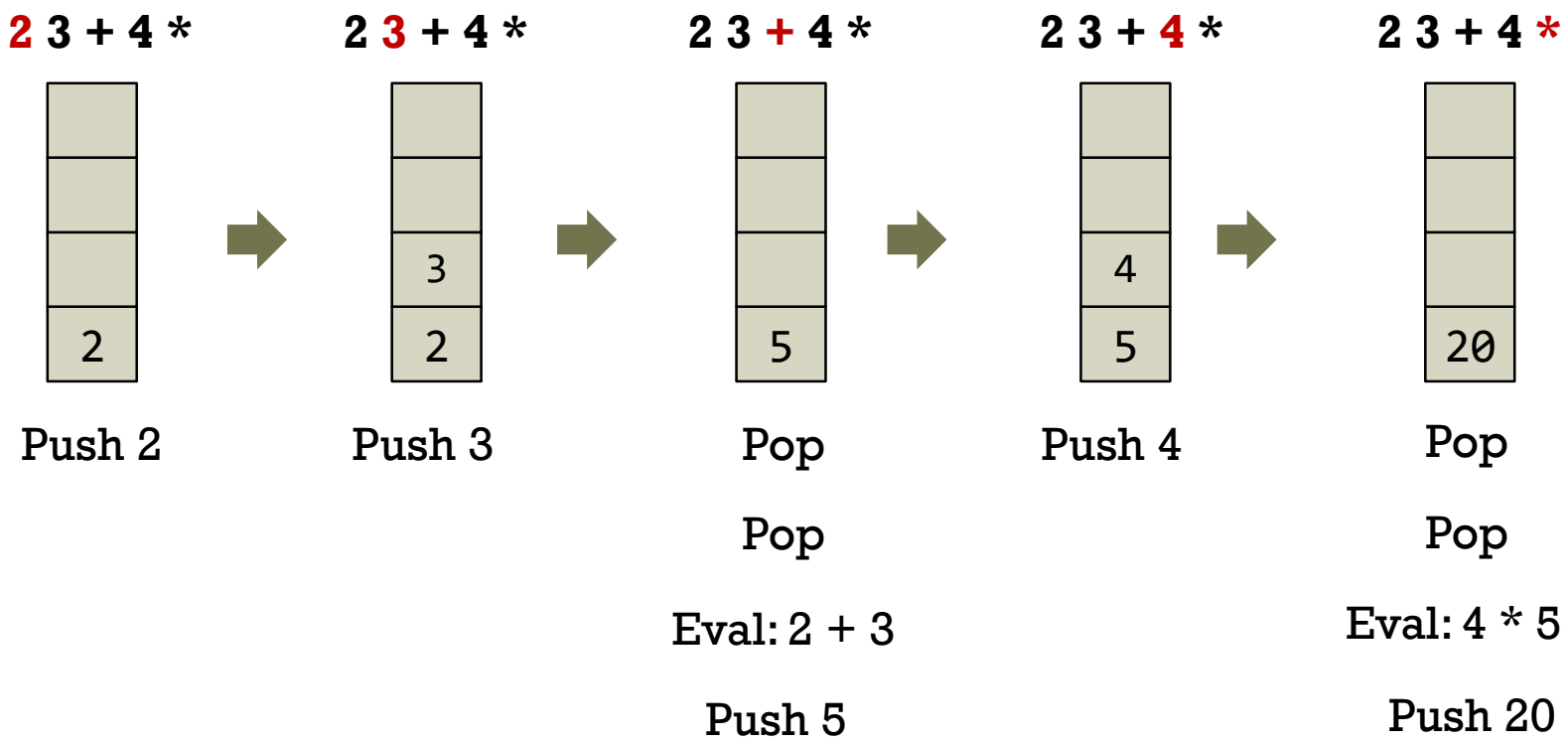
How to Evaluate Postfix Notation

- Example: 2 3 4 * +
  - Push operands on the stack until finding an operator.
  - If the operator is found, pop two operands and evaluate the operator. Then, push the result on the stack.



How to Evaluate Postfix Notation

- Example: 2 3 + 4 *
  - Push operands on the stack until finding an operator.
  - If the operator is found, pop two operands and evaluate the operator. Then, push the result on the stack.



How to Evaluate Postfix Notation

```
int EvalPostfix(char* exp, int len)
{
     Stack stack;
     int op1, op2;
     InitStack(&stack);
     for (int i = 0; i < len; i++) {
          if (isdigit(exp[i]))   // Push an operand.
               Push(&stack, exp[i] - '0');
          else {
               // Evaluate an operator.
               op2 = Peek(&stack); Pop(&stack);
               op1 = Peek(&stack); Pop(&stack);
               if (exp[i] == '+')
                    Push(&stack, op1 + op2);
               else if (exp[i] == '-')
                    Push(&stack, op1 - op2);
               else if (exp[i] == '*')
                    Push(&stack, op1 * op2);
               else if (exp[i] == '/')
                    Push(&stack, op1 / op2);
          }
     }
     return Peek(&stack);
}
```

Converting Infix to Postfix

- Overall process

1. If the operand is found, print it.

2. If the operator is found, push it into stack, but before pushing

2.1. See the operator at the top of the stack.

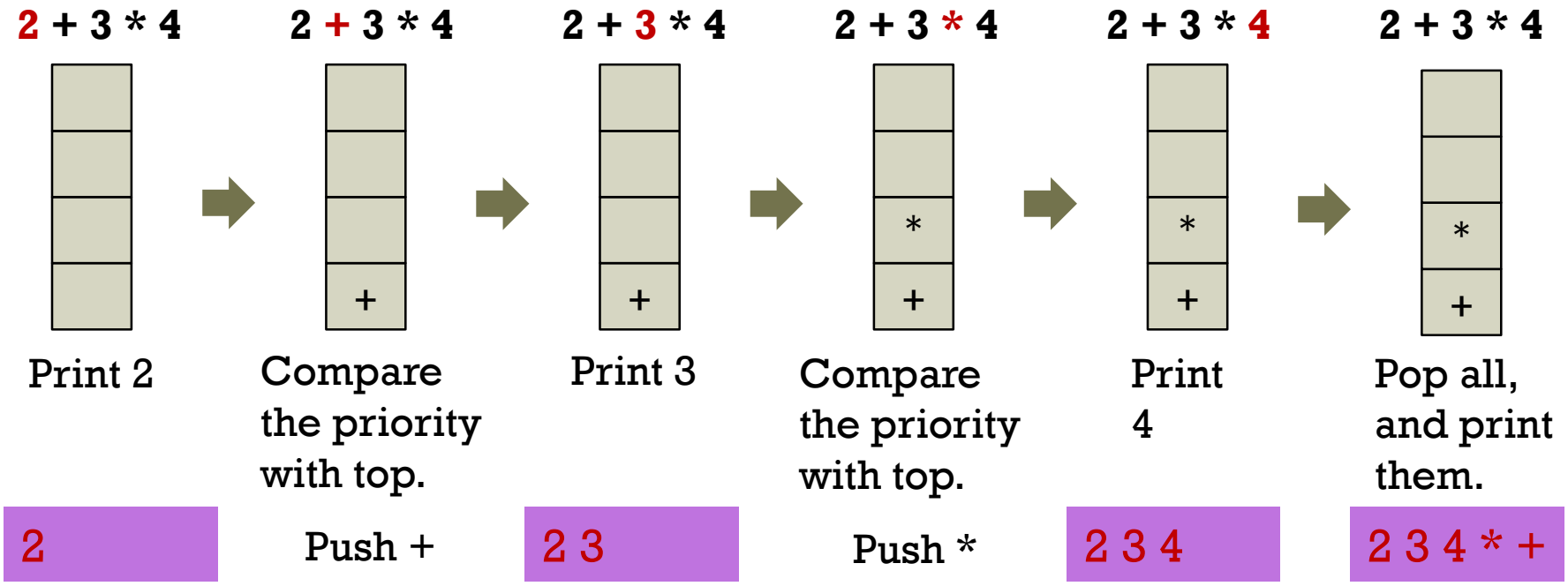
2.2. If the priority of the incoming operator is lower than or equal to the top, pop and print the top and go to step **2.1**.

Repeat steps **1~2** until reading all characters in infix notation.

3. At the end of the infix notation, pop all operators.

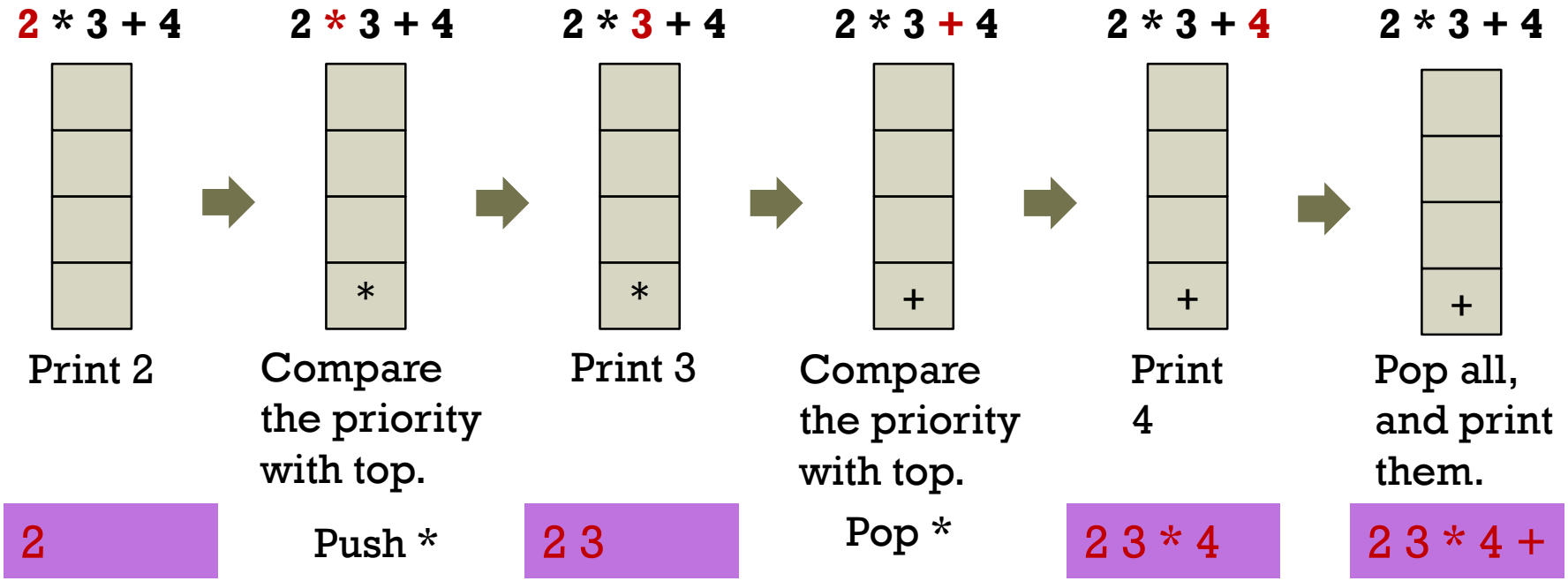
Converting Infix to Postfix

- Example: 2 + 3 * 4
  - While reading a character form infix notation
    - If we find operand, print it.
    - Otherwise, check the priority of operator and determine whether push it into stack.
  - Pop all operators at the end of the infix notation.



Converting Infix to Postfix

- Example: 2 * 3 + 4
  - While reading a character form infix notation
    - If we find operand, print it.
    - Otherwise, check the priority of operator and determine whether push it into stack.
  - Pop all operators at the end of the infix notation.



Converting Infix to Postfix

- Implementation: 2 * 3 + 4

```
void ConvInfixToPostfix(char* exp, char* convExp, int len)
{
     Stack stack;
     int idx = 0;
     InitStack(&stack);
     for (int i = 0; i < len; i++)
     {
          if (isdigit(exp[i]))
               convExp[idx++] = exp[i]; // Print an operand.
          else {
               while (!IsEmpty(&stack) && ComPriority(Peek(&stack), exp[i])) {
                    convExp[idx++] = Peek(&stack); // Print an operator.
                    Pop(&stack); // Pop an operator.
               }
               Push(&stack, exp[i]); // Push an operator.
          }
     }
     while (!IsEmpty(&stack)) {
          convExp[idx++] = Peek(&stack); // Print an operator.
          Pop(&stack); // Pop an operator.
     }
}
```

Converting Infix to Postfix

- Implementation

```
int GetPriority(char op)
{
     if (op == '*' || op == '/')
          return 2;
     else if (op == '+' || op == '-')
          return 1;
     else
          return 0;
}

bool ComparePriority(char op1, char op2)
{
     int op1_pr = GetPriority(op1);
     int op2_pr = GetPriority(op2);

     if (op1_pr >= op2_pr)
          return true;
     else
          return false;
}
```