Stack: isEmpty, isFull and Top

```
int isEmpty(STACK *s) {
    return s->top == -1;
}

int isFull(STACK *s) {
    return s->top == s->limit;
}

int Top(STACK *s) {
    if (isEmpty(s)) {
        printf("Stack is empty\n");
        return -1;
    }
    return s->info[s->top];
}
```

Stack: Push and Pop

```
void push(STACK *s, int element) {
    if (isFull(s)) {
        printf("Stack full: insertion denied\n");
        return;
    }
    s->info[++s->top] = element; // pointer to end
        of list
    return;
}

int pop(STACK *s) {
    if (isEmpty(s)) {
        printf("Stack empty: deletion denied\n");
        return -999;
    }
    return (s->info[s->top--]);
}
```

Stack: Print

```
void printStack(STACK *s) {
    int i;
    if(isEmpty(s)) {
        printf("Stack is empty\n");
        return;
    }
    i = -1;
    while (i < s->top) {
        printf("%d\t", s->info[++i]);   // print
            elements
    }
    printf("\n");
    return;
}
```

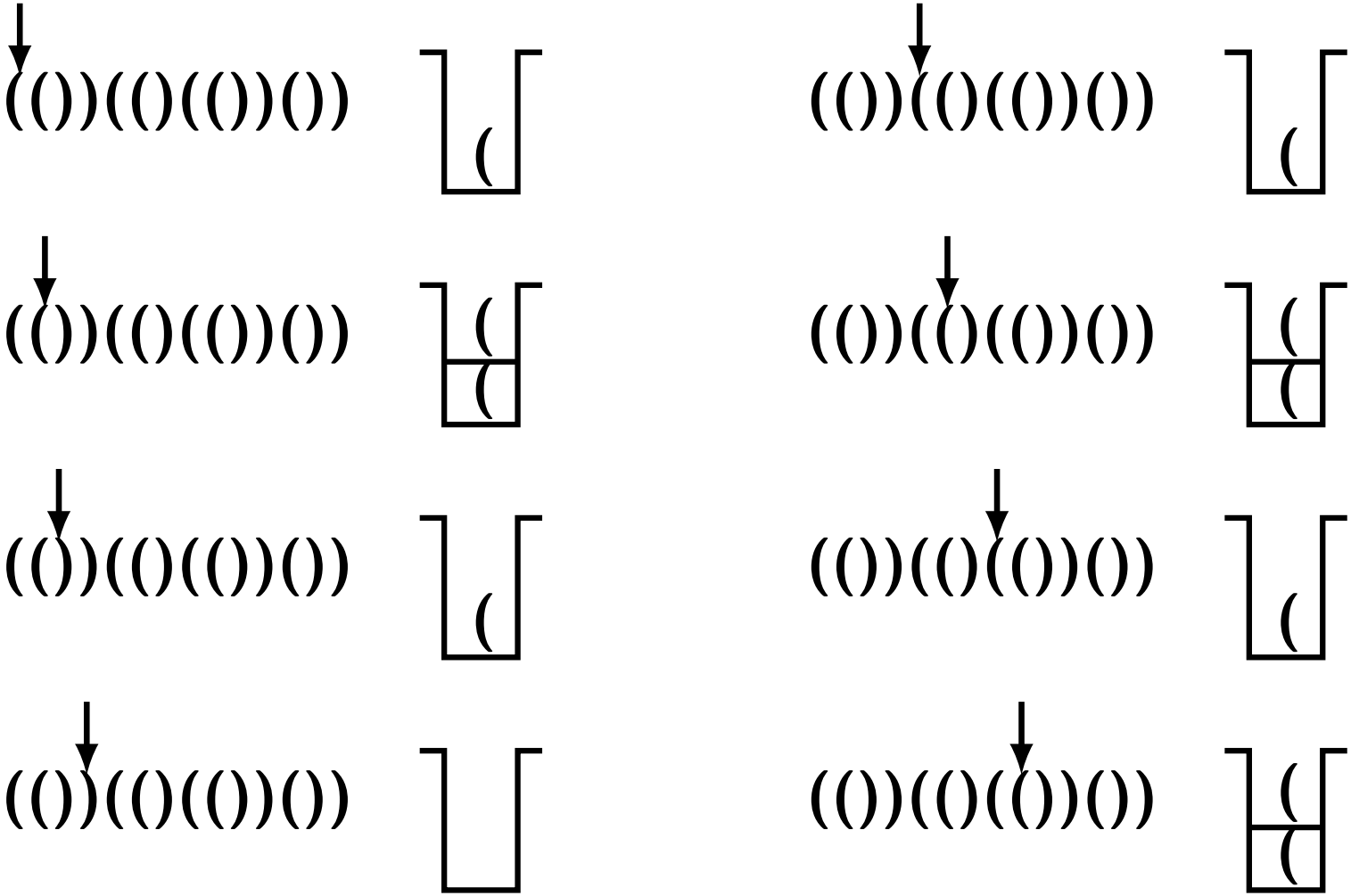
Application of Stack

- Parenthesis matching.
- Evaluation of expression.
- Converting expression from one form to another.
- Used also in DFS, in general backtracking.
- Used in memory management.

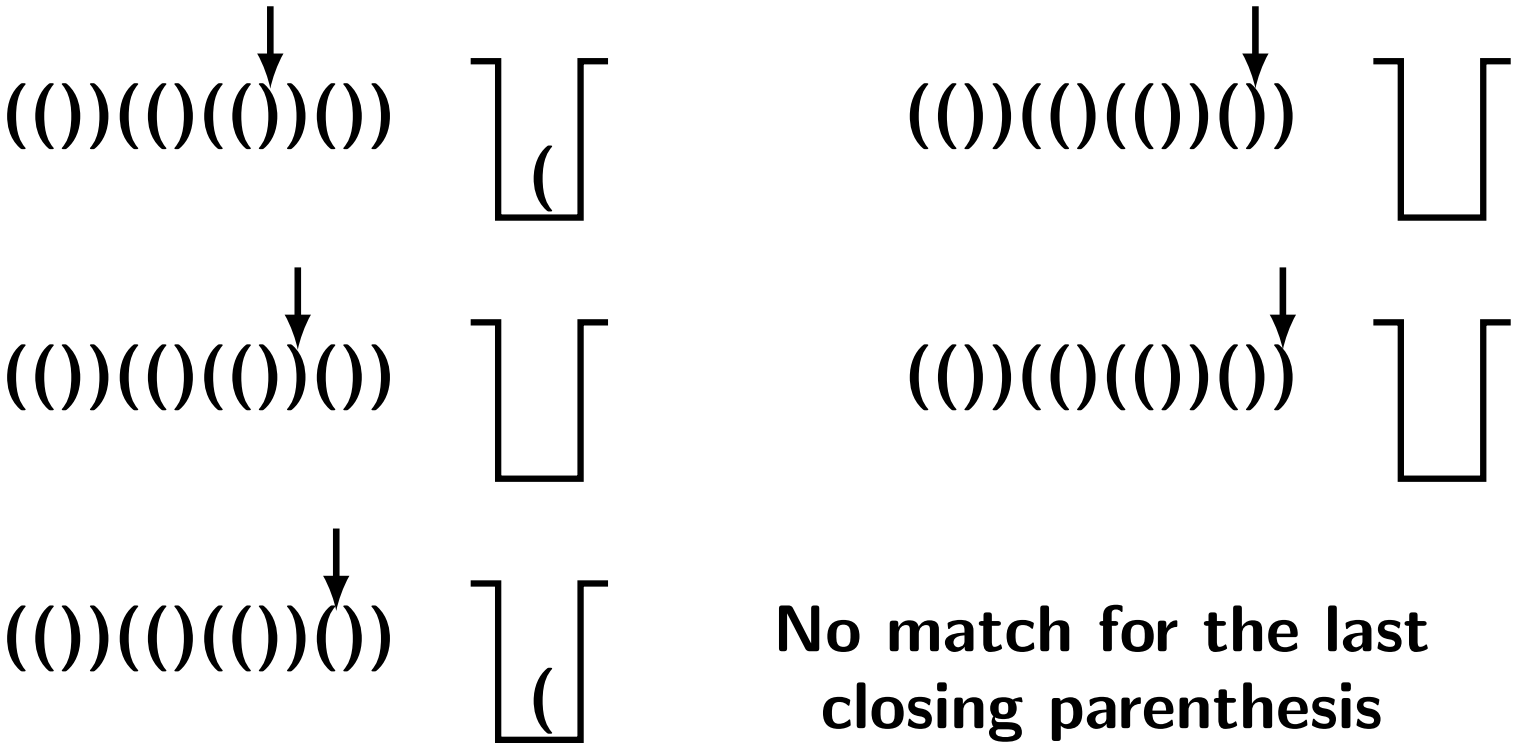
Parenthesis Matching

- For parenthesis matching:
  - Scan the parentheses in left to right order
  - On encountering an opening parenthesis push it to stack.
  - On encountering a closing parenthesis just pop the topmost parenthesis from the stack.
- If after all input scanning is over and stack is empty the parentheses are matched.
- If stack is empty but the next input symbol is an closing parenthesis match fails.
- If stack is not empty but input scanning is over the match fails.

Parenthesis Matching



Parenthesis Matching



No match for the last closing parenthesis

Infix to Postfix Conversion

(A + B) * C - D / (E + F)

~~(~~ A B + ~~)~~ C * D ~~(~~ E F + ~~)~~ / -

A B + C * D E F + / -

Expression Evaluation

- Assume we have postfix expression: AB+C*DEF+/-
- Let A = 4, B = 6, C = 2, D = 12, E = 1, F = 2.
- The evaluation of infix form of the expression give (4 + 6) * 2 - 12 / (1 + 2) = 20 - 12/3 = 16.
- For evaluation, scan the expression from left to right and use following rules repeatedly:
  1. Push symbol onto S if it is an operand.
  2. On encountering an operator, pop the correct number of operands, apply operation and push result on to S.
- When finished the result is on the top of S.

Example: Postfix Evaluation

4 6 + 2 * 12 1 2 + / -				
Step	Symbol	Stack	Output	Description
1	4	4	-	1st symbol
2	6	4, 6	-	2nd symbol
3	+	10	10	3nd symbol
4	3	10, 2	-	4th symbol
5	*	20	20	5th symbol
6	12	20, 12	-	6th symbol
7	1	20, 12, 1	-	7th symbol
8	2	20, 12, 1, 2	-	7th symbol
9	-	20, 12, 3	3	8th symbol
10	/	20, 4	4	9th symbol
11	-	16	16	10th symbol

Infix to Postfix

- Append a ”)” to input expression X.
- Push an ”(” onto an empty stack S.
- Scan X from left to right repeating following steps:
  1. If current symbol is an operand add to output.
  2. If current symbol is an ”(”, push it on to S.
  3. If current symbol is an operator:
     1. Repeatedly pop operators from stack with same or higher precedence and add them to output.
     2. Add the current symbol to stack.
  4. If current symbol is a ”)” then
     1. Repeatedly pop symbols from stack until a matching ”(”.
     2. Pop and discard the ”(”

Example

(A+B)*C - D/(E+F))⟵ added				
Step	Symbol	Stack	Output	Description
1	-	(	-	start
2	(	((	-	1st symbol
3	A	((	A	2nd symbol
4	+	((+	A	3rd symbol
5	B	((+	AB	4th symbol
6	*	((*	AB+	5th symbol
7	C	((*	AB+C	6th symbol
8	-	((-	AB+C*	7th symbol

Example Continued

(A+B)*C - D/(E+F))				
Step	Symbol	Stack	Output	Description
9	D	((-	AB+C*D	8th symbol
9	/	((/	AB+C*D-	9th symbol
10	(	((/(	AB+C*D-	10th symbol
11	E	((/(	AB+C*D-E	11th symbol
12	+	((/(+	AB+C*D-E	12th symbol
12	F	((/(+	AB+C*D-EF/	13th symbol
13	)	(	AB+C*D-EF+/	14th symbol
14	)	-	AB+C*D-EF+/	Added) symbol