Computer Programming II

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

Advanced Stack

Use array for stack

- Example: arr_stack/arr_stack.c

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 50

void push(int i);
int pop(void);

int  *tos, *ptr, stack[SIZE];

int main(void) {
    int value;

    tos = stack; /* tos points to the top of stack */
    ptr = stack; /* initialize ptr */

    do {
        printf("Enter value [0 for pop; -1 for exit]: ");
        scanf("%d", &value);

        if(value != 0) push(value);
        else printf("value on top is %d\n", pop());

    } while(value != -1);

    return 0;
}
```

Use array for stack

- Example: arr_stack/arr_stack.c

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 50

void push(int i);
int pop(void);

int  *tos, *ptr, stack[SIZE];

int main(void) {
    int value;

    tos = stack; /* tos points to the top of stack */
    ptr = stack; /* initialize ptr */

    do {
        printf("Enter value [0 for pop; -1 for exit]: ");
        scanf("%d", &value);

        if(value != 0) push(value);
        else printf("value on top is %d\n", pop());

    } while(value != -1);

    return 0;
}
```

Use array for stack

- Example: arr_stack/arr_stack.c

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 50

void push(int i);
int pop(void);

int  *tos, *ptr, stack[SIZE];

int main(void) {
    int value;

    tos = stack; /* tos points to the top of stack */
    ptr = stack; /* initialize ptr */

    do {
        printf("Enter value [0 for pop; -1 for exit]: ");
        scanf("%d", &value);

        if(value != 0) push(value);
        else printf("value on top is %d\n", pop());

    } while(value != -1);

    return 0;
}
```

Use array for stack

- Example: arr_stack/arr_stack.c

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 50

void push(int i);
int pop(void);

int  *tos, *ptr, stack[SIZE];

int main(void) {
    int value;

    tos = stack; /* tos points to the top of stack */
    ptr = stack; /* initialize ptr */

    do {
        printf("Enter value [0 for pop; -1 for exit]: ");
        scanf("%d", &value);

        if(value != 0) push(value);
        else printf("value on top is %d\n", pop());

    } while(value != -1);

    return 0;
}
```

Use array for stack

- Example: arr_stack/arr_stack.c

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 50

void push(int i);
int pop(void);

int  *tos, *ptr, stack[SIZE];

int main(void) {
    int value;

    tos = stack; /* tos points to the top of stack */
    ptr = stack; /* initialize ptr */

    do {
        printf("Enter value [0 for pop; -1 for exit]: ");
        scanf("%d", &value);

        if(value != 0) push(value);
        else printf("value on top is %d\n", pop());

    } while(value != -1);

    return 0;
}
```

*tos

Use array for stack

- Example: arr_stack/arr_stack.c

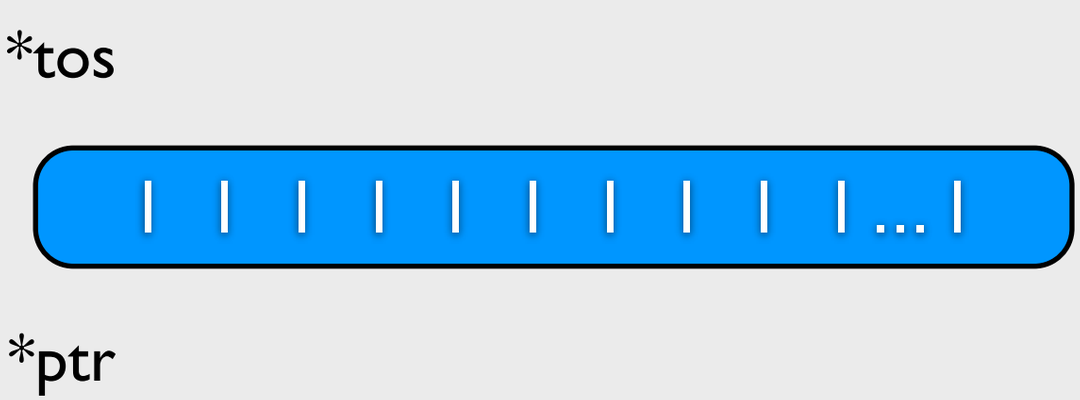
```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 50

void push(int i);
int pop(void);

int  *tos, *ptr, stack[SIZE];

int main(void) {
    int value;

    tos = stack; /* tos points to the top of stack */
    ptr = stack; /* initialize ptr */

    do {
        printf("Enter value [0 for pop; -1 for exit]: ");
        scanf("%d", &value);

        if(value != 0) push(value);
        else printf("value on top is %d\n", pop());

    } while(value != -1);

    return 0;
}
```

*tos

Use array for stack

- Example: arr_stack/arr_stack.c

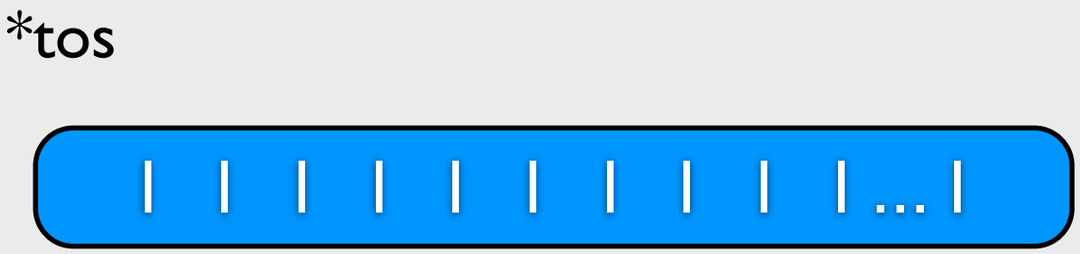
```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 50

void push(int i);
int pop(void);

int  *tos, *ptr, stack[SIZE];

int main(void) {
    int value;

    tos = stack; /* tos points to the top of stack */
    ptr = stack; /* initialize ptr */

    do {
        printf("Enter value [0 for pop; -1 for exit]: ");
        scanf("%d", &value);

        if(value != 0) push(value);
        else printf("value on top is %d\n", pop());

    } while(value != -1);

    return 0;
}
```



Use array for stack

- Example: arr_stack/arr_stack.c

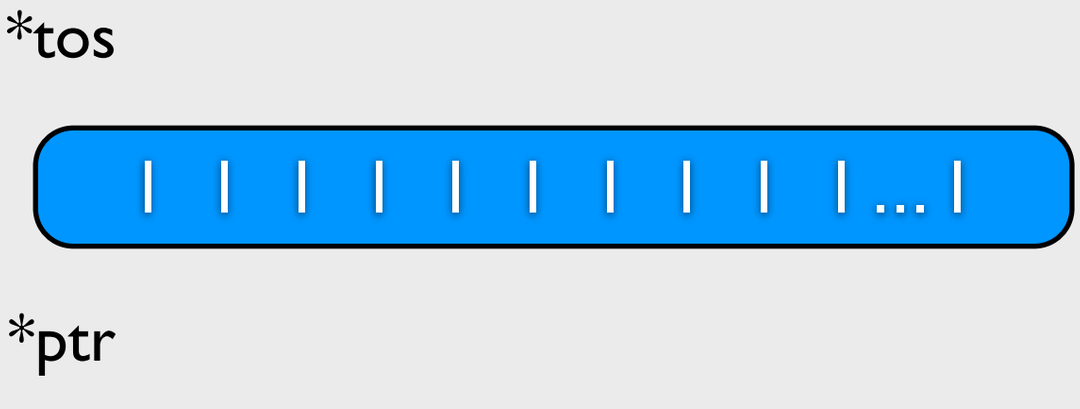
```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 50

void push(int i);
int pop(void);

int  *tos, *ptr, stack[SIZE];

int main(void) {
    int value;

    tos = stack; /* tos points to the top of stack */
    ptr = stack; /* initialize ptr */

    do {
        printf("Enter value [0 for pop; -1 for exit]: ");
        scanf("%d", &value);

        if(value != 0) push(value);
        else printf("value on top is %d\n", pop());

    } while(value != -1);

    return 0;
}
```



Use array for stack

- Example: arr_stack/arr_stack.c

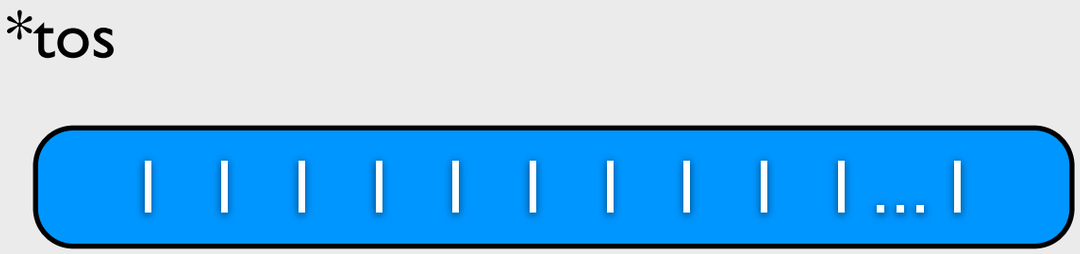
```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 50

void push(int i);
int pop(void);

int  *tos, *ptr, stack[SIZE];

int main(void) {
    int value;

    tos = stack; /* tos points to the top of stack */
    ptr = stack; /* initialize ptr */

    do {
        printf("Enter value [0 for pop; -1 for exit]: ");
        scanf("%d", &value);

        if(value != 0) push(value);
        else printf("value on top is %d\n", pop());

    } while(value != -1);

    return 0;
}
```



Use array for stack

- Example: arr_stack/arr_stack.c

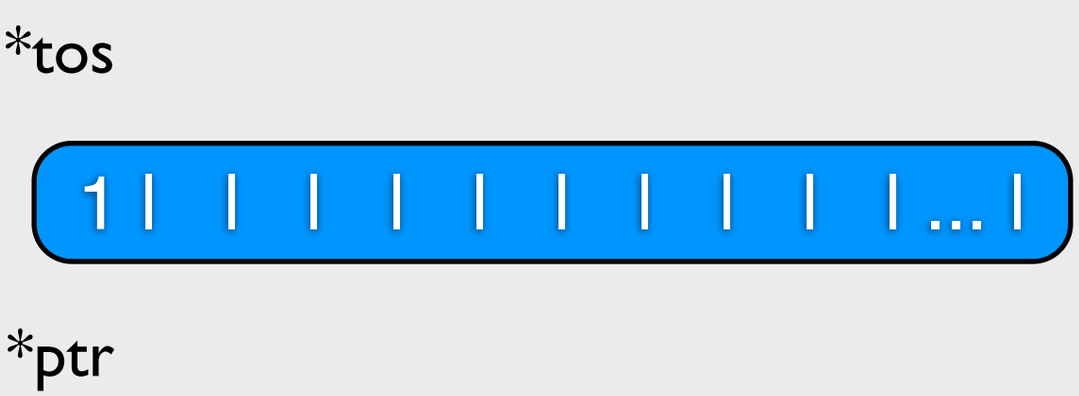
```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 50

void push(int i);
int pop(void);

int  *tos, *ptr, stack[SIZE];

int main(void) {
    int value;

    tos = stack; /* tos points to the top of stack */
    ptr = stack; /* initialize ptr */

    do {
        printf("Enter value [0 for pop; -1 for exit]: ");
        scanf("%d", &value);

        if(value != 0) push(value);
        else printf("value on top is %d\n", pop());

    } while(value != -1);

    return 0;
}
```



Use array for stack

- Example: arr_stack/arr_stack.c

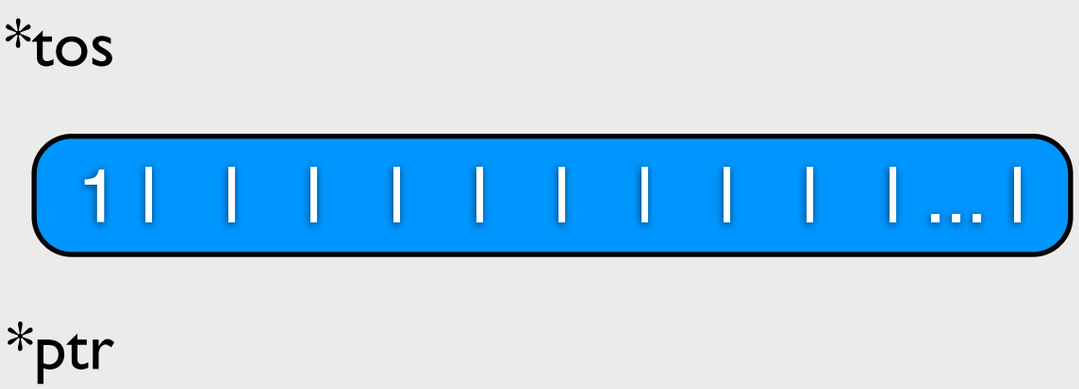
```
28 void push(int i) {
29     ptr++;
30     if(ptr == (tos+SIZE)) {
31         printf("Stack Overflow.\n");
32         exit(1);
33     }
34     *ptr = i;
35 }
```



Use array for stack

- Example: arr_stack/arr_stack.c

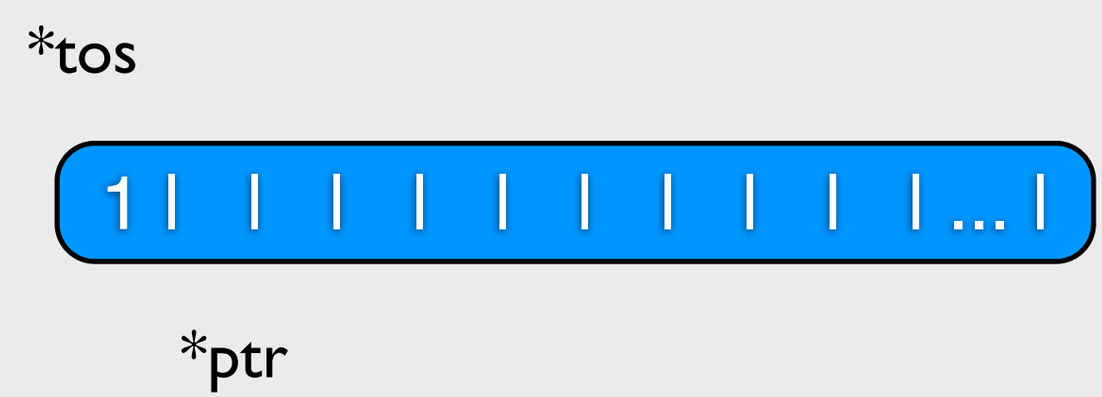
```
28 void push(int i) {
29     ptr++;
30     if(ptr == (tos+SIZE)) {
31         printf("Stack Overflow.\n");
32         exit(1);
33     }
34     *ptr = i;
35 }
```



Use array for stack

- Example: arr_stack/arr_stack.c

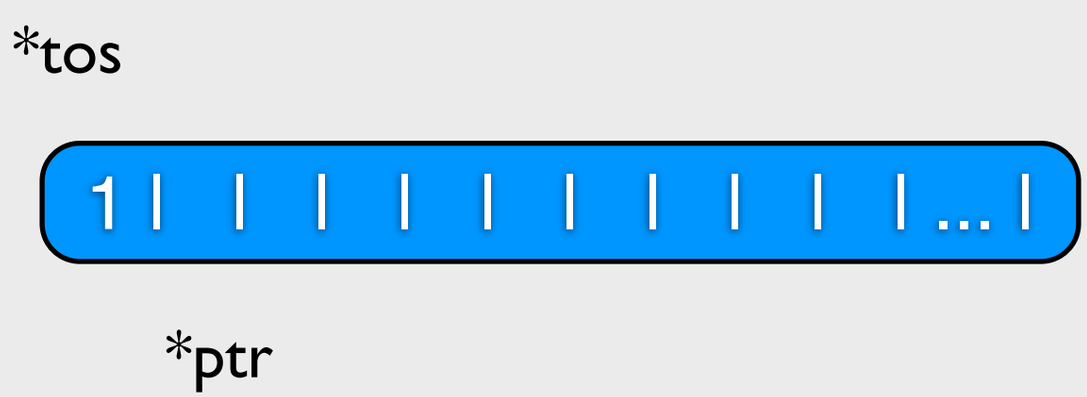
```
28 void push(int i) {
29     ptr++;
30     if(ptr == (tos+SIZE)) {
31         printf("Stack Overflow.\n");
32         exit(1);
33     }
34     *ptr = i;
35 }
```



Use array for stack

- Example: arr_stack/arr_stack.c

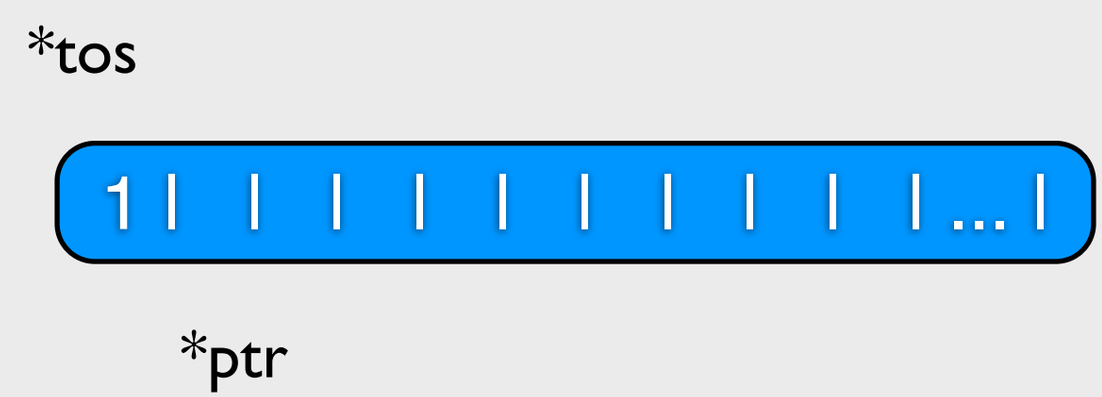
```
28 void push(int i) {
29     ptr++;
30     if(ptr == (tos+SIZE)) {
31         printf("Stack Overflow.\n");
32         exit(1);
33     }
34     *ptr = i;
35 }
```



Use array for stack

- Example: arr_stack/arr_stack.c

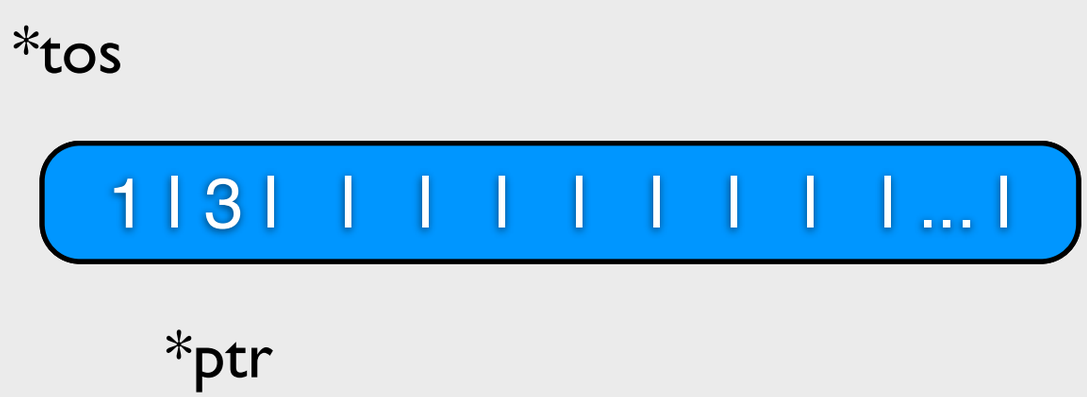
```
28 void push(int i) {
29     ptr++;
30     if(ptr == (tos+SIZE)) {
31         printf("Stack Overflow.\n");
32         exit(1);
33     }
34     *ptr = i;
35 }
```



Use array for stack

- Example: arr_stack/arr_stack.c

```
28 void push(int i) {
29     ptr++;
30     if(ptr == (tos+SIZE)) {
31         printf("Stack Overflow.\n");
32         exit(1);
33     }
34     *ptr = i;
35 }
```



Use array for stack

- Example: arr_stack/arr_stack.c

```
37 int pop(void) {
38     if(ptr == tos) {
39         printf("Stack Underflow.\n");
40         exit(1);
41     }
42     ptr--;
43     return *(ptr+1);
44 }
```

Use array for stack

- Example: arr_stack/arr_stack.c

```
37 int pop(void) {
38     if(ptr == tos) {
39         printf("Stack Underflow.\n");
40         exit(1);
41     }
42     ptr--;
43     return *(ptr+1);
44 }
```

Use array for stack

- Example: arr_stack/arr_stack.c

```
37 int pop(void) {
38     if(ptr == tos) {
39         printf("Stack Underflow.\n");
40         exit(1);
41     }
42     ptr--;
43     return *(ptr+1);
44 }
```

Use array for stack

- Example: arr_stack/arr_stack.c

```
37 int pop(void) {
38     if(ptr == tos) {
39         printf("Stack Underflow.\n");
40         exit(1);
41     }
42     ptr--;
43     return *(ptr+1);
44 }
```

Use array for stack

- Example: arr_stack/arr_stack.c

```
37 int pop(void) {
38     if(ptr == tos) {
39         printf("Stack Underflow.\n");
40         exit(1);
41     }
42     ptr--;
43     return *(ptr+1);
44 }
```

Example

- Infix to Postfix
  - Infix
    - (1+2) * (3+4)
  - Postfix
    - 12+34+*

Example

- Infix to Postfix
  - Infix
    - (1+2) * (3+4)
  - Postfix
    - 12+34+*

12+34+*

Example

- Infix to Postfix
  - Infix
    - (1+2) * (3+4)
  - Postfix
    - 12+34+*

12+34+*

12+34+*

Example

- Infix to Postfix
  - Infix
    - (1+2) * (3+4)
  - Postfix
    - 12+34+*

12+34+*

12+34+*

334+*

Example

- Infix to Postfix
  - Infix
    - (1+2) * (3+4)
  - Postfix
    - 12+34+*

12+34+*

334+*

37*

Example

- Infix to Postfix
  - Infix
    - (1+2) * (3+4)
  - Postfix
    - 12+34+*

12+34+*

334+*

37*

21

Stack Example

- Example: stk_exp/postfix_cal.c

```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```

Stack Example

- Example: stk_exp/postfix_cal.c

```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```

Stack Example

- Example: stk_exp/postfix_cal.c

```c
void eval(char* postfix) {
    double stack[80] = {0.0};
    char temp[2];
    char token;
    int top = 0, i = 0;

    temp[1] = '\0';

    while(1) {
        token = postfix[i];
        switch(token) {
            case '\0':
                printf("ans = %f\n", stack[top]);
                return;
            case '+': case '-': case '*': case '/':
                stack[top-1] =
                    cal(stack[top-1], token, stack[top]);
                top--;
                break;
            default:
                if(top < sizeof(stack) / sizeof(float)) {
                    temp[0] = postfix[i];
                    top++;
                    stack[top] = atof(temp);
                }
                break;
        }
        i++;
    }
}
```

1 | 2 | + | 3 | 4 | + | * | '\0'

Stack Example

- Example: stk_exp/postfix_cal.c

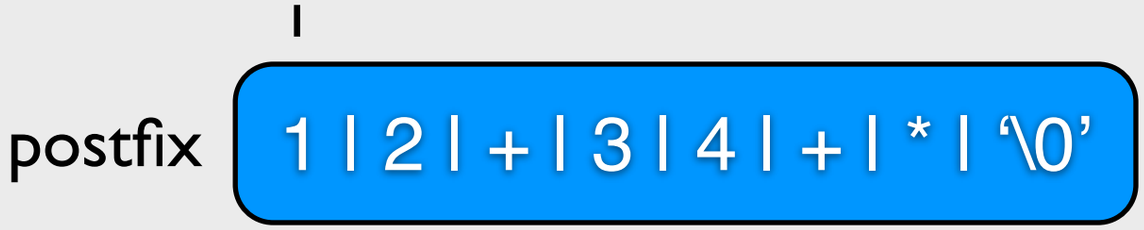
```c
void eval(char* postfix) {
    double stack[80] = {0.0};
    char temp[2];
    char token;
    int top = 0, i = 0;

    temp[1] = '\0';

    while(1) {
        token = postfix[i];
        switch(token) {
            case '\0':
                printf("ans = %f\n", stack[top]);
                return;
            case '+': case '-': case '*': case '/':
                stack[top-1] =
                    cal(stack[top-1], token, stack[top]);
                top--;
                break;
            default:
                if(top < sizeof(stack) / sizeof(float)) {
                    temp[0] = postfix[i];
                    top++;
                    stack[top] = atof(temp);
                }
                break;
        }
        i++;
    }
}
```

postfix 1 | 2 | + | 3 | 4 | + | * | '\0'

Stack Example

- Example: stk_exp/postfix_cal.c

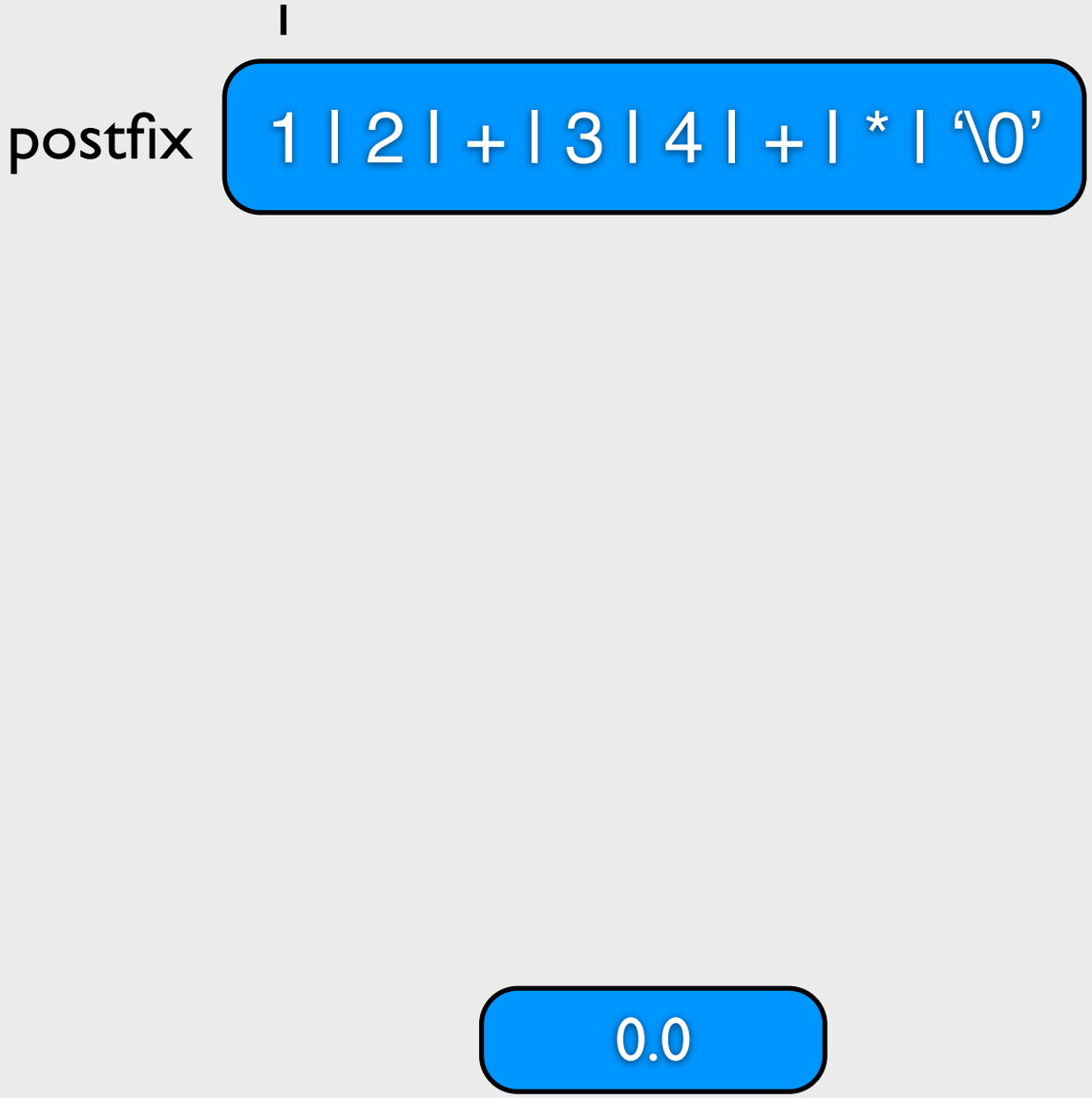
```c
void eval(char* postfix) {
    double stack[80] = {0.0};
    char temp[2];
    char token;
    int top = 0, i = 0;

    temp[1] = '\0';

    while(1) {
        token = postfix[i];
        switch(token) {
            case '\0':
                printf("ans = %f\n", stack[top]);
                return;
            case '+': case '-': case '*': case '/':
                stack[top-1] =
                    cal(stack[top-1], token, stack[top]);
                top--;
                break;
            default:
                if(top < sizeof(stack) / sizeof(float)) {
                    temp[0] = postfix[i];
                    top++;
                    stack[top] = atof(temp);
                }
                break;
        }
        i++;
    }
}
```



Stack Example

- Example: stk_exp/postfix_cal.c

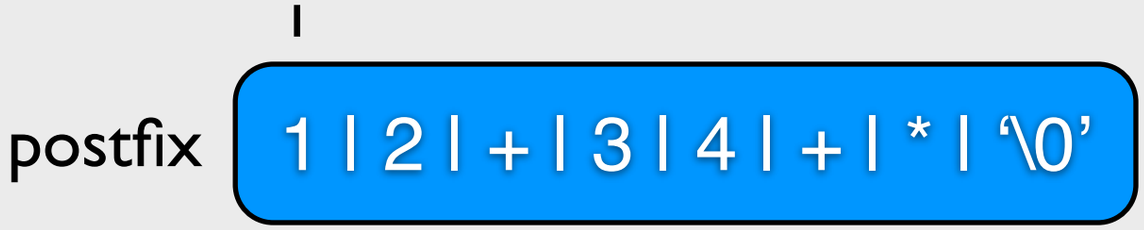
```c
void eval(char* postfix) {
    double stack[80] = {0.0};
    char temp[2];
    char token;
    int top = 0, i = 0;

    temp[1] = '\0';

    while(1) {
        token = postfix[i];
        switch(token) {
            case '\0':
                printf("ans = %f\n", stack[top]);
                return;
            case '+': case '-': case '*': case '/':
                stack[top-1] =
                    cal(stack[top-1], token, stack[top]);
                top--;
                break;
            default:
                if(top < sizeof(stack) / sizeof(float)) {
                    temp[0] = postfix[i];
                    top++;
                    stack[top] = atof(temp);
                }
                break;
        }
        i++;
    }
}
```

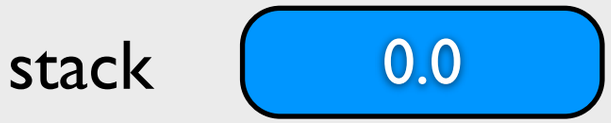


Stack Example

- Example: stk_exp/postfix_cal.c

```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```

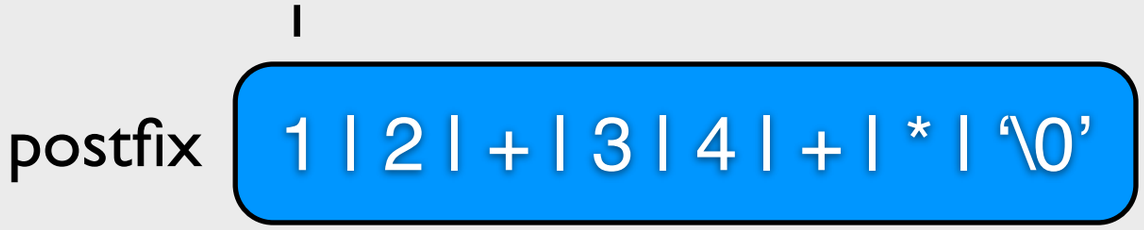




Stack Example

- Example: stk_exp/postfix_cal.c

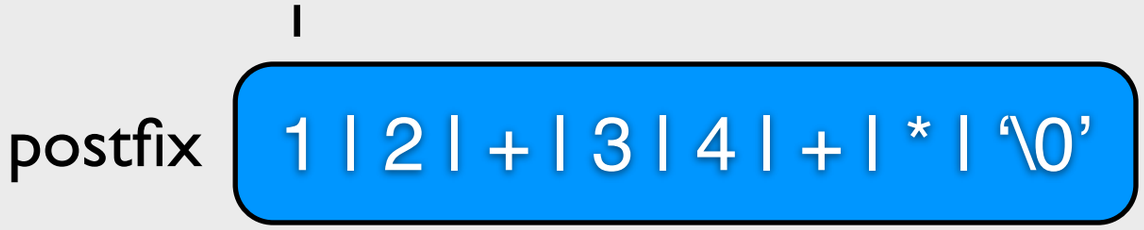
```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```



Stack Example

- Example: stk_exp/postfix_cal.c

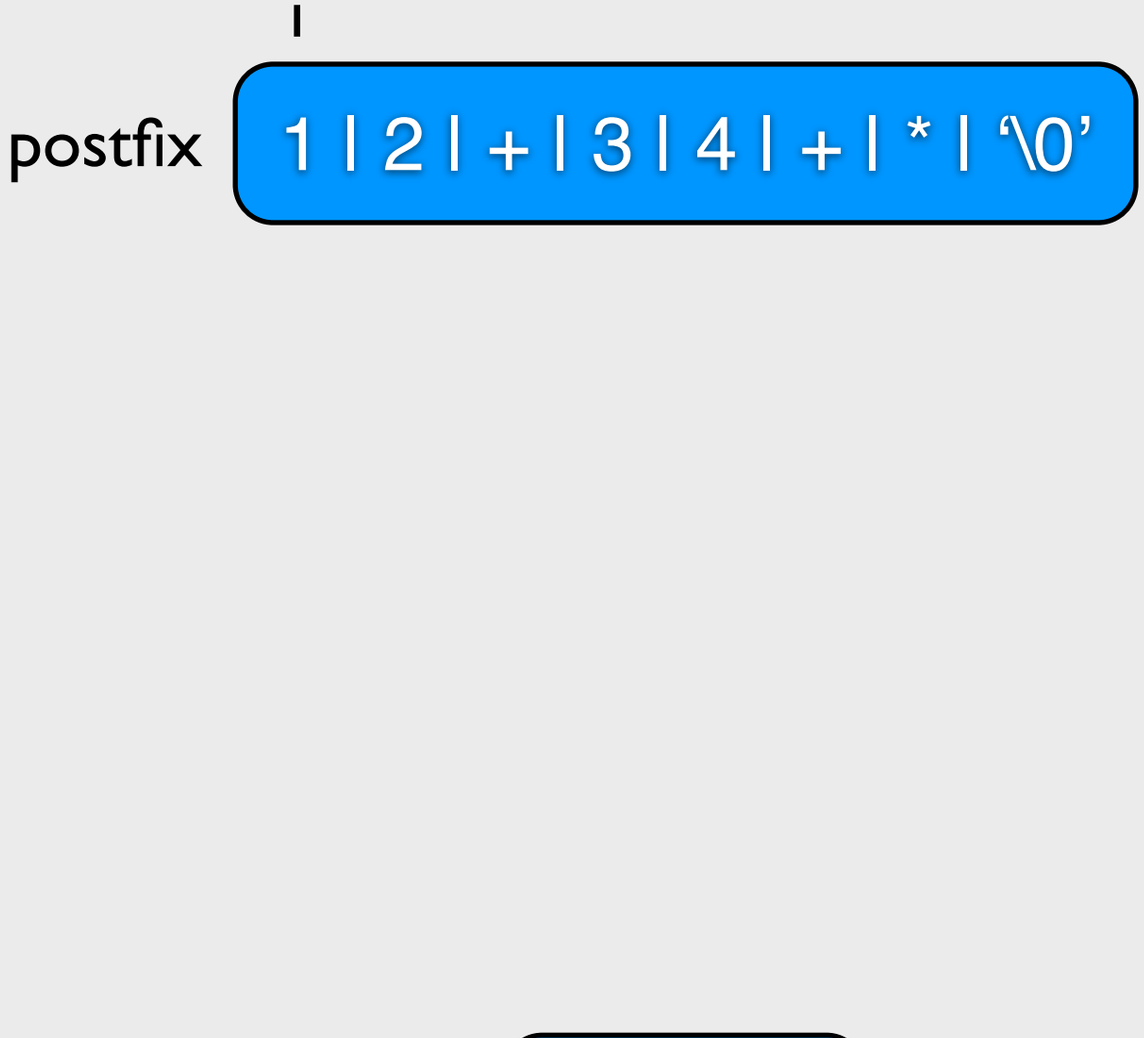
```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```



Stack Example

- Example: stk_exp/postfix_cal.c

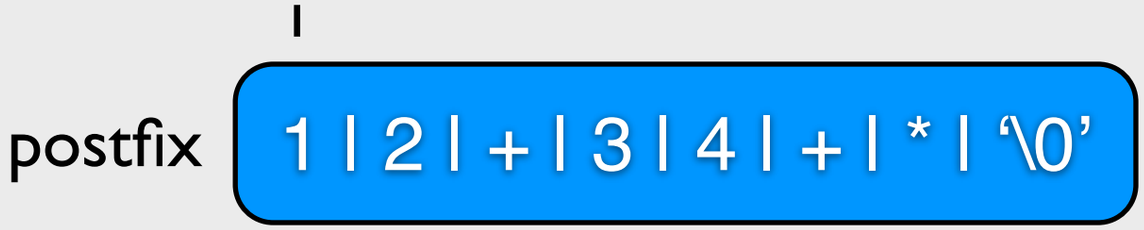
```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```



Stack Example

- Example: stk_exp/postfix_cal.c

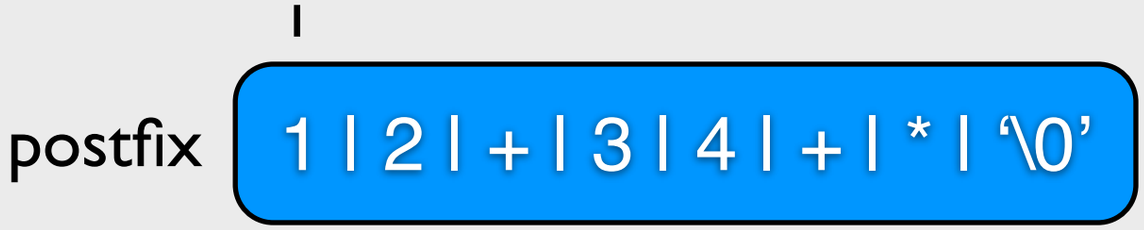
```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```



Stack Example

- Example: stk_exp/postfix_cal.c

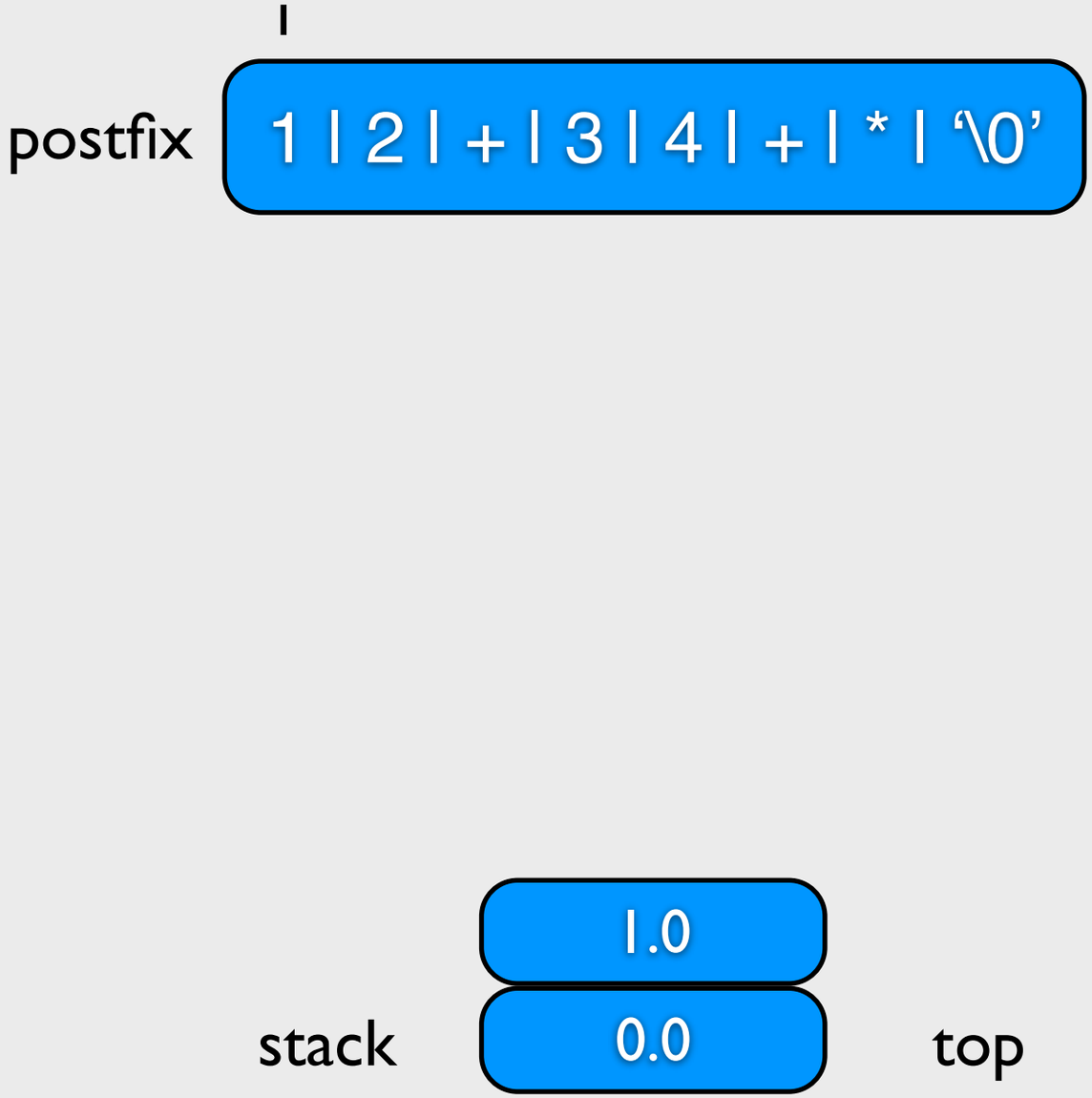
```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```



Stack Example

- Example: stk_exp/postfix_cal.c

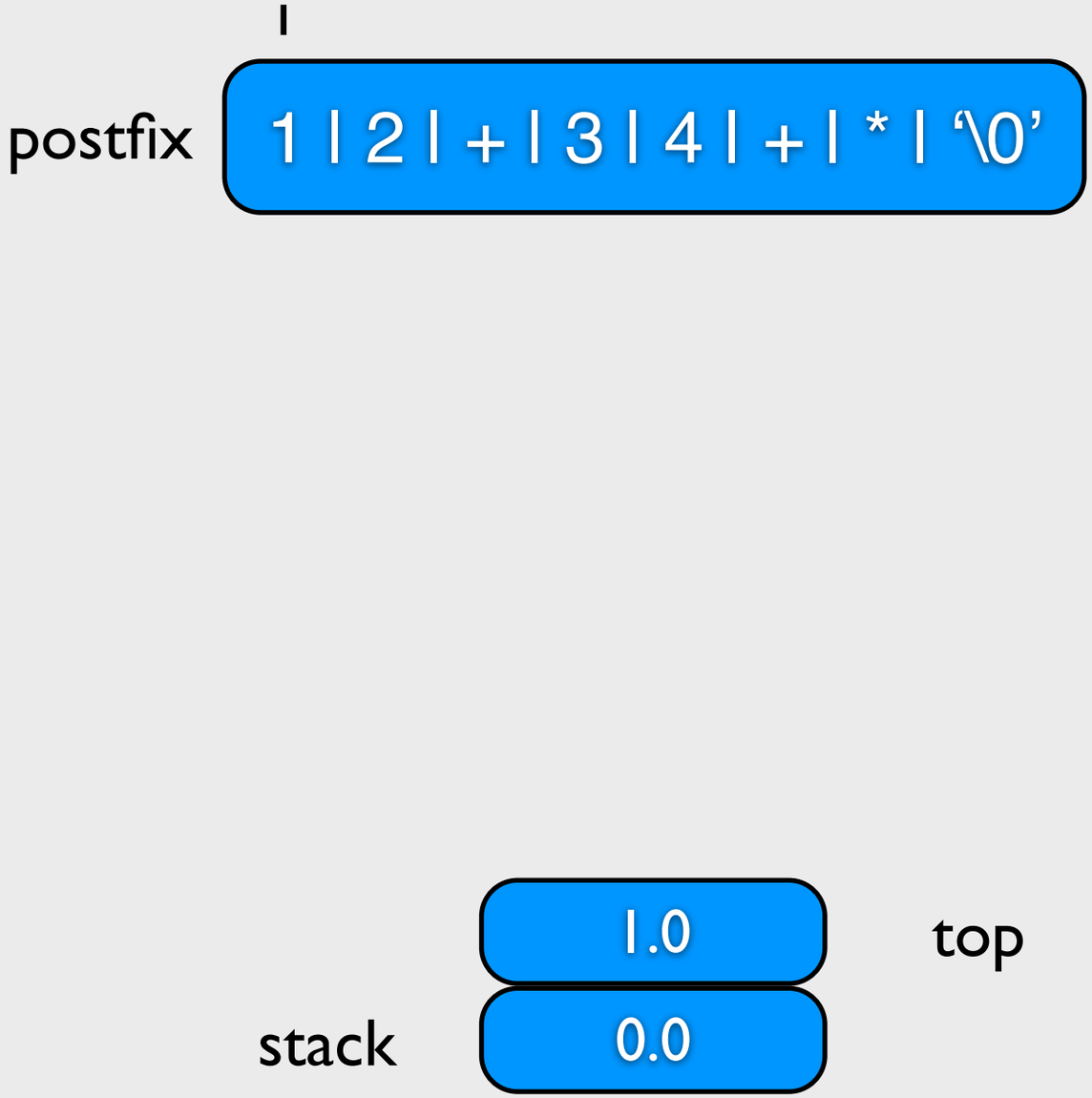
```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```



Stack Example

- Example: stk_exp/postfix_cal.c

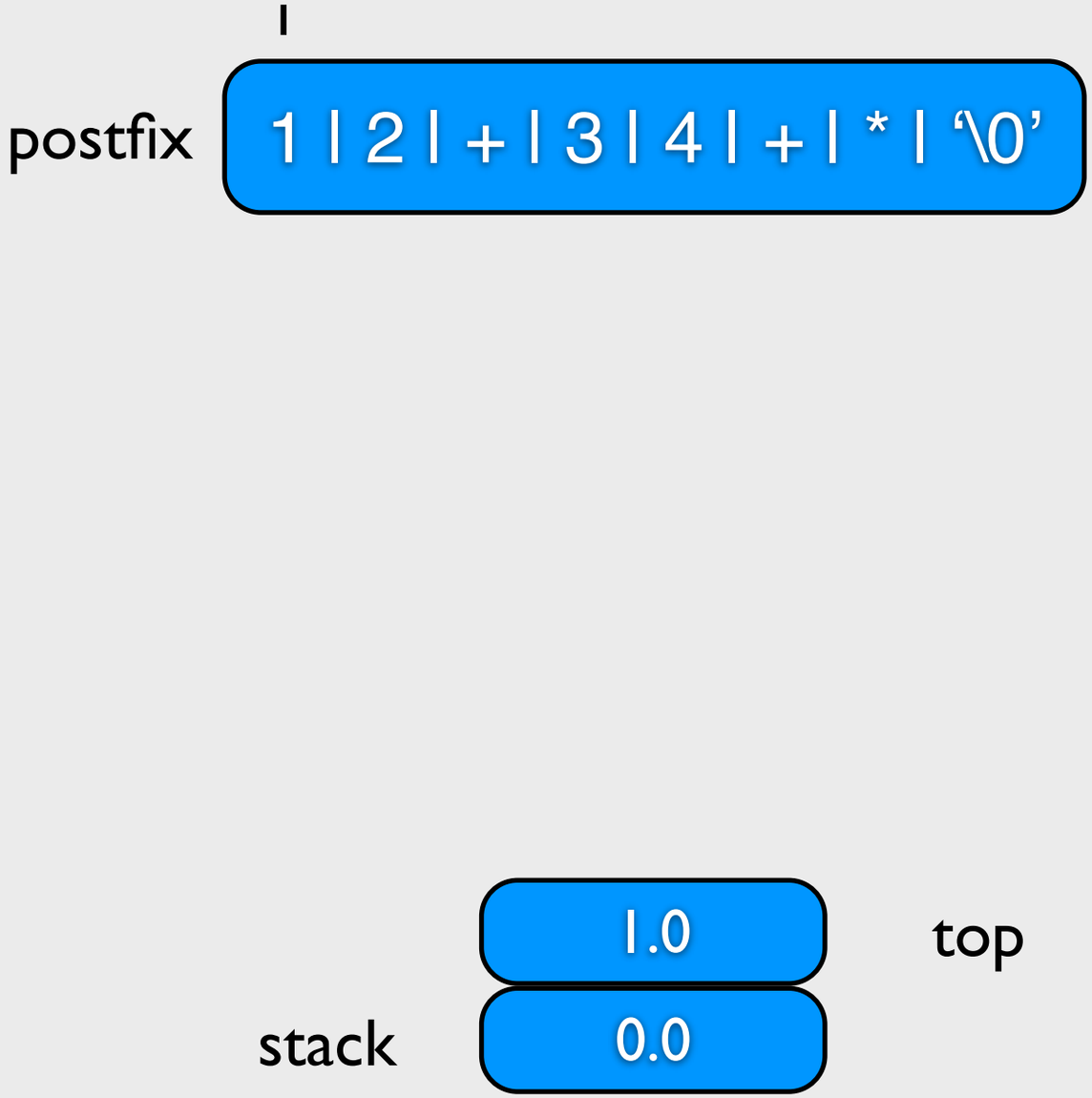
```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```



Stack Example

- Example: stk_exp/postfix_cal.c

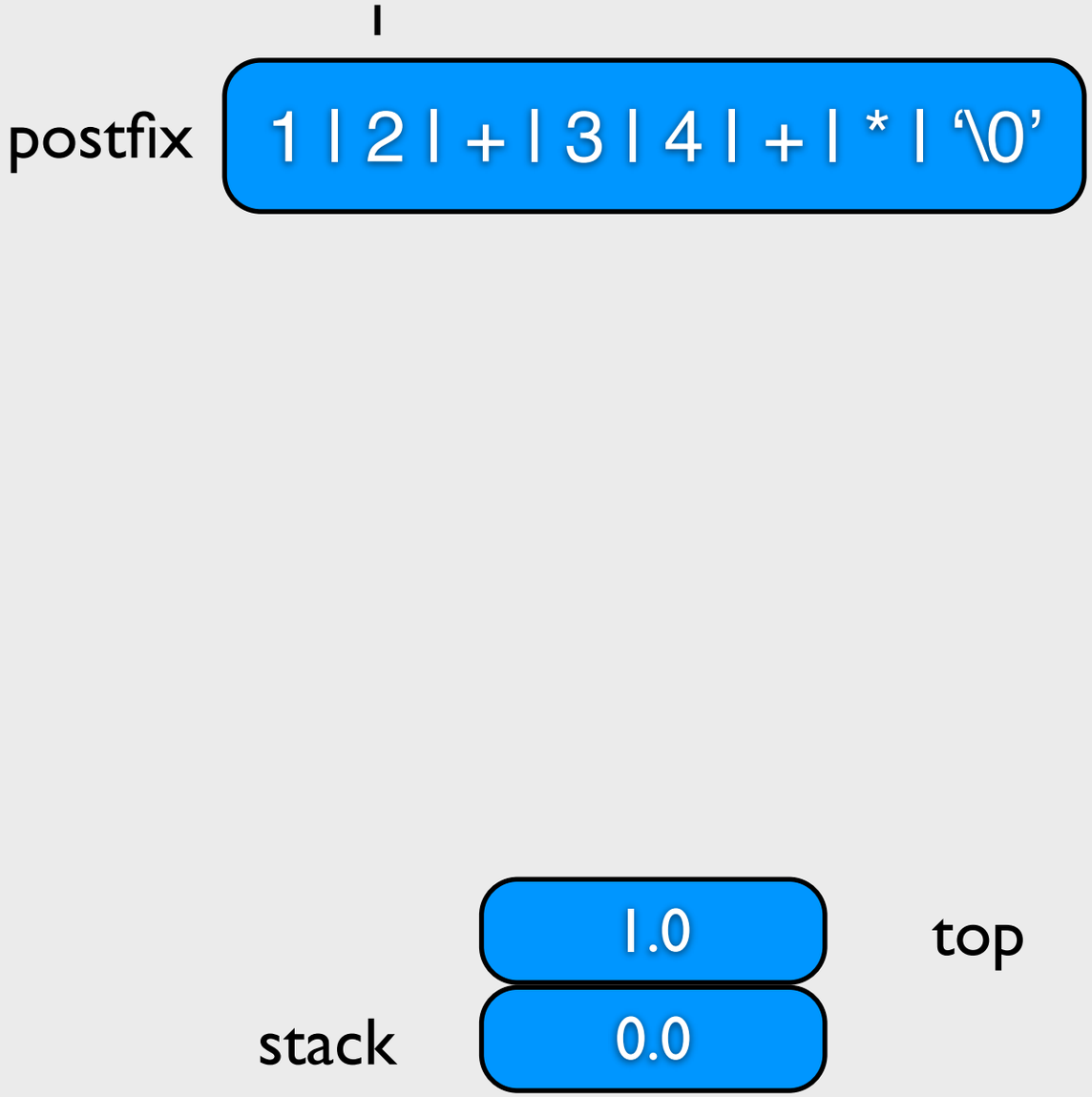
```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```



Stack Example

- Example: stk_exp/postfix_cal.c

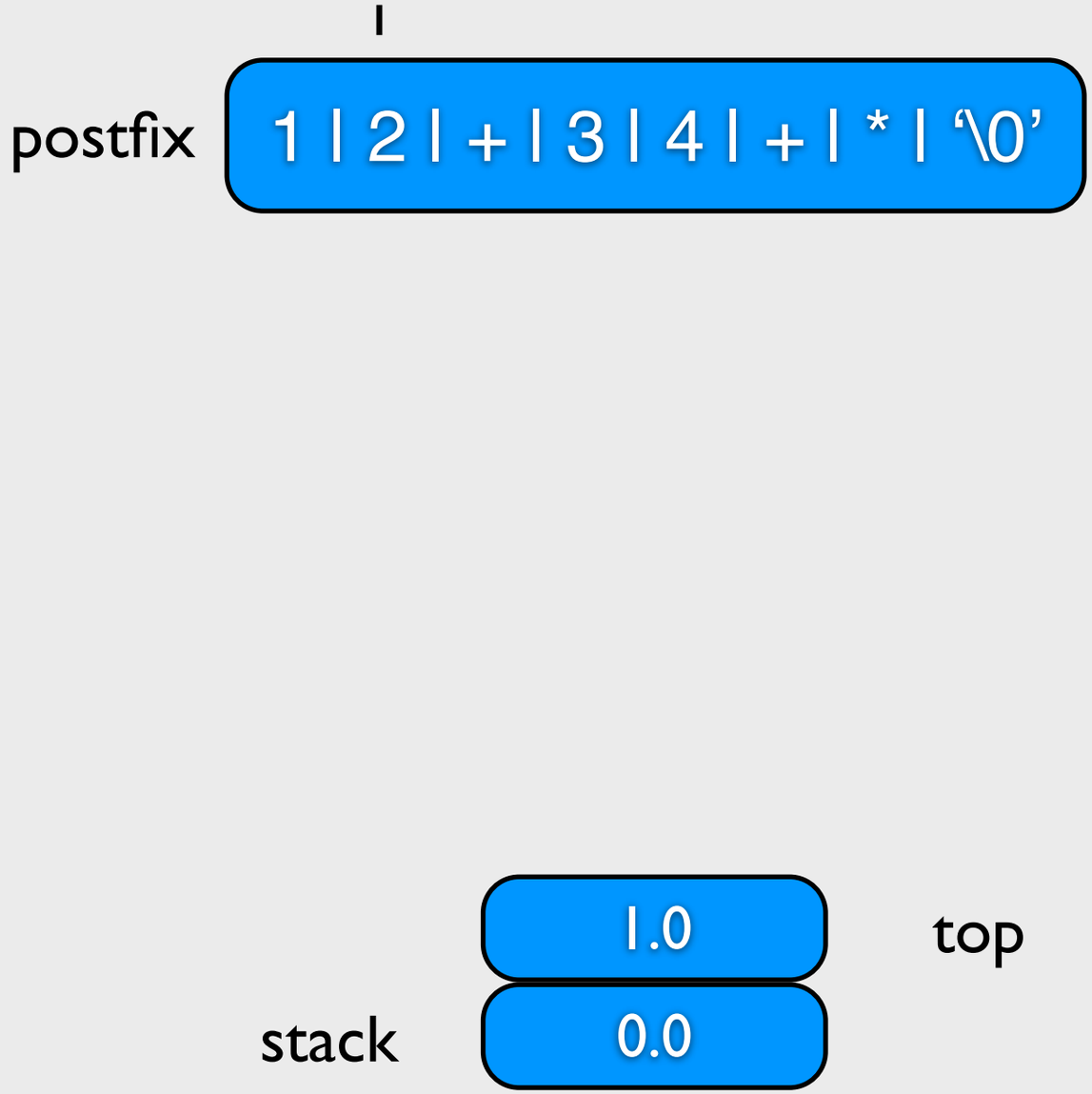
```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```



Stack Example

- Example: stk_exp/postfix_cal.c

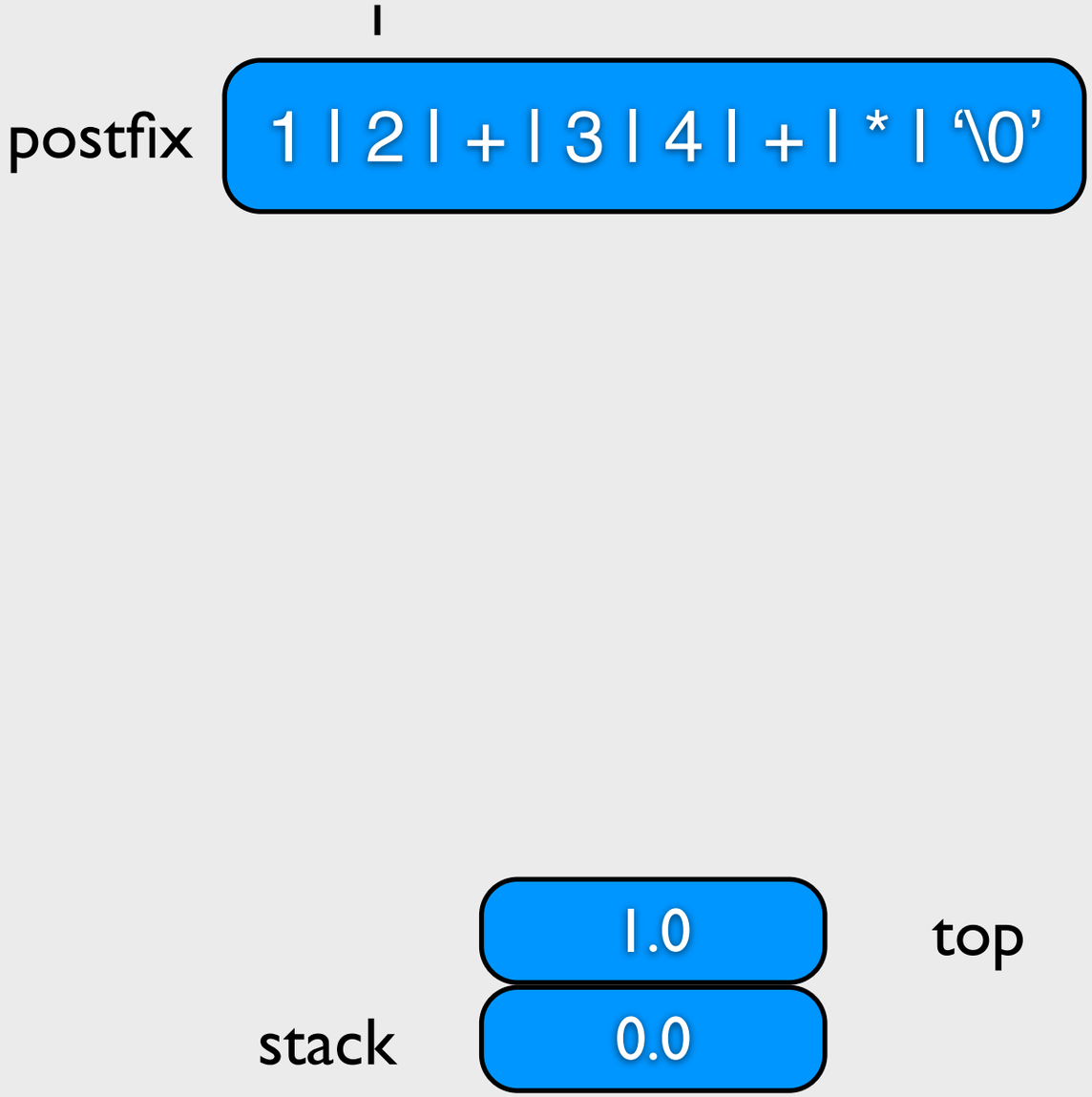
```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```



Stack Example

- Example: stk_exp/postfix_cal.c

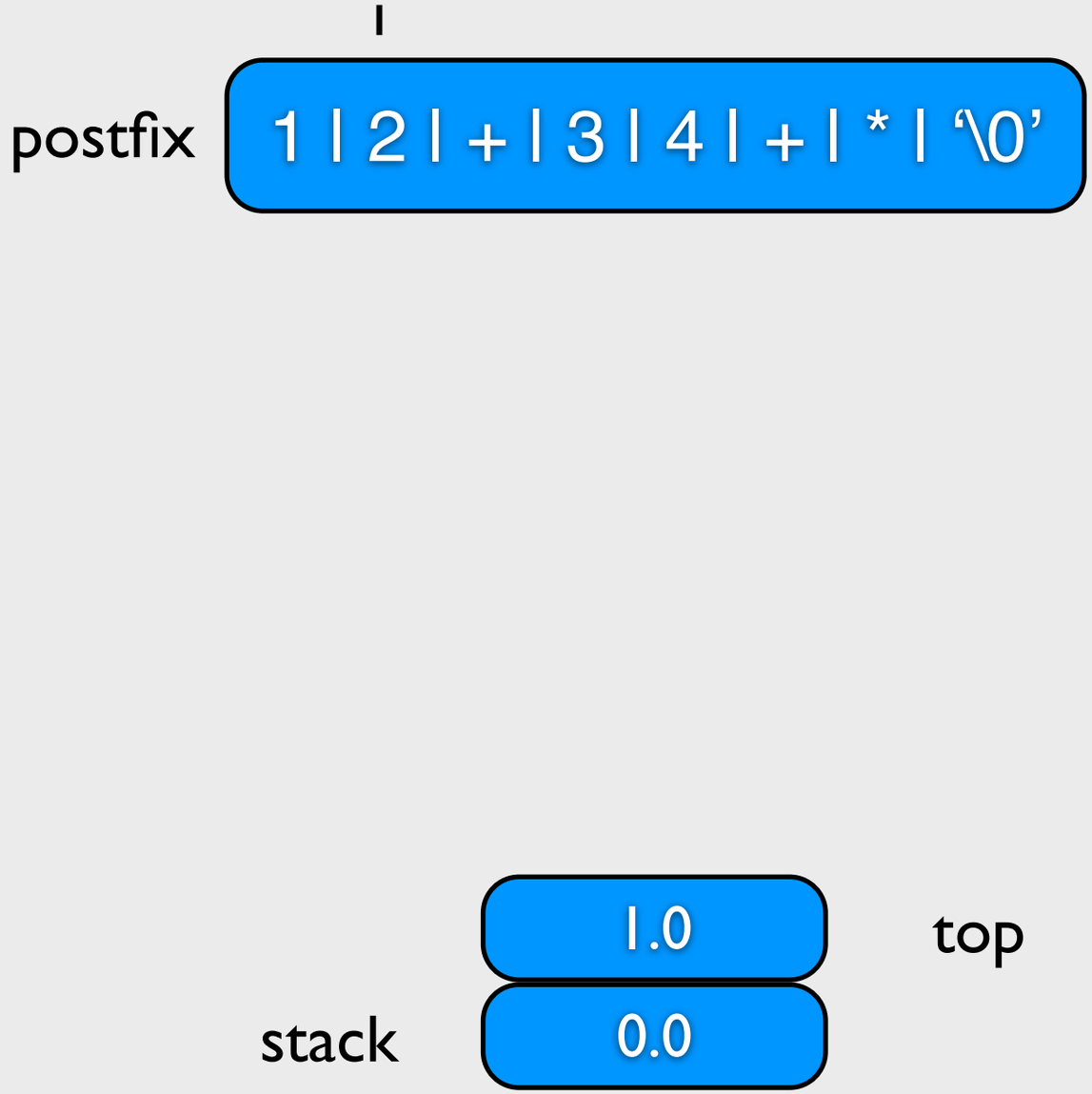
```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```



Stack Example

- Example: stk_exp/postfix_cal.c

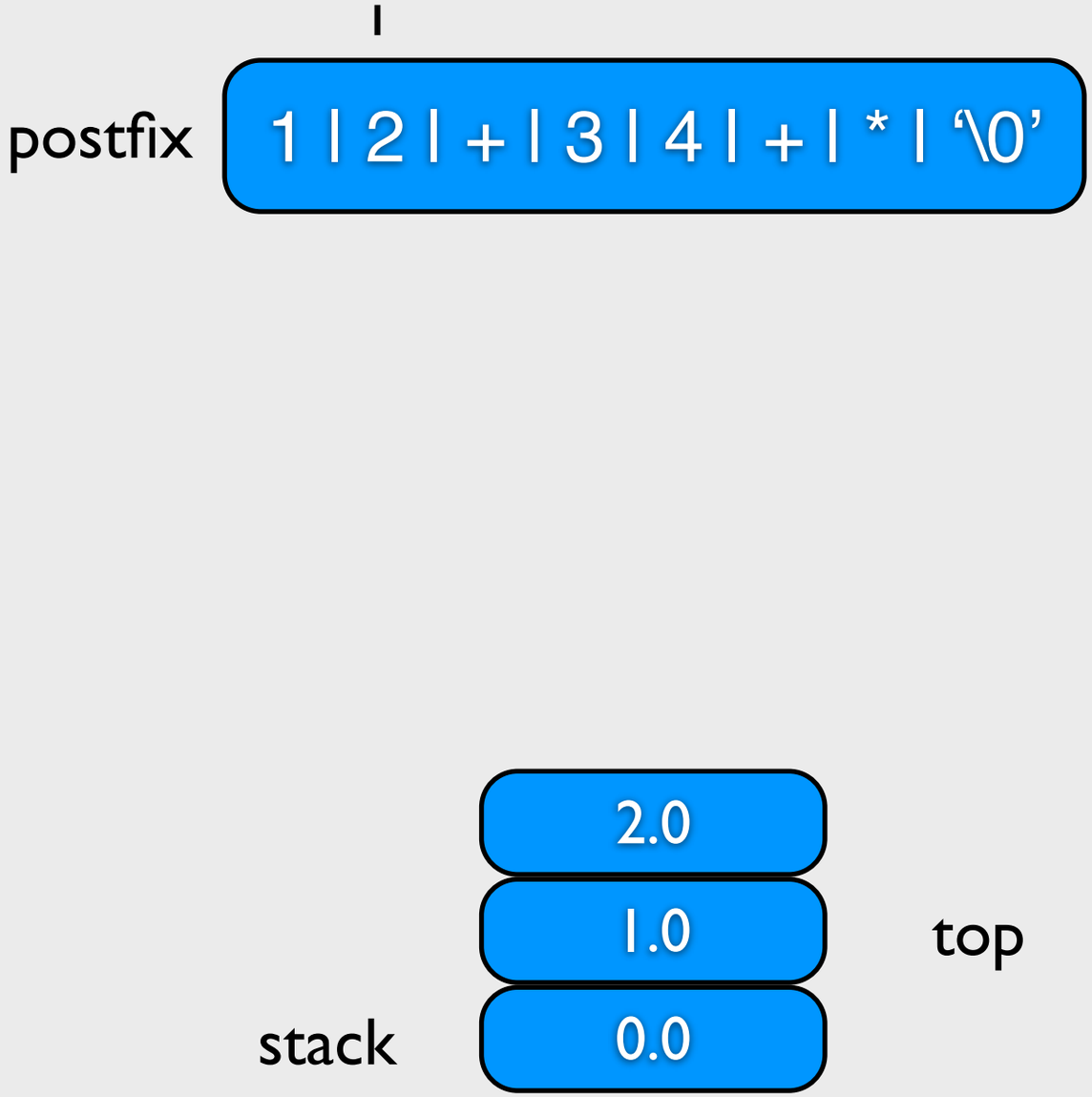
```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```



Stack Example

- Example: stk_exp/postfix_cal.c

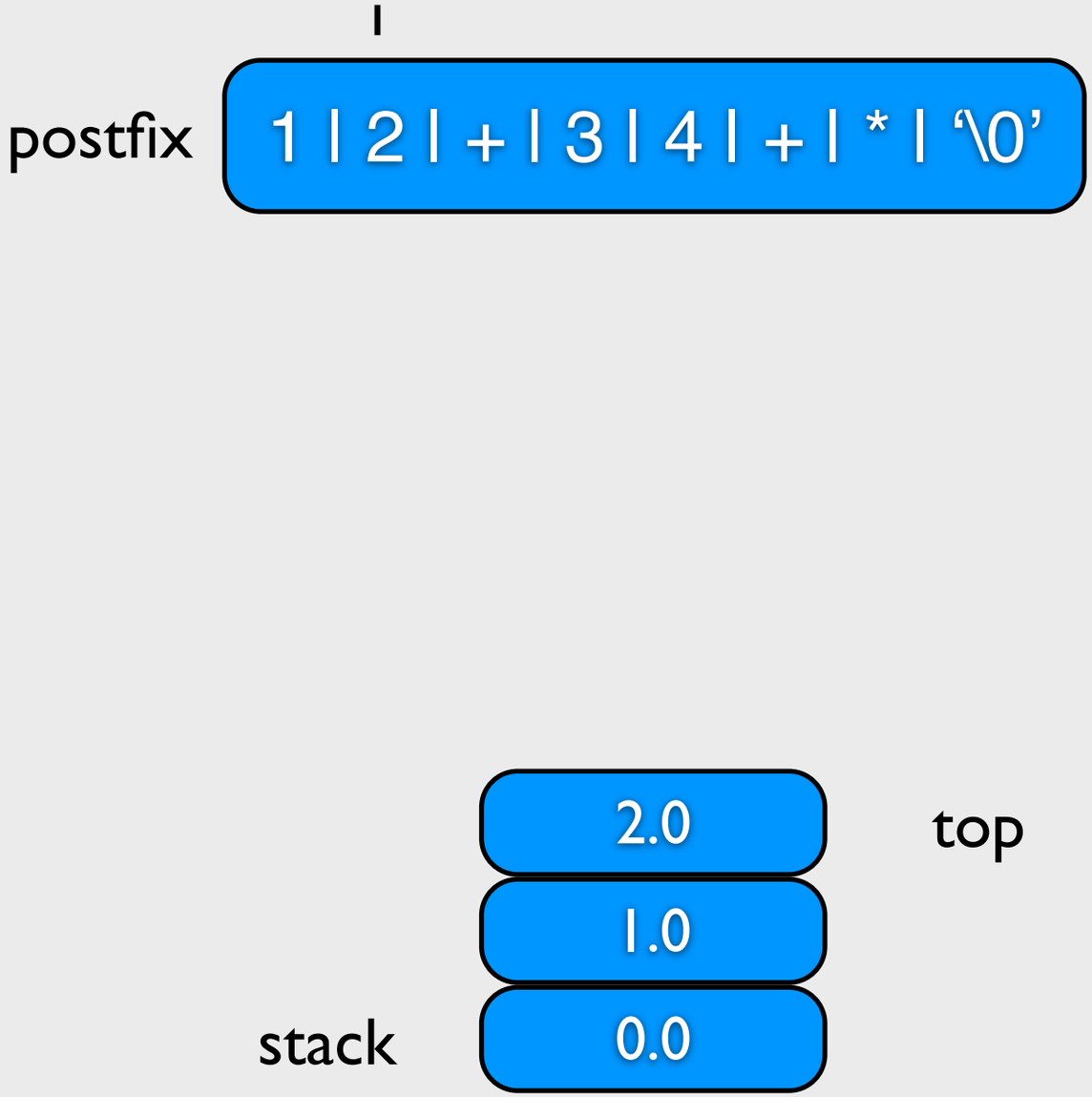
```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```



Stack Example

- Example: stk_exp/postfix_cal.c

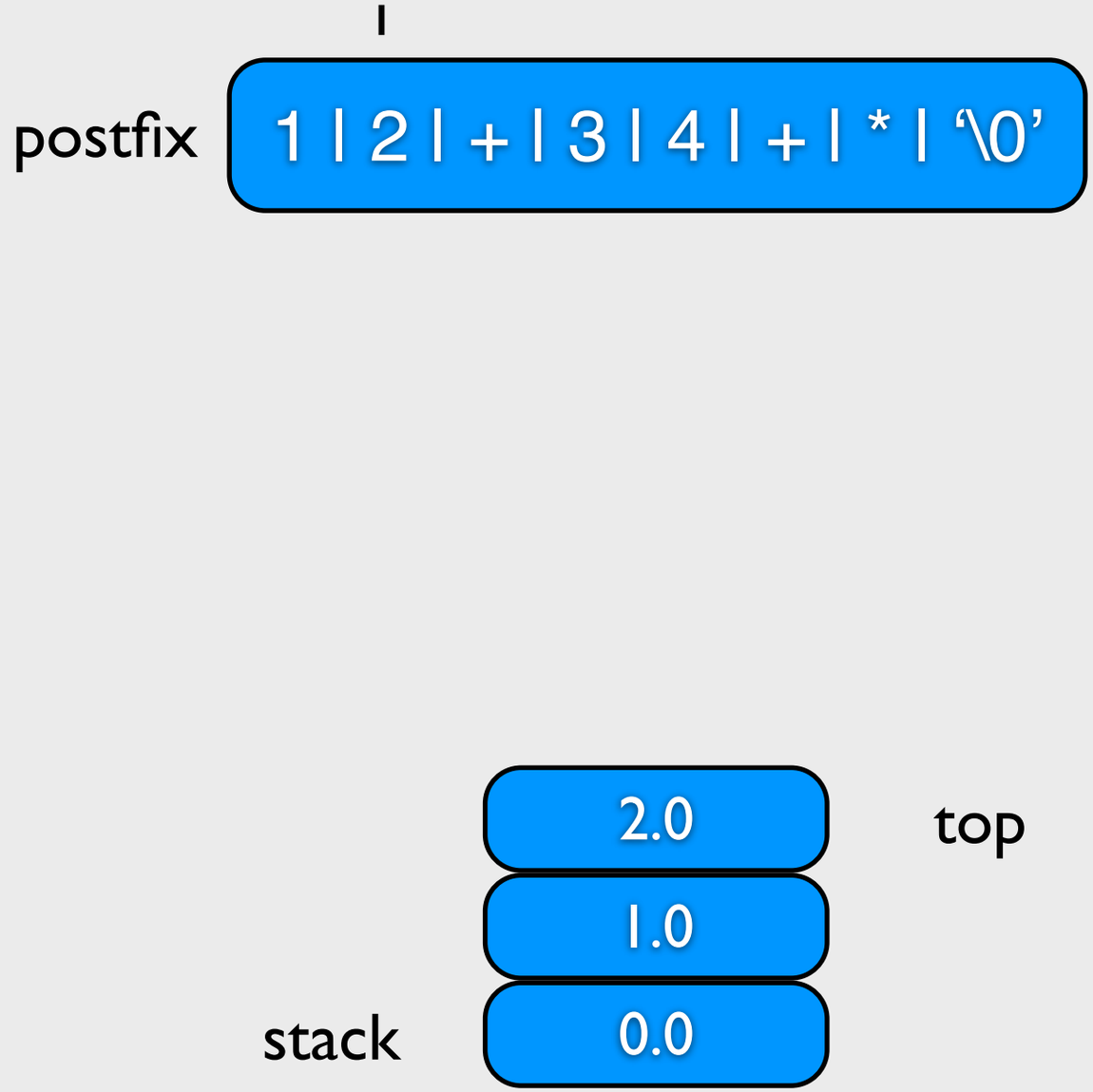
```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```



Stack Example

- Example: stk_exp/postfix_cal.c

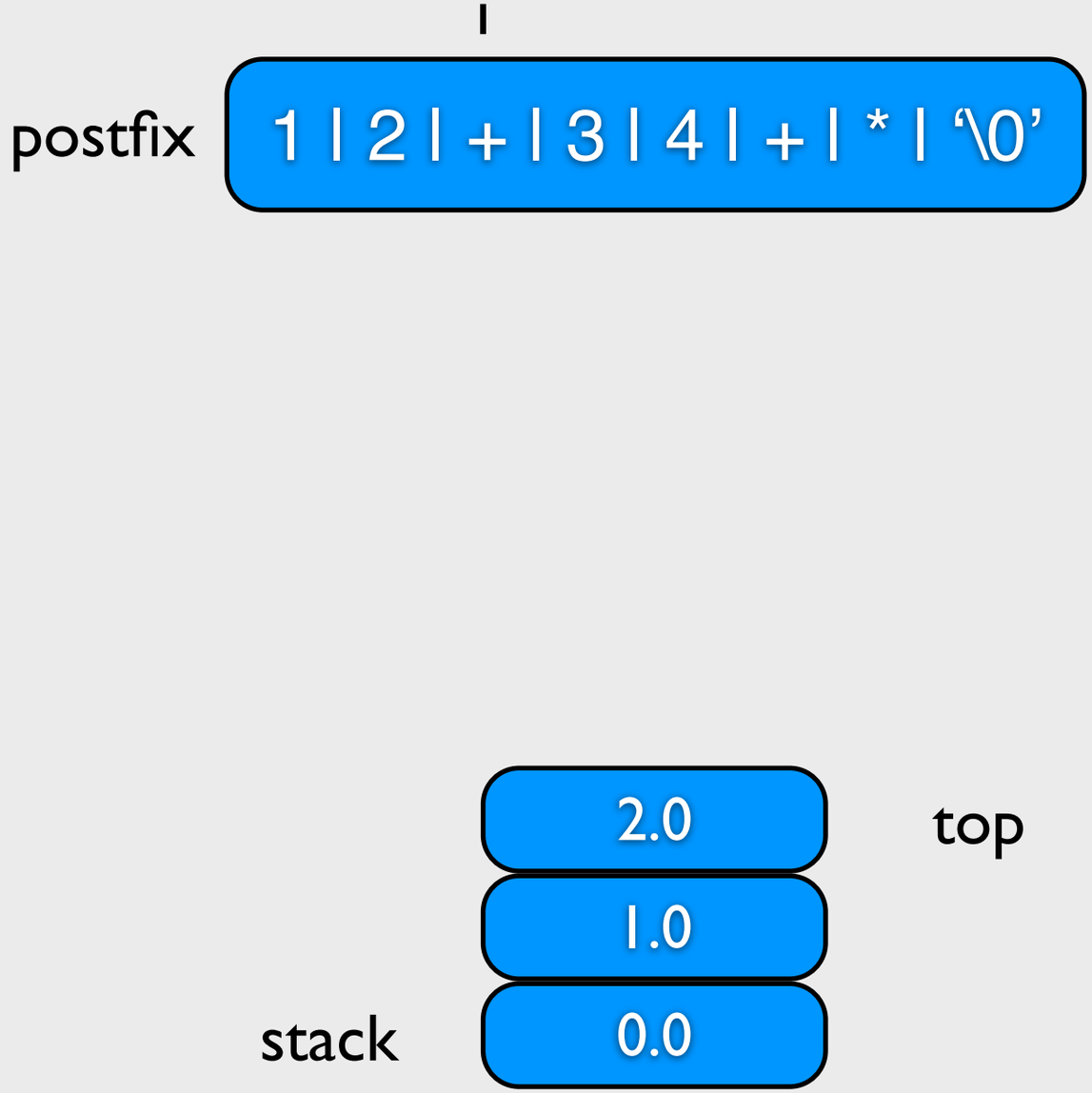
```c
void eval(char* postfix) {
    double stack[80] = {0.0};
    char temp[2];
    char token;
    int top = 0, i = 0;

    temp[1] = '\0';

    while(1) {
        token = postfix[i];
        switch(token) {
            case '\0':
                printf("ans = %f\n", stack[top]);
                return;
            case '+': case '-': case '*': case '/':
                stack[top-1] =
                    cal(stack[top-1], token, stack[top]);
                top--;
                break;
            default:
                if(top < sizeof(stack) / sizeof(float)) {
                    temp[0] = postfix[i];
                    top++;
                    stack[top] = atof(temp);
                }
                break;
        }
        i++;
    }
}
```



Stack Example

- Example: stk_exp/postfix_cal.c

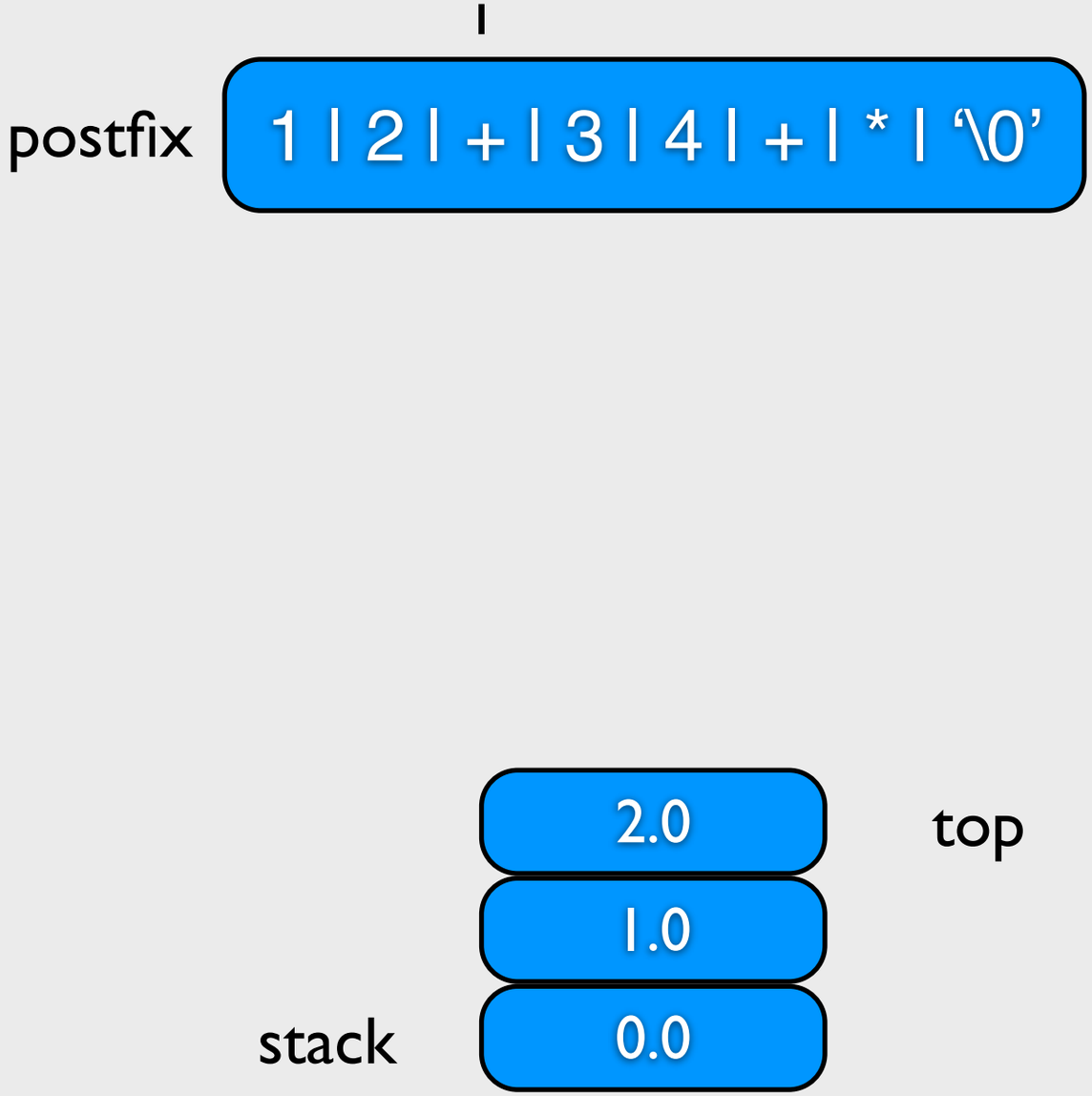
```c
void eval(char* postfix) {
    double stack[80] = {0.0};
    char temp[2];
    char token;
    int top = 0, i = 0;

    temp[1] = '\0';

    while(1) {
        token = postfix[i];
        switch(token) {
            case '\0':
                printf("ans = %f\n", stack[top]);
                return;
            case '+': case '-': case '*': case '/':
                stack[top-1] =
                    cal(stack[top-1], token, stack[top]);
                top--;
                break;
            default:
                if(top < sizeof(stack) / sizeof(float)) {
                    temp[0] = postfix[i];
                    top++;
                    stack[top] = atof(temp);
                }
                break;
        }
        i++;
    }
}
```



Stack Example

- Example: stk_exp/postfix_cal.c

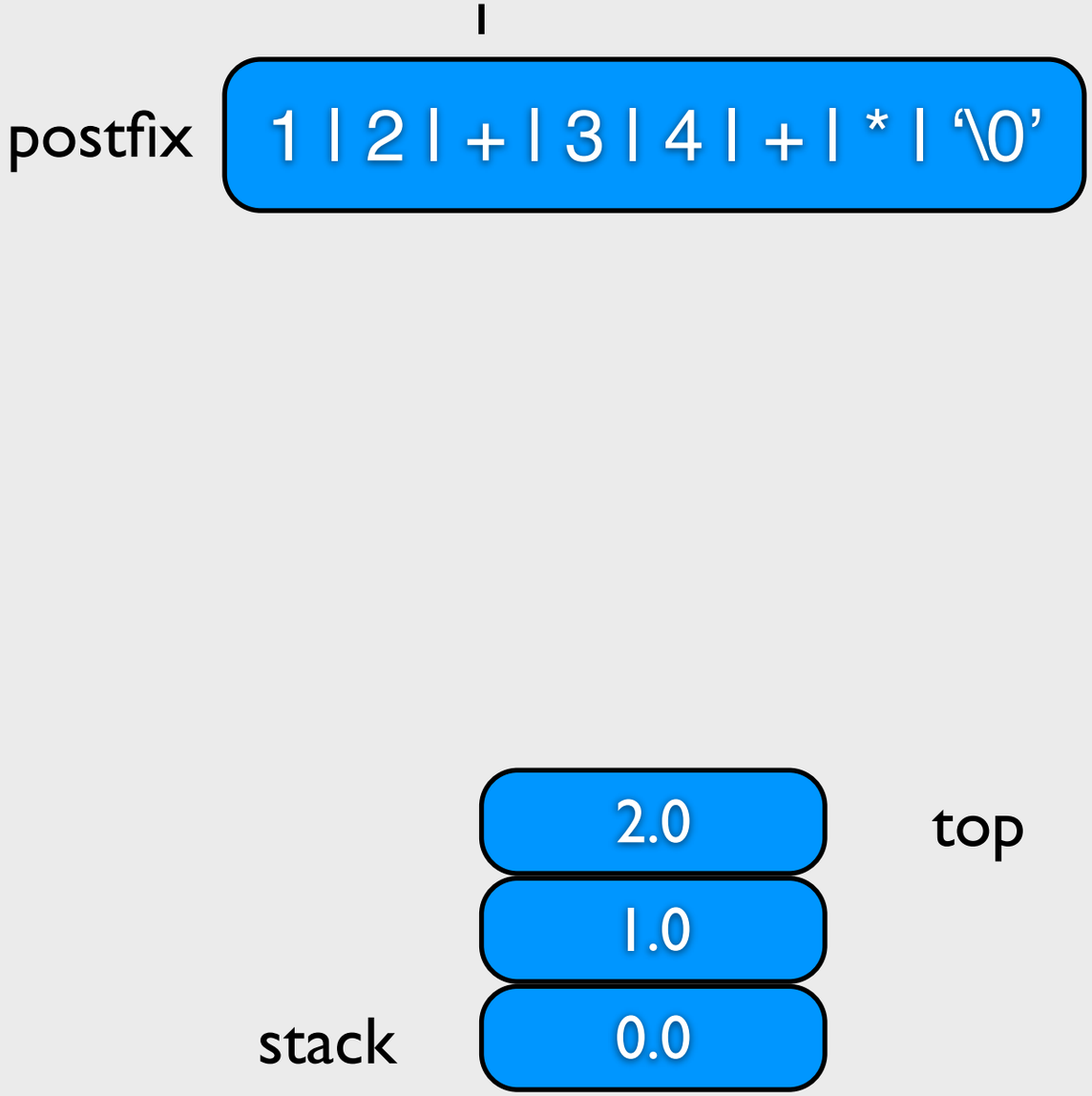
```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```



Stack Example

- Example: stk_exp/postfix_cal.c

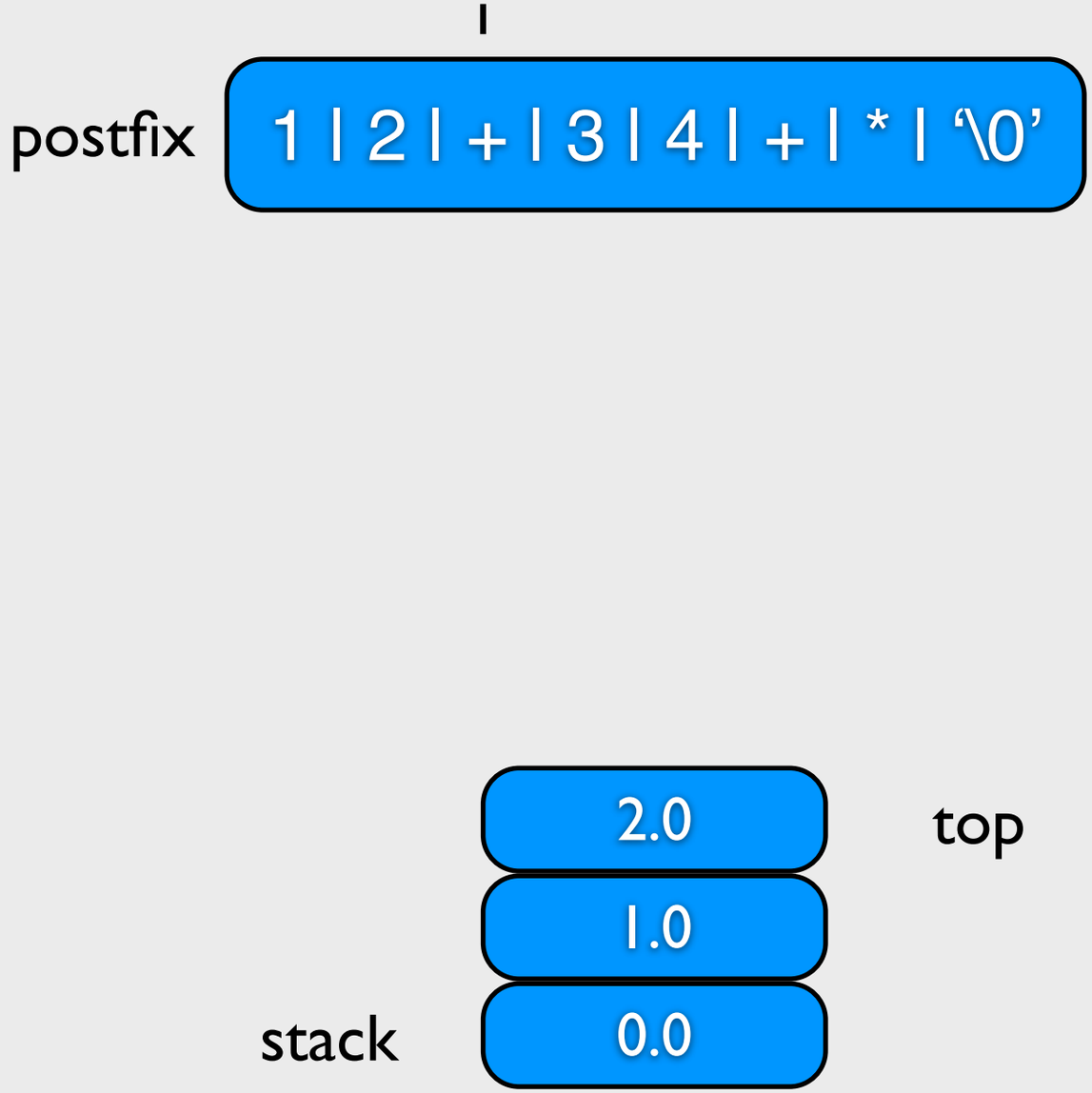
```c
void eval(char* postfix) {
    double stack[80] = {0.0};
    char temp[2];
    char token;
    int top = 0, i = 0;

    temp[1] = '\0';

    while(1) {
        token = postfix[i];
        switch(token) {
            case '\0':
                printf("ans = %f\n", stack[top]);
                return;
            case '+': case '-': case '*': case '/':
                stack[top-1] =
                    cal(stack[top-1], token, stack[top]);
                top--;
                break;
            default:
                if(top < sizeof(stack) / sizeof(float)) {
                    temp[0] = postfix[i];
                    top++;
                    stack[top] = atof(temp);
                }
                break;
        }
        i++;
    }
}
```



Stack Example

- Example: stk_exp/postfix_cal.c

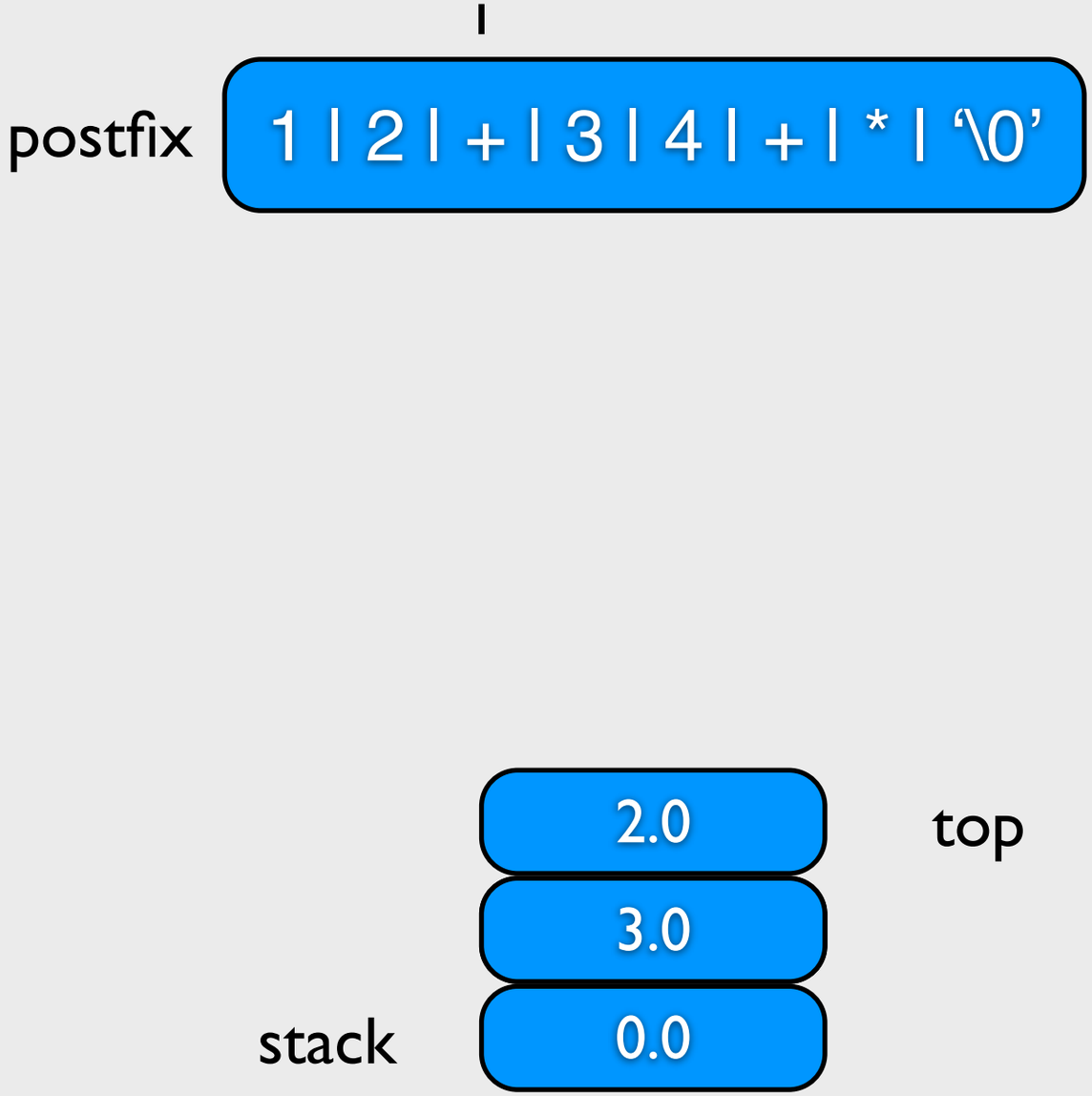
```c
void eval(char* postfix) {
    double stack[80] = {0.0};
    char temp[2];
    char token;
    int top = 0, i = 0;

    temp[1] = '\0';

    while(1) {
        token = postfix[i];
        switch(token) {
            case '\0':
                printf("ans = %f\n", stack[top]);
                return;
            case '+': case '-': case '*': case '/':
                stack[top-1] =
                    cal(stack[top-1], token, stack[top]);
                top--;
                break;
            default:
                if(top < sizeof(stack) / sizeof(float)) {
                    temp[0] = postfix[i];
                    top++;
                    stack[top] = atof(temp);
                }
                break;
        }
        i++;
    }
}
```



Stack Example

- Example: stk_exp/postfix_cal.c

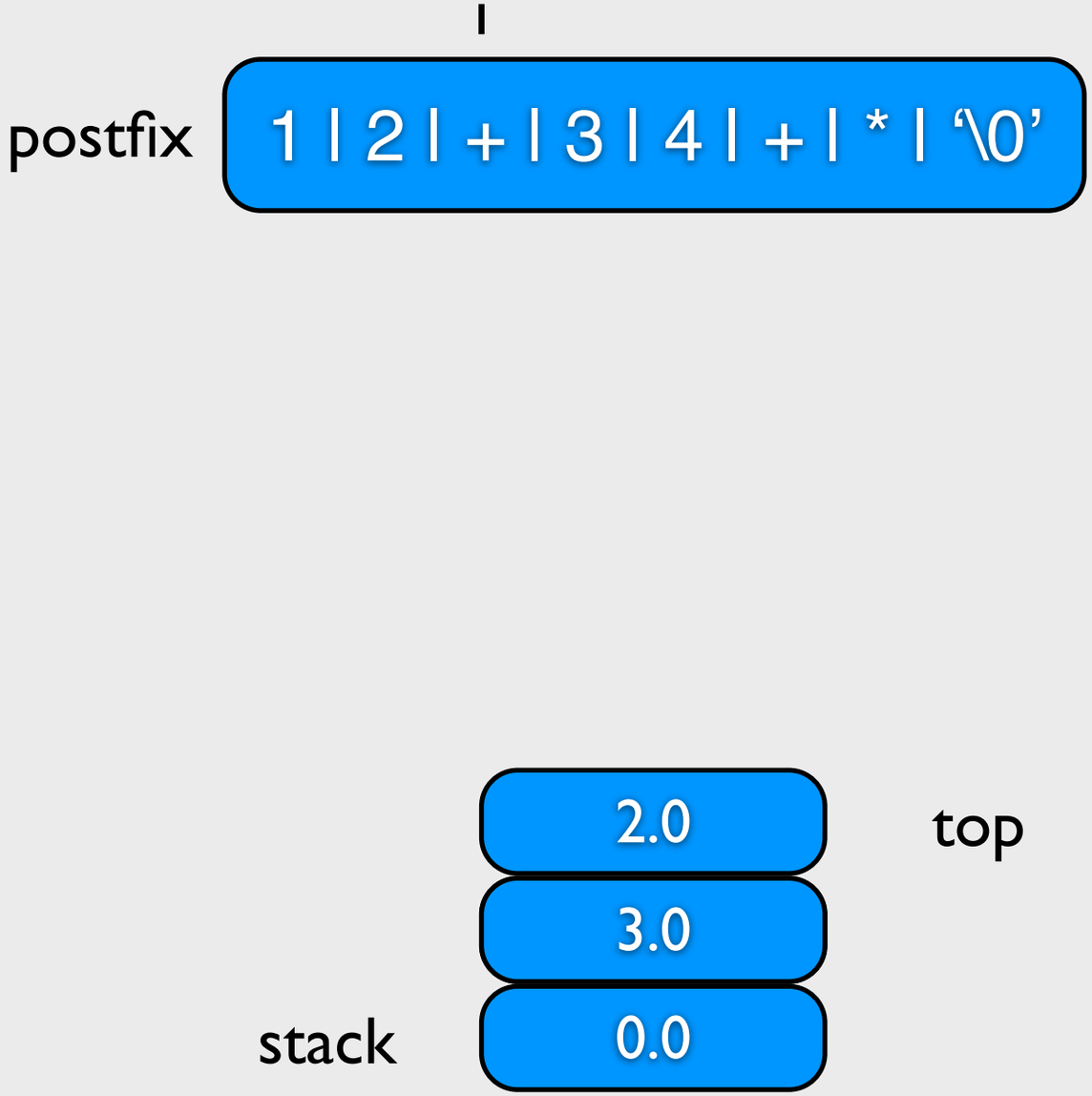
```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```



Stack Example

- Example: stk_exp/postfix_cal.c

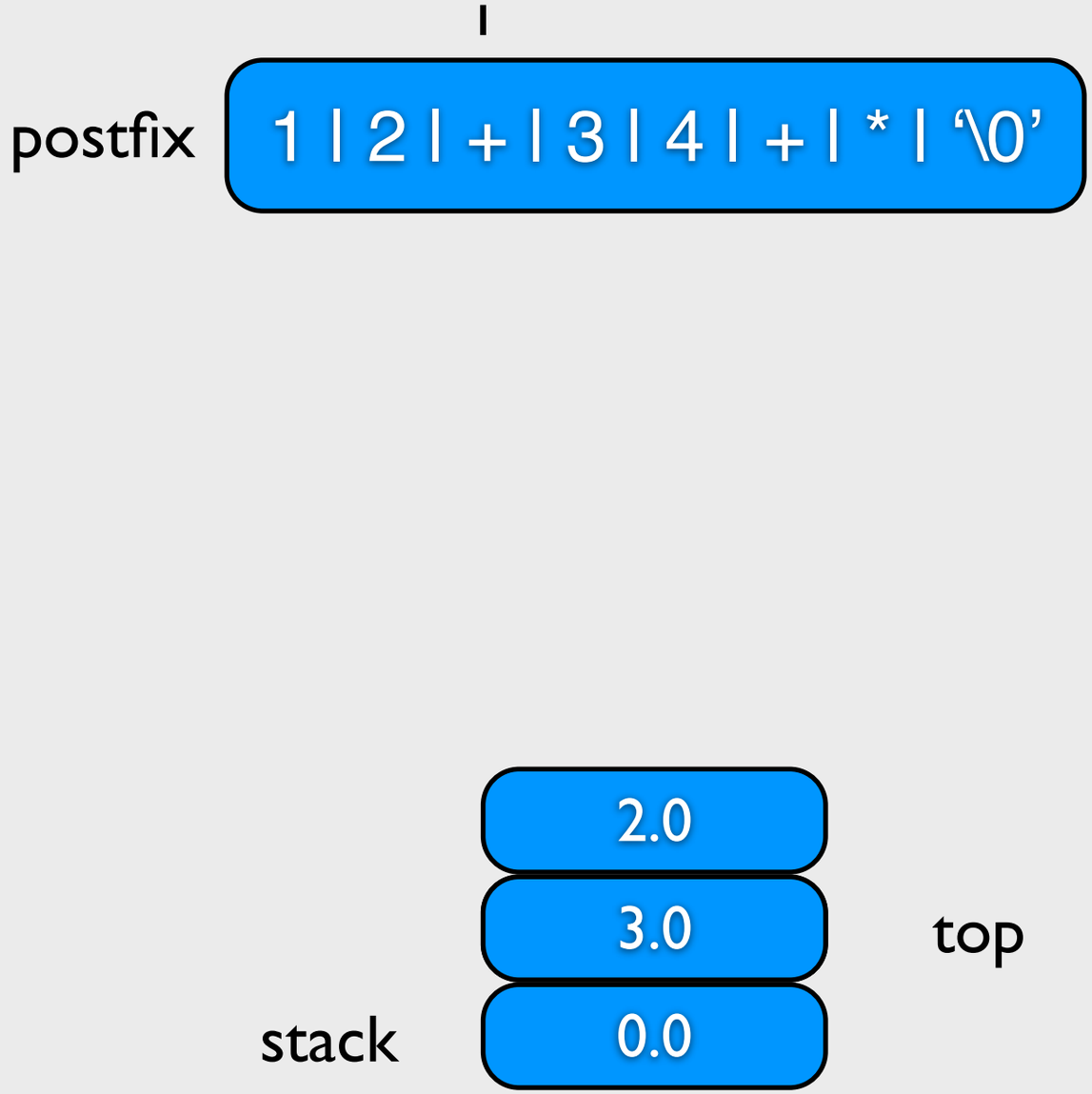
```
17 void eval(char* postfix) {
18     double stack[80] = {0.0};
19     char temp[2];
20     char token;
21     int top = 0, i = 0;
22
23     temp[1] = '\0';
24
25     while(1) {
26         token = postfix[i];
27         switch(token) {
28             case '\0':
29                 printf("ans = %f\n", stack[top]);
30                 return;
31             case '+': case '-': case '*': case '/':
32                 stack[top-1] =
33                     cal(stack[top-1], token, stack[top]);
34                 top--;
35                 break;
36             default:
37                 if(top < sizeof(stack) / sizeof(float)) {
38                     temp[0] = postfix[i];
39                     top++;
40                     stack[top] = atof(temp);
41                 }
42                 break;
43         }
44         i++;
45     }
46 }
```



Stack Example

- Example: stk_exp/postfix_cal.c

```c
void eval(char* postfix) {
    double stack[80] = {0.0};
    char temp[2];
    char token;
    int top = 0, i = 0;

    temp[1] = '\0';

    while(1) {
        token = postfix[i];
        switch(token) {
            case '\0':
                printf("ans = %f\n", stack[top]);
                return;
            case '+': case '-': case '*': case '/':
                stack[top-1] =
                    cal(stack[top-1], token, stack[top]);
                top--;
                break;
            default:
                if(top < sizeof(stack) / sizeof(float)) {
                    temp[0] = postfix[i];
                    top++;
                    stack[top] = atof(temp);
                }
                break;
        }
        i++;
    }
}
```



Stack Example

- Example: stk_exp/postfix_cal.c

```
48 double cal(double p1, char op, double p2) {
49     switch(op) {
50         case '+':
51             return p1 + p2;
52         case '-':
53             return p1 - p2;
54         case '*':
55             return p1 * p2;
56         case '/':
57             return p1 / p2;
58         default:
59             printf("invalid operation");
60             exit(-1);
61     }
62 }
```

Infix - Postfix

- Use stack to convert infix to postfix
  - a + b * c ==> a b c * +
  - a * (b + c) / d + k ==> a b c + * d / k +
- This case only deals with + - * / ( )
  - no sign for numbers
- Main concept
  - Use stack to maintain operators

Stack Example: polish

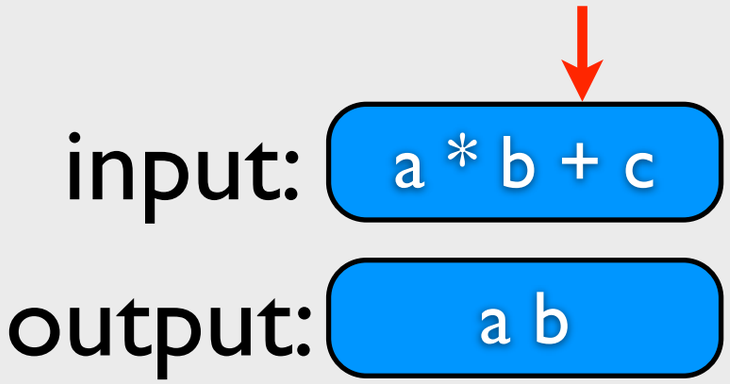
- Define the precedence

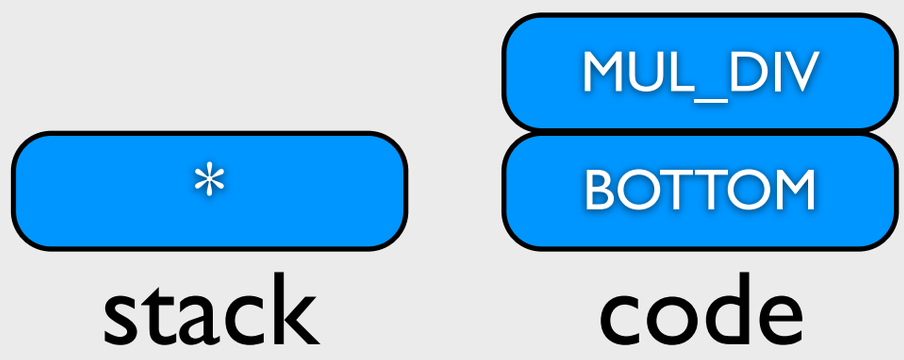
```
14 #define   BOTTOM      '\0'  /* precedence of stack bot. */
15 #define   EOL         '\1'  /* precedence of end-of-line*/
16 #define   LEFT_PAR    '\2'  /* precedence of (          */
17 #define   RIGHT_PAR   '\3'  /* precedence of )          */
18 #define   PLUS_MINUS  '\4'  /* precedence of + and -    */
19 #define   MUL_DIV     '\5'  /* precedence of * and /    */
```

- Some related functions
  - void initial (void): push BOTTOM on the code[ ] first
  - void char stack_top (void)
  - void push (char operator, char opr_code)
  - char pop (void)

Example: polish/polish.c

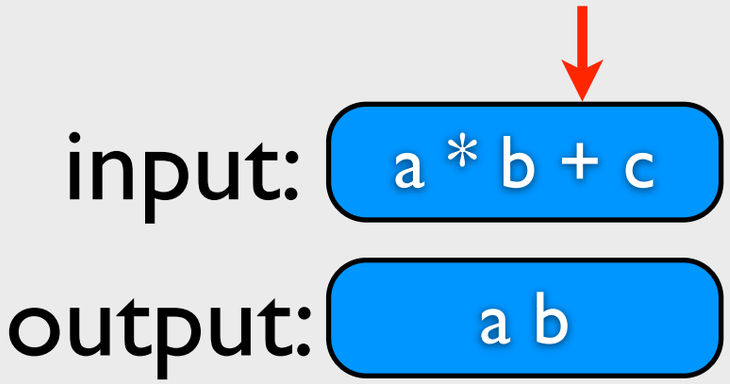
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();       /* remove ( and )   */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);      /* otherwise, push */
71          }
72      }
73  }
```

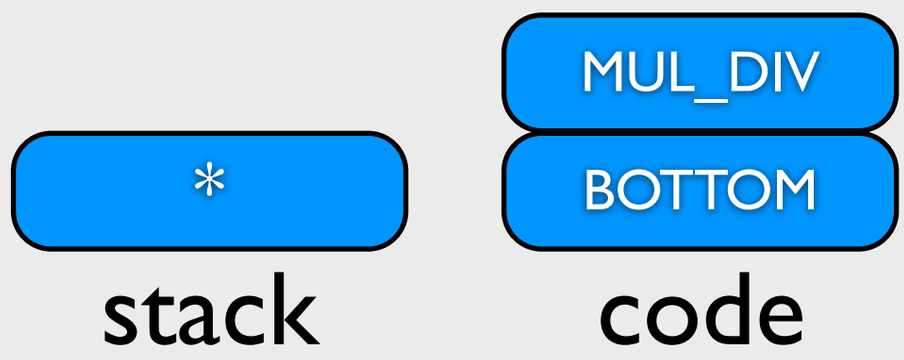




Example: polish/polish.c

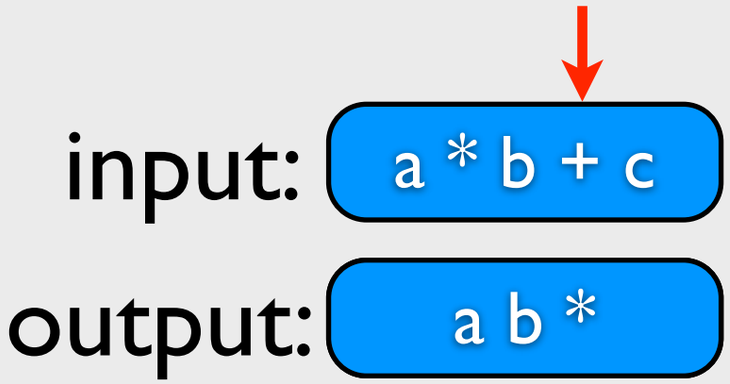
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();       /* remove ( and )   */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);      /* otherwise, push */
71          }
72      }
73  }
```

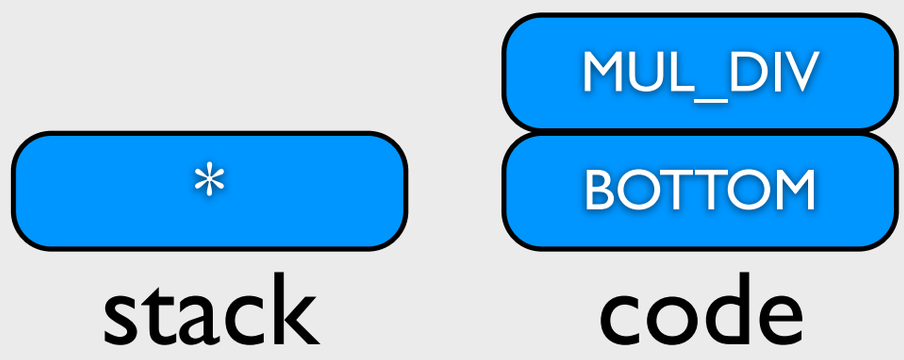




Example: polish/polish.c

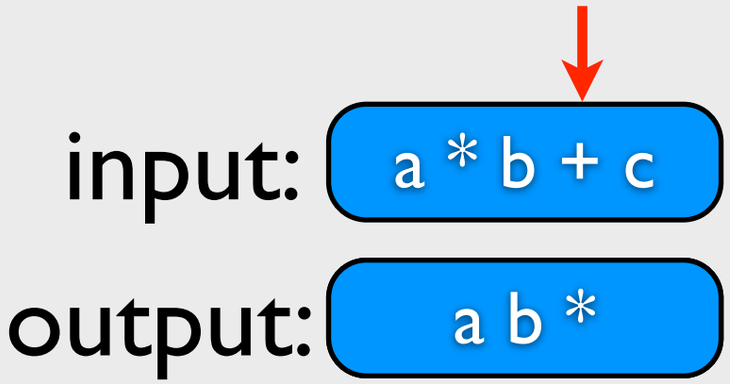
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();       /* remove ( and )   */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);      /* otherwise, push */
71          }
72      }
73  }
```

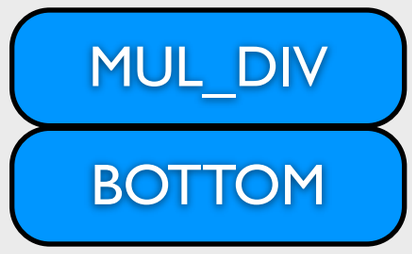




Example: polish/polish.c

```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();       /* remove ( and )   */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);      /* otherwise, push */
71          }
72      }
73  }
```

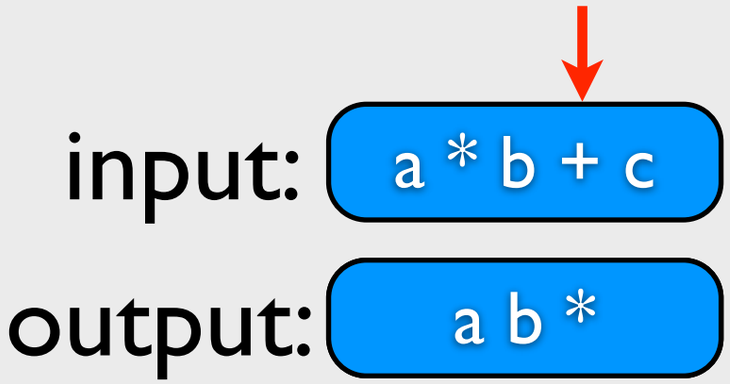


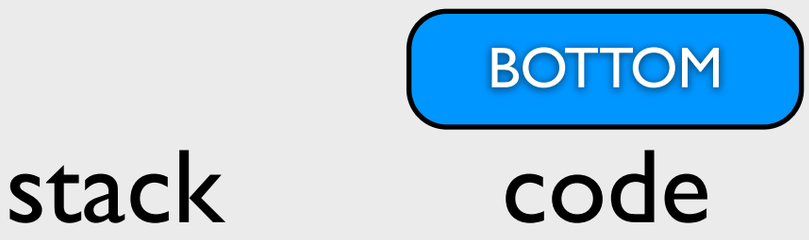


stack code

Example: polish/polish.c

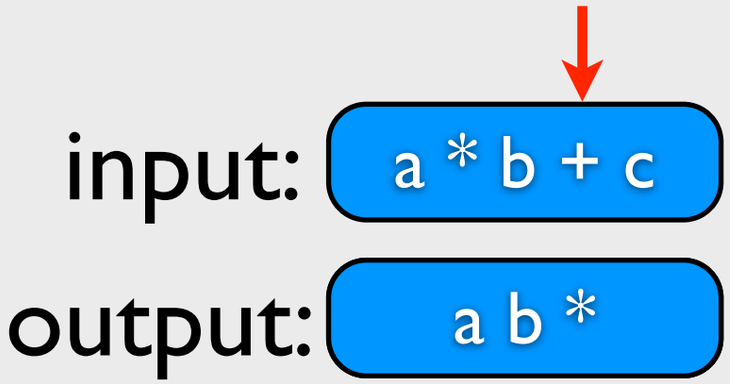
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();       /* remove ( and )   */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);      /* otherwise, push */
71          }
72      }
73  }
```

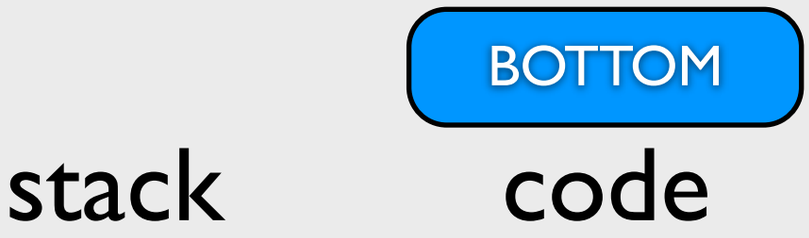




Example: polish/polish.c

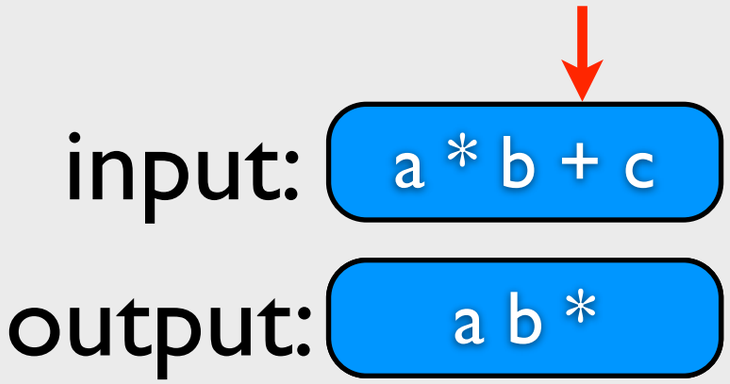
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();       /* remove ( and )   */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);      /* otherwise, push */
71          }
72      }
73  }
```

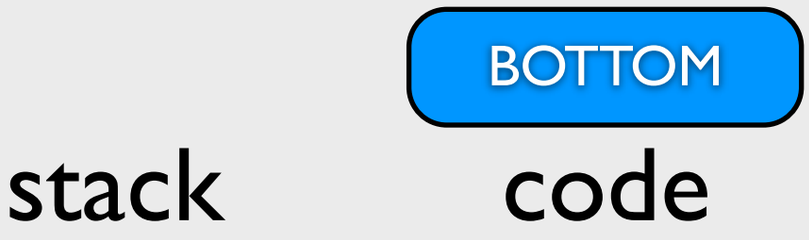




Example: polish/polish.c

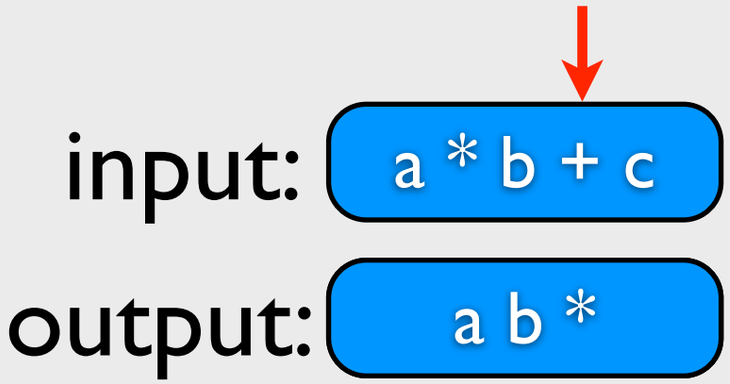
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43           if (isalpha(*p))   /* is it an operator        */
44                printf("%c ", *p);  /* YES, output it      */
45           else if (*p == '(') /* or, is it a '('         */
46                push(*p, LEFT_PAR); /* YES! push it.       */
47           else if (!isspace(*p)) { /* or, not a space?    */
48                switch(*p) {   /* it must be an operator   */
49                     case '+' :         /* handle + and -   */
50                     case '-' : opr = PLUS_MINUS;
51                                break;
52                     case '*' :         /* handle * and /   */
53                     case '/' : opr = MUL_DIV;
54                                break;
55                     case ')' : opr = RIGHT_PAR; /* ')'     */
56                                break;
57                     case '\0': opr = EOL;       /* EOL     */
58                                break;
59                     default  : printf("*** Unrecognizable char ***");
60                                exit(EXIT_FAILURE);
61                }
62                while ((t=stack_top()) >= opr) /* pop low  */
63                     printf("%c ", pop());     /* & print  */
64                if (t == LEFT_PAR && opr == RIGHT_PAR)
65                     (void) pop();      /* remove ( and )  */
66                else if (opr == EOL){    /* end of parsing  */
67                     printf("\n");
68                     exit(EXIT_SUCCESS);
69                } else
70                     push(*p, opr);     /* otherwise, push */
71           }
72      }
73 }
```

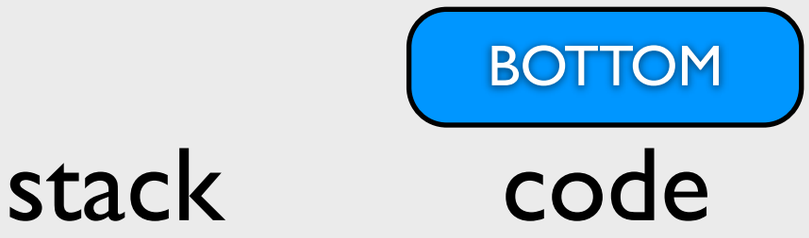




Example: polish/polish.c

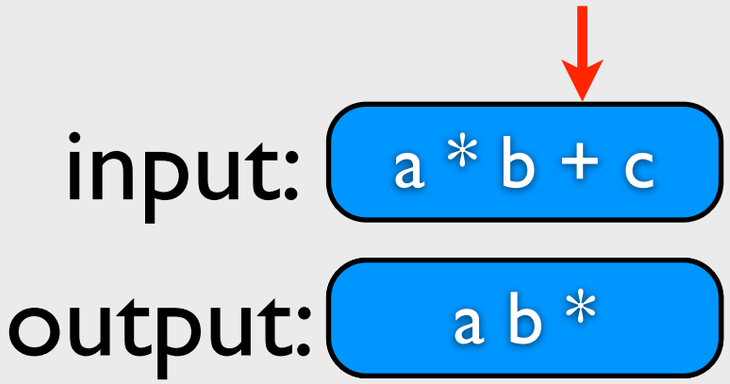
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();      /* remove ( and )    */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);     /* otherwise, push */
71          }
72      }
73  }
```

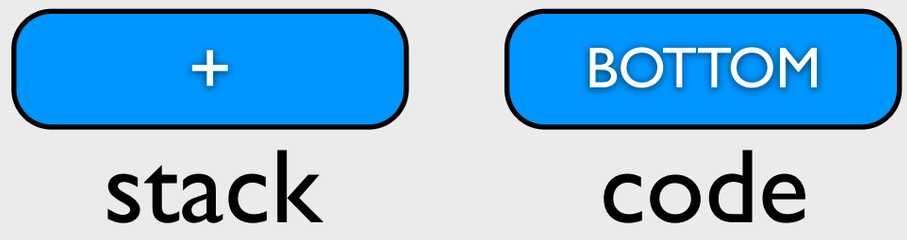




Example: polish/polish.c

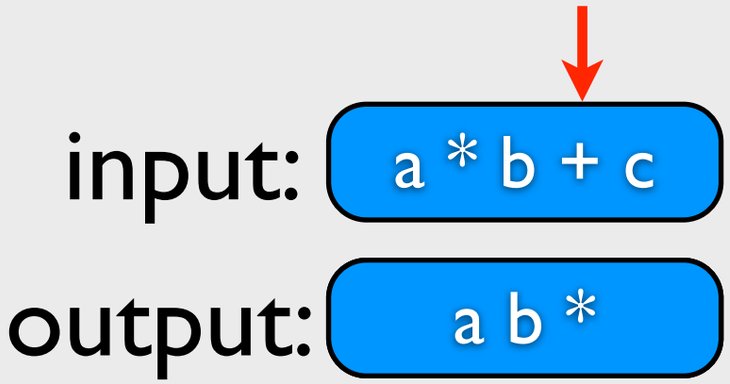
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43           if (isalpha(*p))   /* is it an operator        */
44                printf("%c ", *p);  /* YES, output it      */
45           else if (*p == '(') /* or, is it a '('         */
46                push(*p, LEFT_PAR); /* YES! push it.       */
47           else if (!isspace(*p)) { /* or, not a space?    */
48                switch(*p) {   /* it must be an operator   */
49                     case '+' :         /* handle + and -   */
50                     case '-' : opr = PLUS_MINUS;
51                                break;
52                     case '*' :         /* handle * and /   */
53                     case '/' : opr = MUL_DIV;
54                                break;
55                     case ')' : opr = RIGHT_PAR; /* ')'     */
56                                break;
57                     case '\0': opr = EOL;       /* EOL     */
58                                break;
59                     default  : printf("*** Unrecognizable char ***");
60                                exit(EXIT_FAILURE);
61                }
62                while ((t=stack_top()) >= opr) /* pop low  */
63                     printf("%c ", pop());     /* & print  */
64                if (t == LEFT_PAR && opr == RIGHT_PAR)
65                     (void) pop();      /* remove ( and )  */
66                else if (opr == EOL){    /* end of parsing  */
67                     printf("\n");
68                     exit(EXIT_SUCCESS);
69                } else
70                     push(*p, opr);     /* otherwise, push */
71           }
72      }
73 }
```

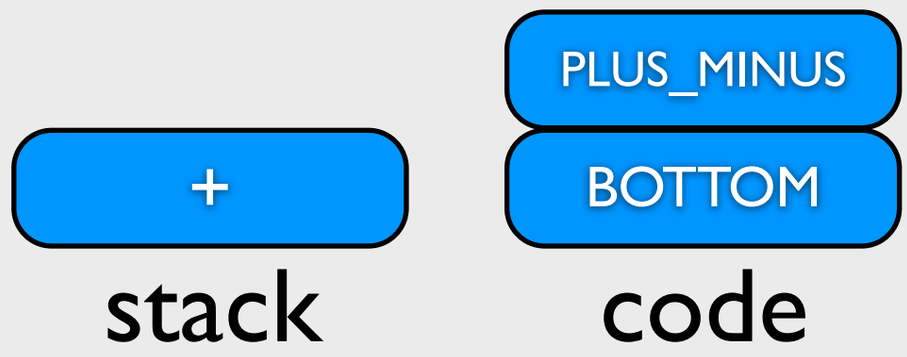




Example: polish/polish.c

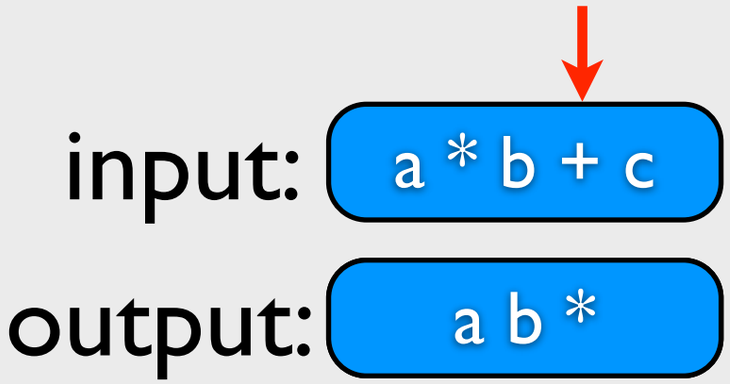
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();      /* remove ( and )    */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);     /* otherwise, push */
71          }
72      }
73  }
```

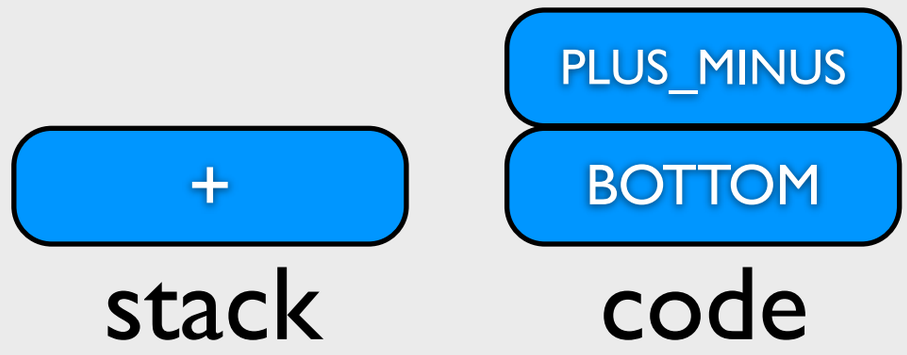




Example: polish/polish.c

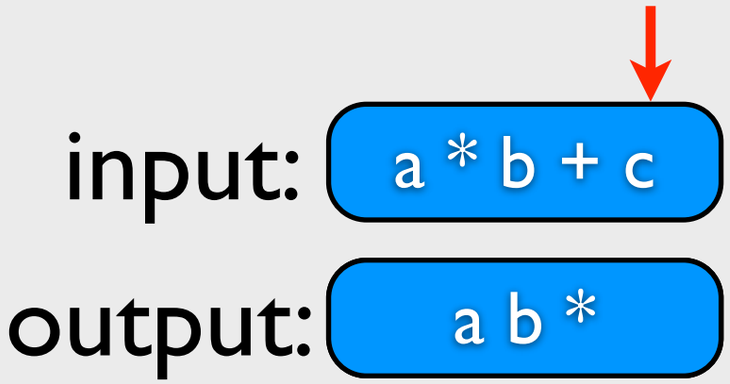
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();      /* remove ( and )    */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);     /* otherwise, push */
71          }
72      }
73  }
```

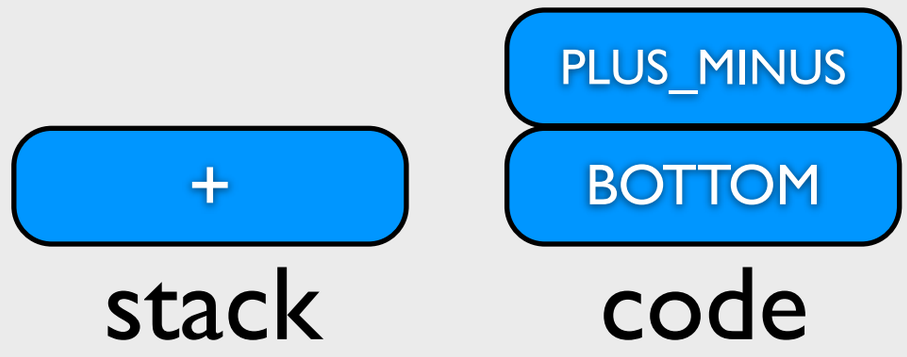




Example: polish/polish.c

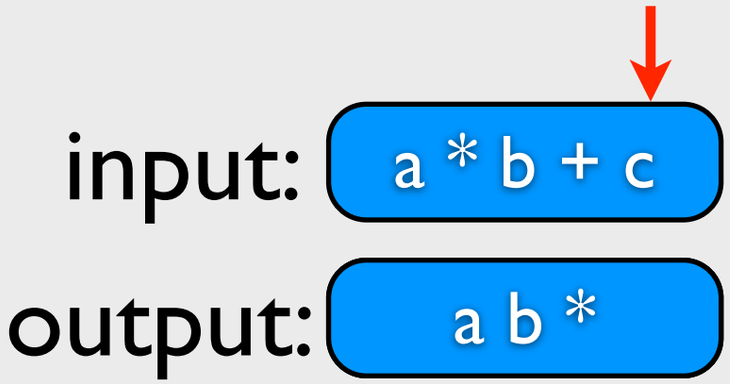
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();      /* remove ( and )    */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);     /* otherwise, push */
71          }
72      }
73  }
```

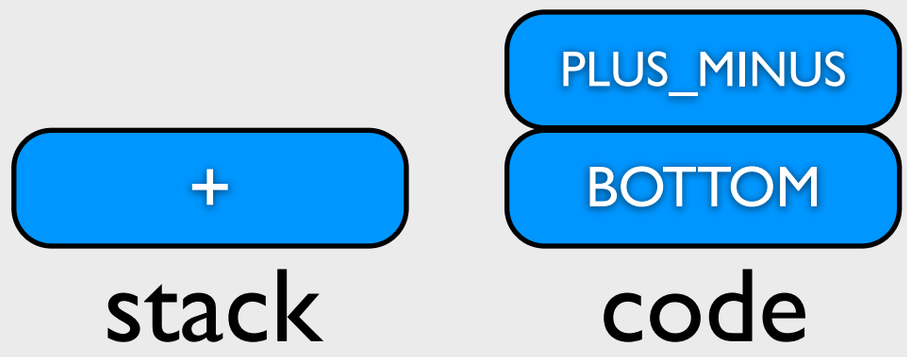




Example: polish/polish.c

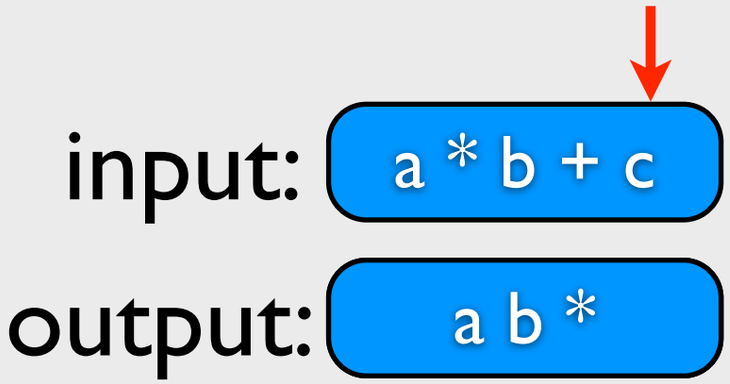
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();      /* remove ( and )    */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);     /* otherwise, push */
71          }
72      }
73  }
```

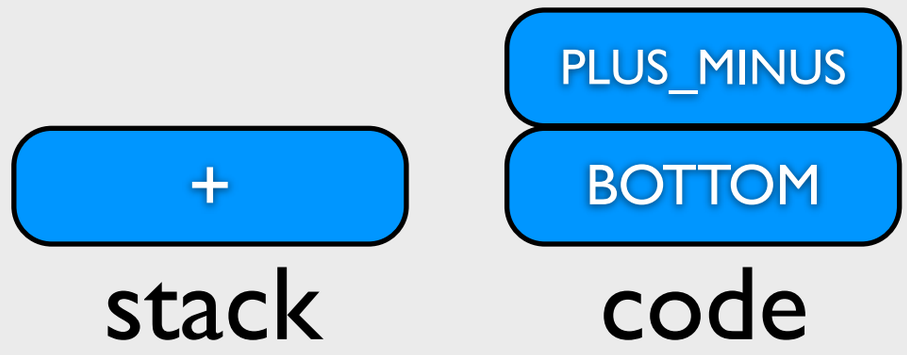




Example: polish/polish.c

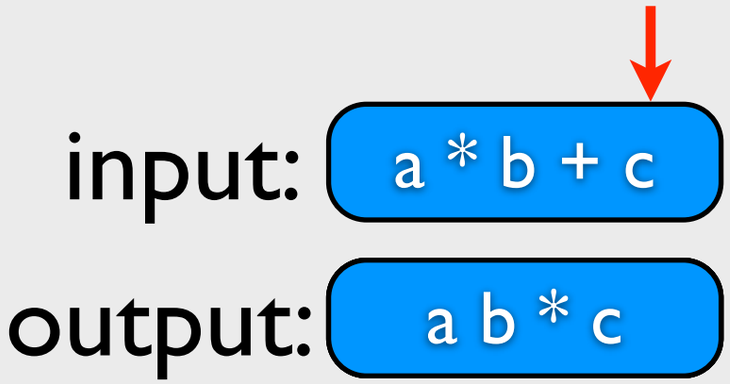
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();      /* remove ( and )    */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);     /* otherwise, push */
71          }
72      }
73  }
```

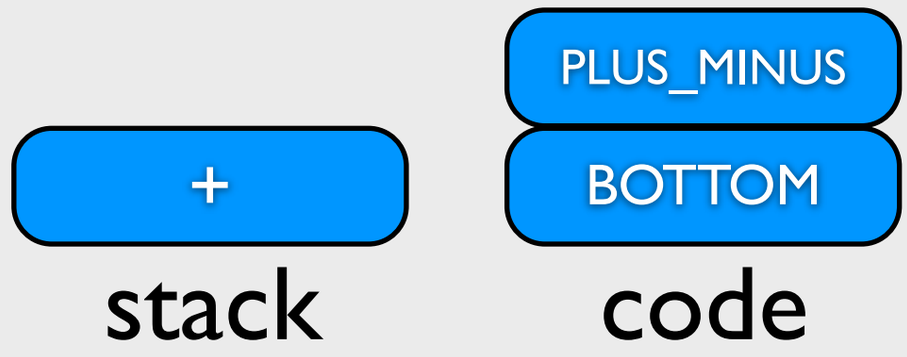




Example: polish/polish.c

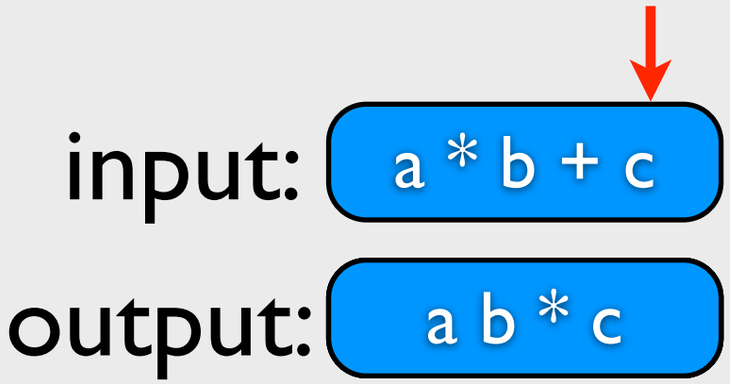
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {    /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();       /* remove ( and )   */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);      /* otherwise, push */
71          }
72      }
73  }
```

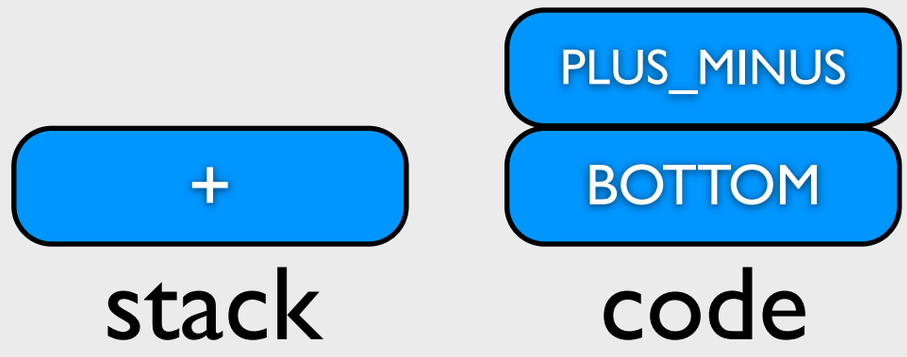




Example: polish/polish.c

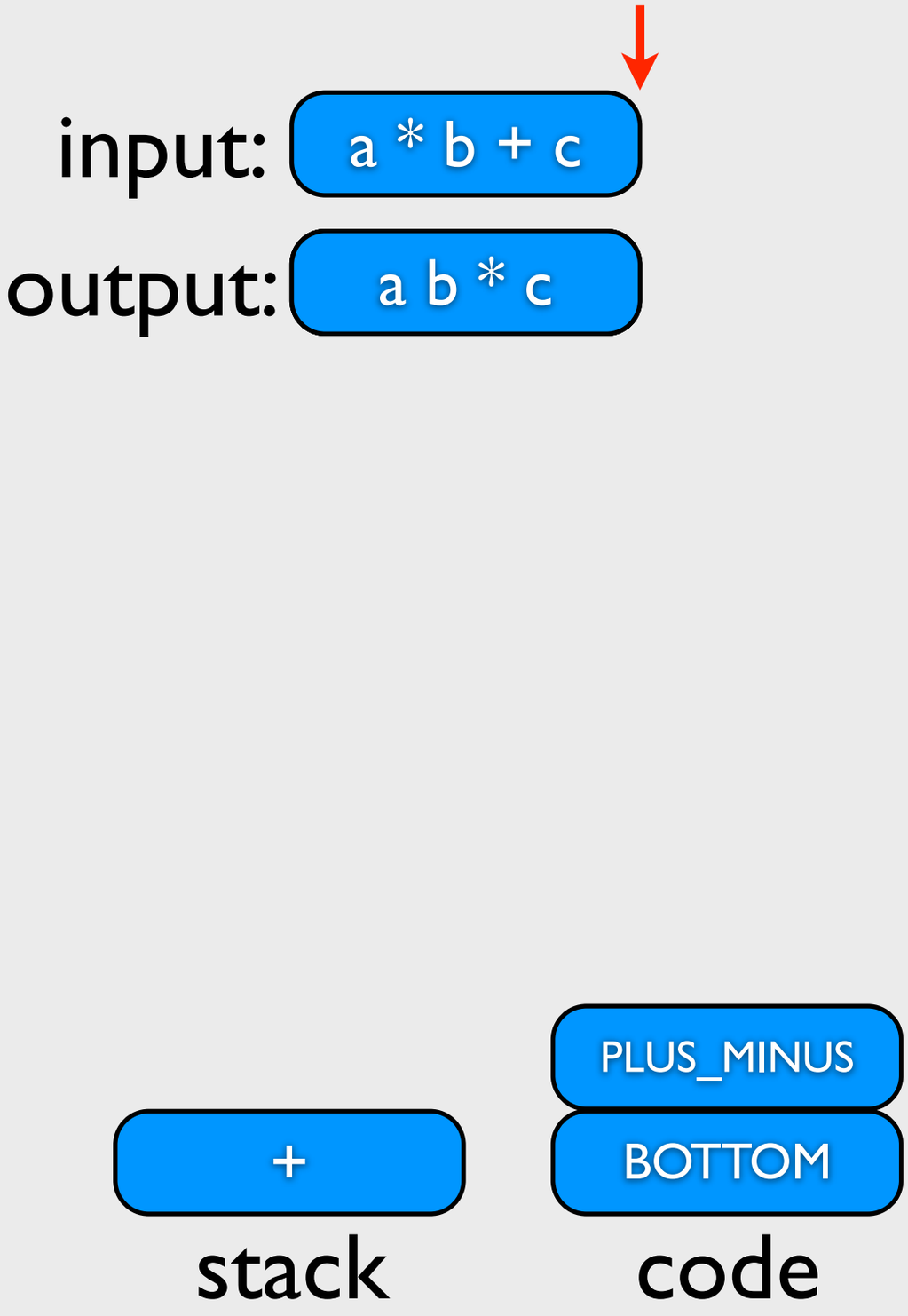
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();      /* remove ( and )    */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);     /* otherwise, push */
71          }
72      }
73  }
```





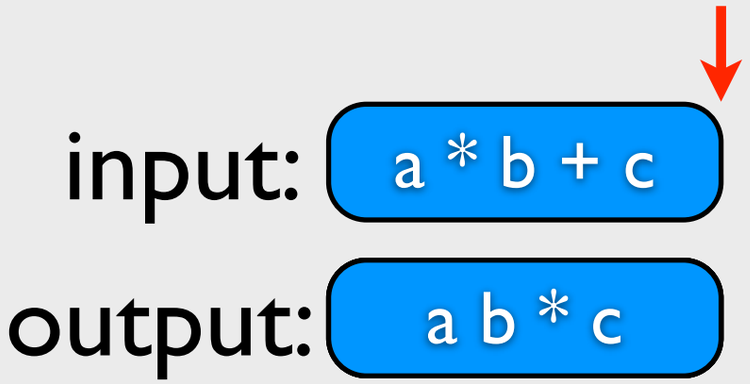
Example: polish/polish.c

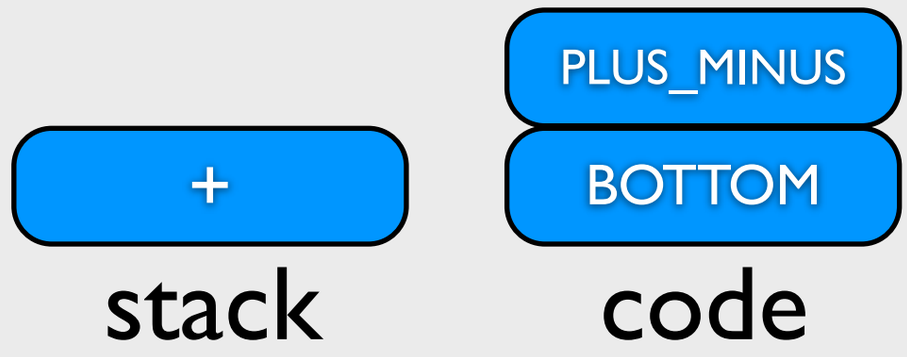
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {    /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();       /* remove ( and )   */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);      /* otherwise, push */
71          }
72      }
73  }
```



Example: polish/polish.c

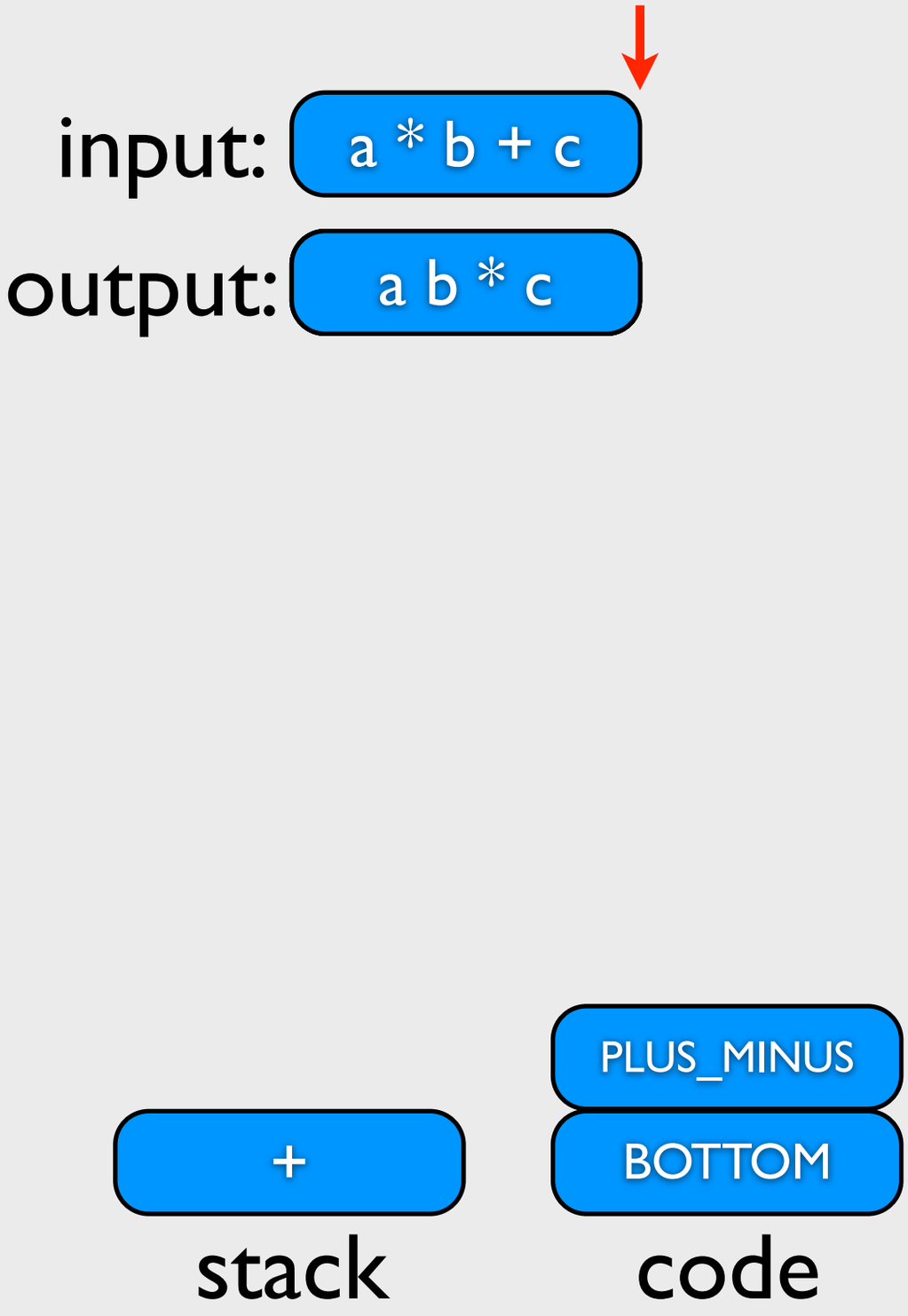
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator    */
49                  case '+' :          /* handle + and -    */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /    */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'       */
56                             break;
57                  case '\0': opr = EOL;       /* EOL       */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print    */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();      /* remove ( and )    */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);     /* otherwise, push */
71          }
72      }
73  }
```





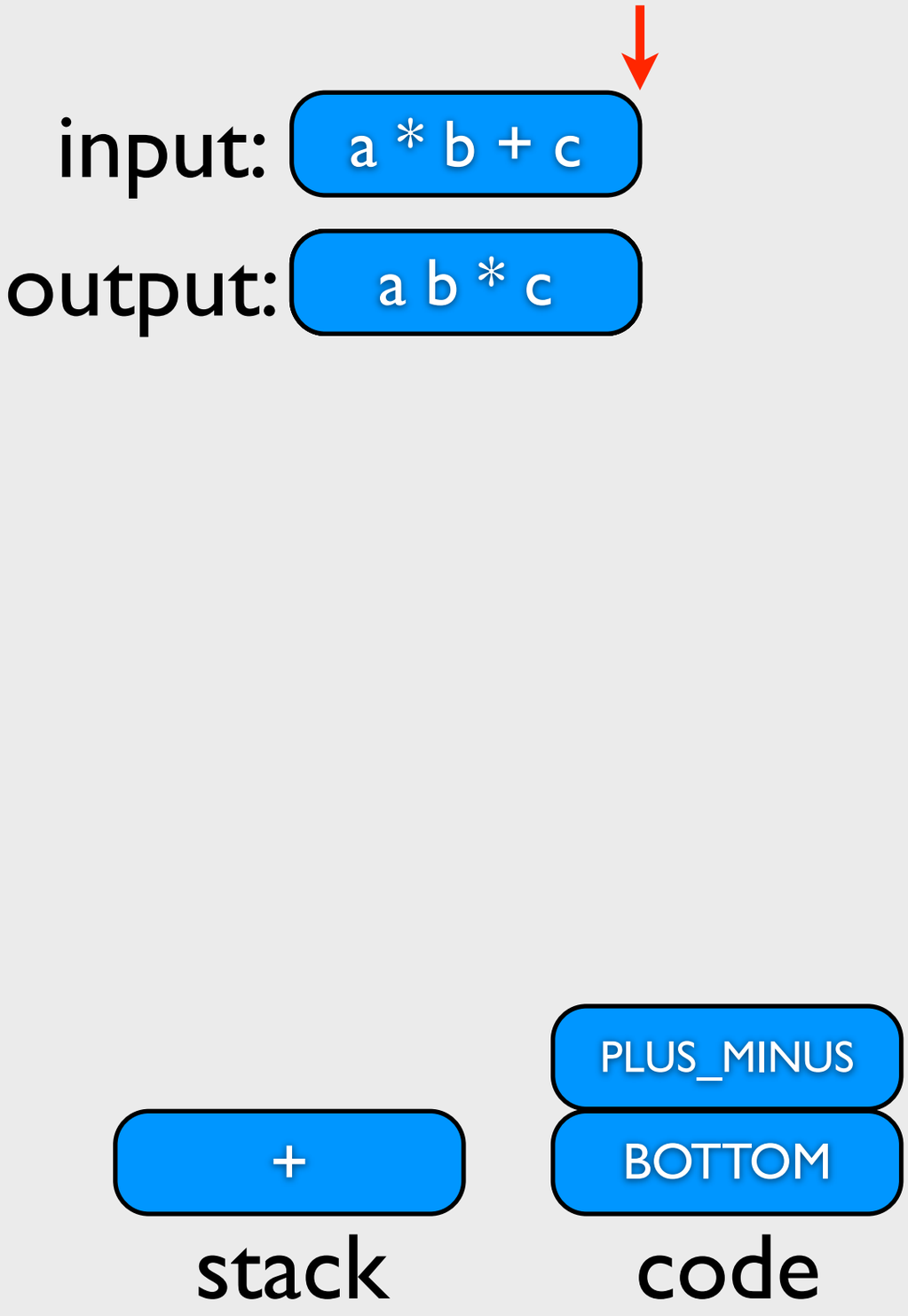
Example: polish/polish.c

```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();       /* remove ( and )   */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);      /* otherwise, push */
71          }
72      }
73  }
```



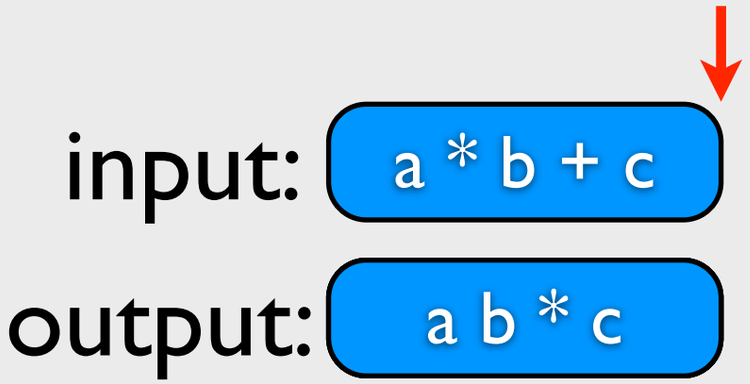
Example: polish/polish.c

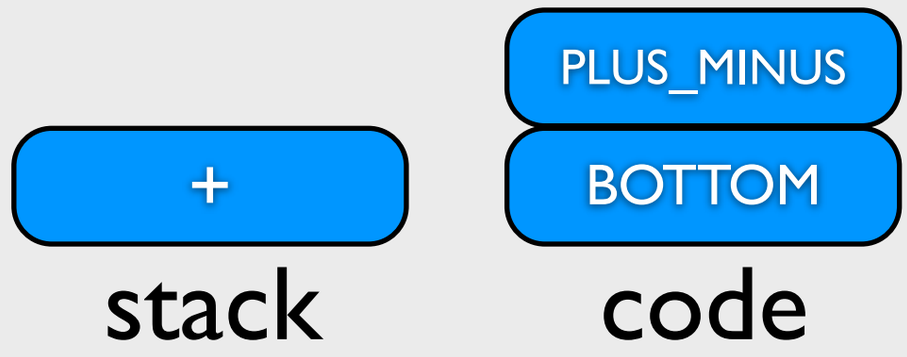
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();       /* remove ( and )   */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);      /* otherwise, push */
71          }
72      }
73  }
```



Example: polish/polish.c

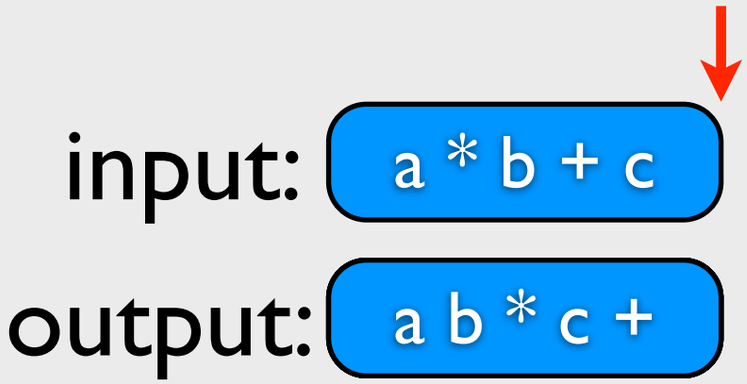
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();       /* remove ( and )   */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);      /* otherwise, push */
71          }
72      }
73  }
```

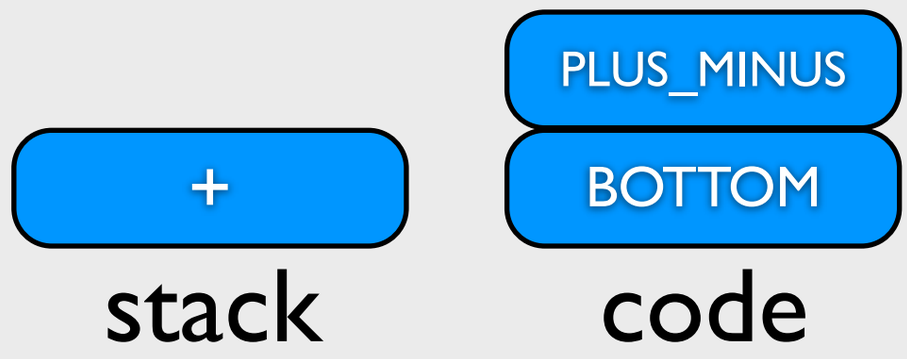




Example: polish/polish.c

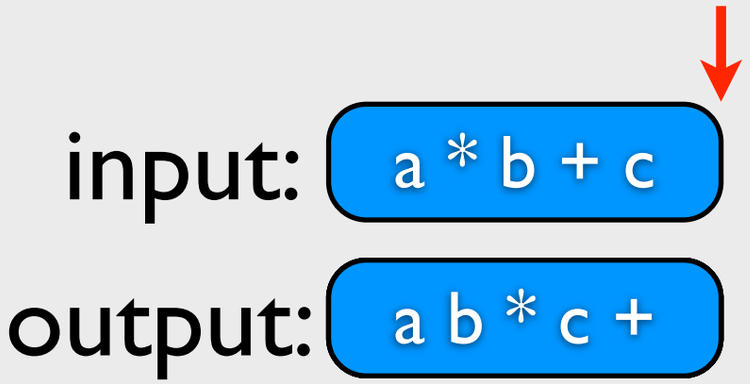
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();       /* remove ( and )   */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);      /* otherwise, push */
71          }
72      }
73  }
```





Example: polish/polish.c

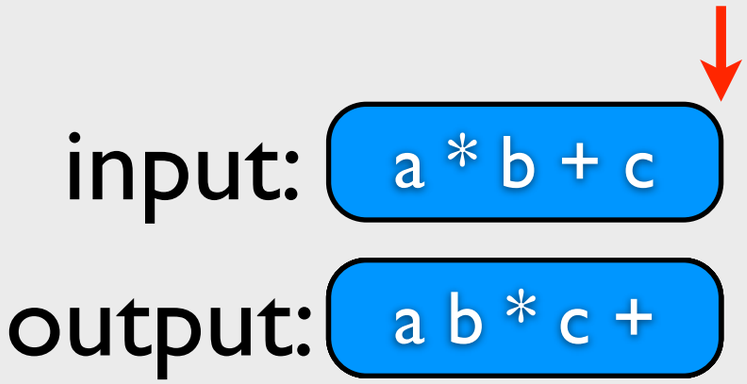
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();       /* remove ( and )   */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);      /* otherwise, push */
71          }
72      }
73  }
```

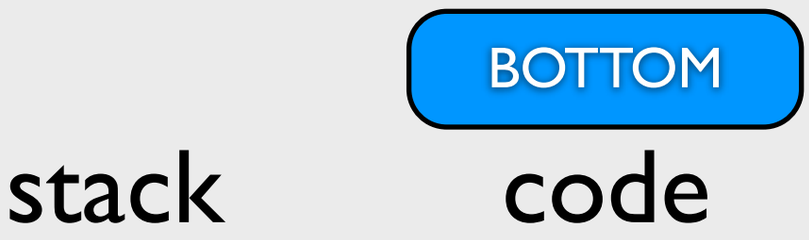


stack code

Example: polish/polish.c

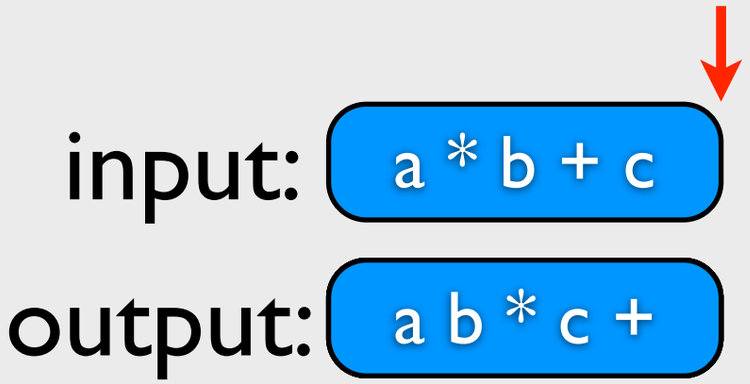
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();       /* remove ( and )   */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);      /* otherwise, push */
71          }
72      }
73  }
```





Example: polish/polish.c

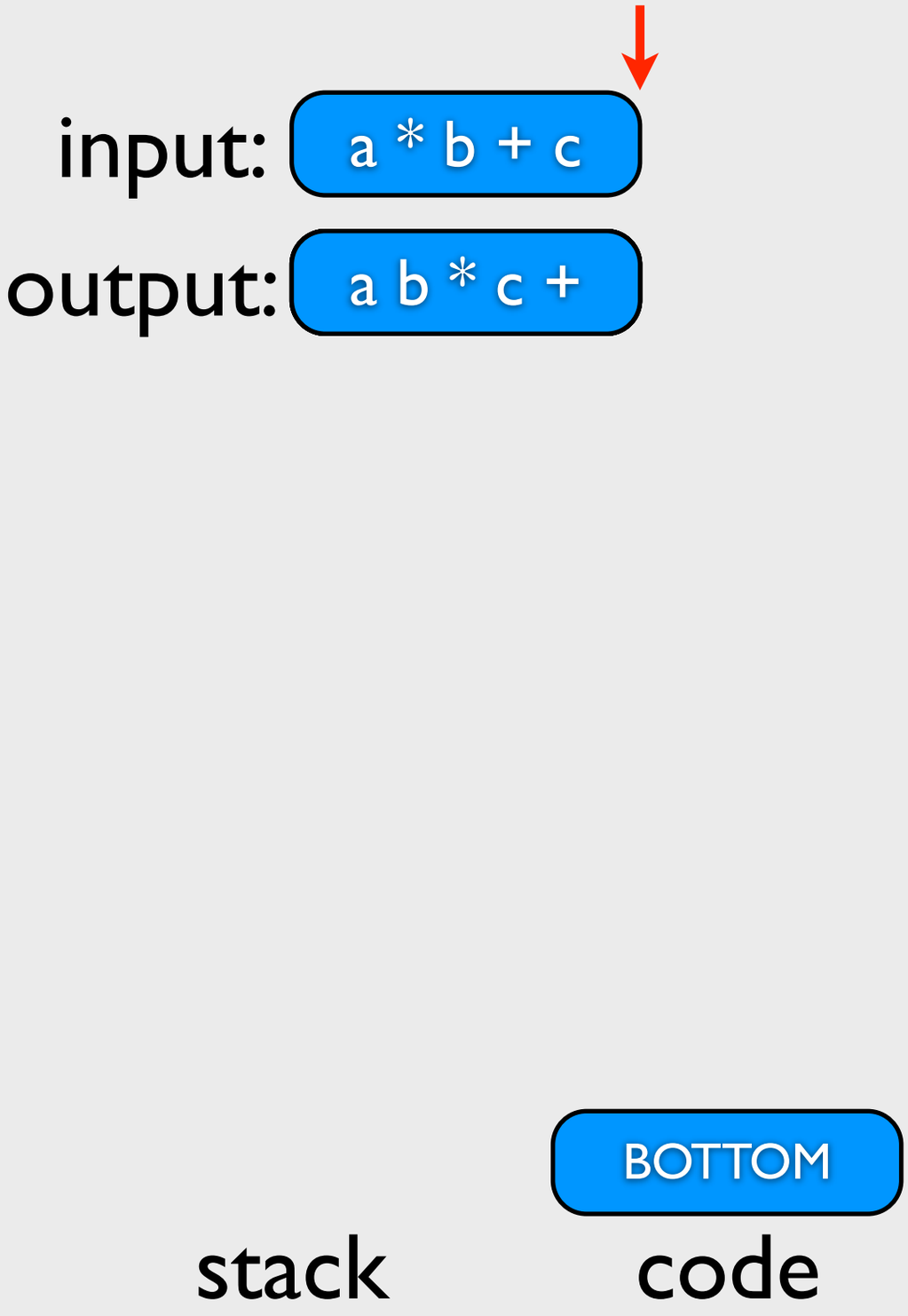
```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();      /* remove ( and )    */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);     /* otherwise, push */
71          }
72      }
73  }
```



stack code

Example: polish/polish.c

```
41      initial();              /* initialize the two stacks*/
42      for (; ; p++) {         /* loop for each char.      */
43          if (isalpha(*p))    /* is it an operator        */
44              printf("%c ", *p);  /* YES, output it       */
45          else if (*p == '(') /* or, is it a '('          */
46              push(*p, LEFT_PAR); /* YES! push it.        */
47          else if (!isspace(*p)) { /* or, not a space?    */
48              switch(*p) {   /* it must be an operator   */
49                  case '+' :          /* handle + and -   */
50                  case '-' : opr = PLUS_MINUS;
51                             break;
52                  case '*' :          /* handle * and /   */
53                  case '/' : opr = MUL_DIV;
54                             break;
55                  case ')' : opr = RIGHT_PAR; /* ')'      */
56                             break;
57                  case '\0': opr = EOL;       /* EOL      */
58                             break;
59                  default  : printf("*** Unrecognizable char ***");
60                             exit(EXIT_FAILURE);
61              }
62              while ((t=stack_top()) >= opr) /* pop low   */
63                  printf("%c ", pop());     /* & print   */
64              if (t == LEFT_PAR && opr == RIGHT_PAR)
65                  (void) pop();      /* remove ( and )    */
66              else if (opr == EOL){    /* end of parsing  */
67                  printf("\n");
68                  exit(EXIT_SUCCESS);
69              } else
70                  push(*p, opr);     /* otherwise, push */
71          }
72      }
73  }
```



Infix - Postfix

- Another case: a * (b + c) / d + k

Infix - Postfix

- Another case: a * (b + c) / d + k

output: a b c **input: / d + k**
stack: * (+ **next operation:)**

Infix - Postfix

- Another case: a * (b + c) / d + k

output: a b c input: / d + k
stack: * (+ next operation:)

output: a b c + input: / d + k
stack: * (next operation:)

Infix - Postfix

- Another case: a * (b + c) / d + k

output: a b c input: / d + k
stack: * (+ next operation:)

output: a b c + input: / d + k
stack: * (next operation:)

dismiss ( )

Infix - Postfix

- Another case: a * (b + c) / d + k

output: a b c **input: / d + k**
stack: * (+ **next operation:)**

output: a b c + **input: / d + k**
stack: * (**next operation:)**

dismiss ( )

output: a b c + **input: d + k**
stack: * **next operation: /**

Infix - Postfix

- Another case: a * (b + c) / d + k

output: a b c **input: / d + k**
stack: * (+ **next operation:)**

output: a b c + **input: / d + k**
stack: * (**next operation:)**

dismiss ( )

output: a b c + **input: d + k**
stack: * **next operation: /**

output: a b c + * d **input: k**
stack: / **next operation: +**

Infix - Postfix

- Another case: a * (b + c) / d + k

output: a b c
stack: * (+

input: / d + k
next operation:)

output: a b c +
stack: * (

input: / d + k
next operation:)

dismiss ( )

output: a b c +
stack: *

input: d + k
next operation: /

output: a b c + * d
stack: /

input: k
next operation: +

output: a b c + * d / k
stack: +

input:
next operation: EOL

Infix - Postfix

- Another case: a * (b + c) / d + k

output: a b c
stack: * (+

input: / d + k
next operation:)

output: a b c +
stack: * (

input: / d + k
next operation:)

dismiss ( )

output: a b c +
stack: *

input: d + k
next operation: /

output: a b c + * d
stack: /

input: k
next operation: +

output: a b c + * d / k
stack: +

input:
next operation: EOL

output: a b c + * d / k +
stack:

input:
next operation:

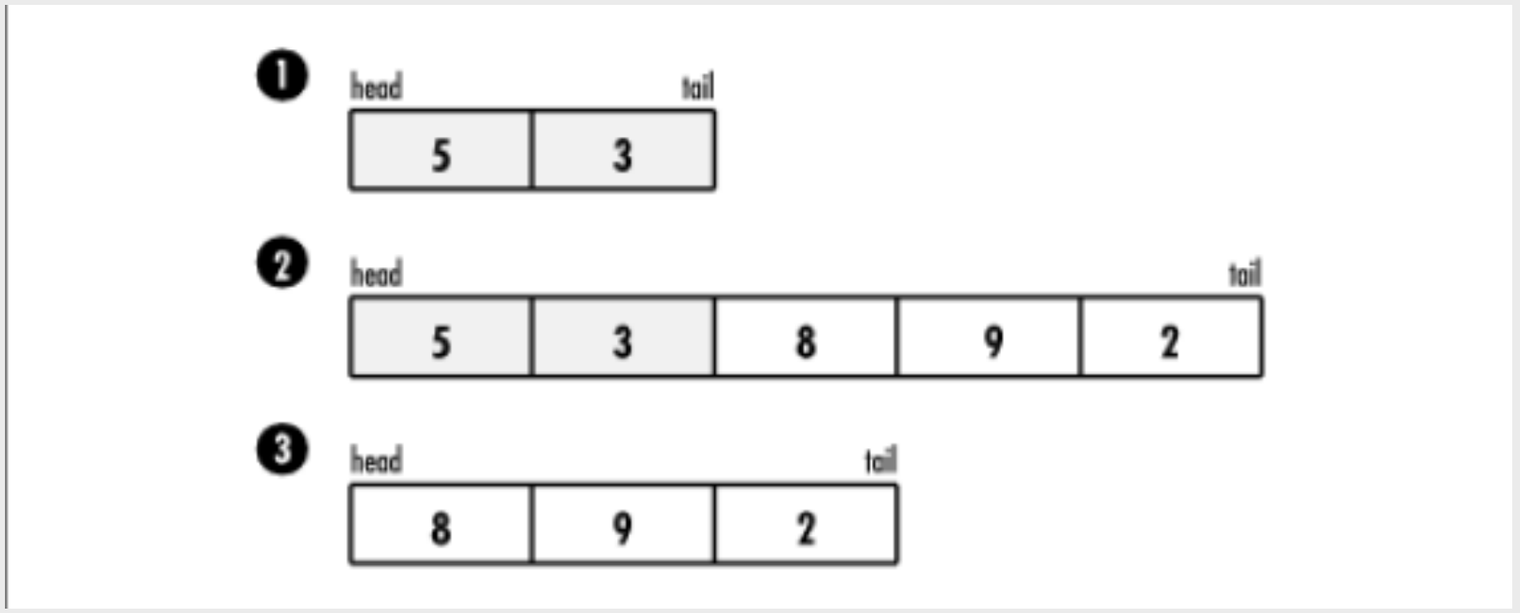
Advanced Queue

Queues

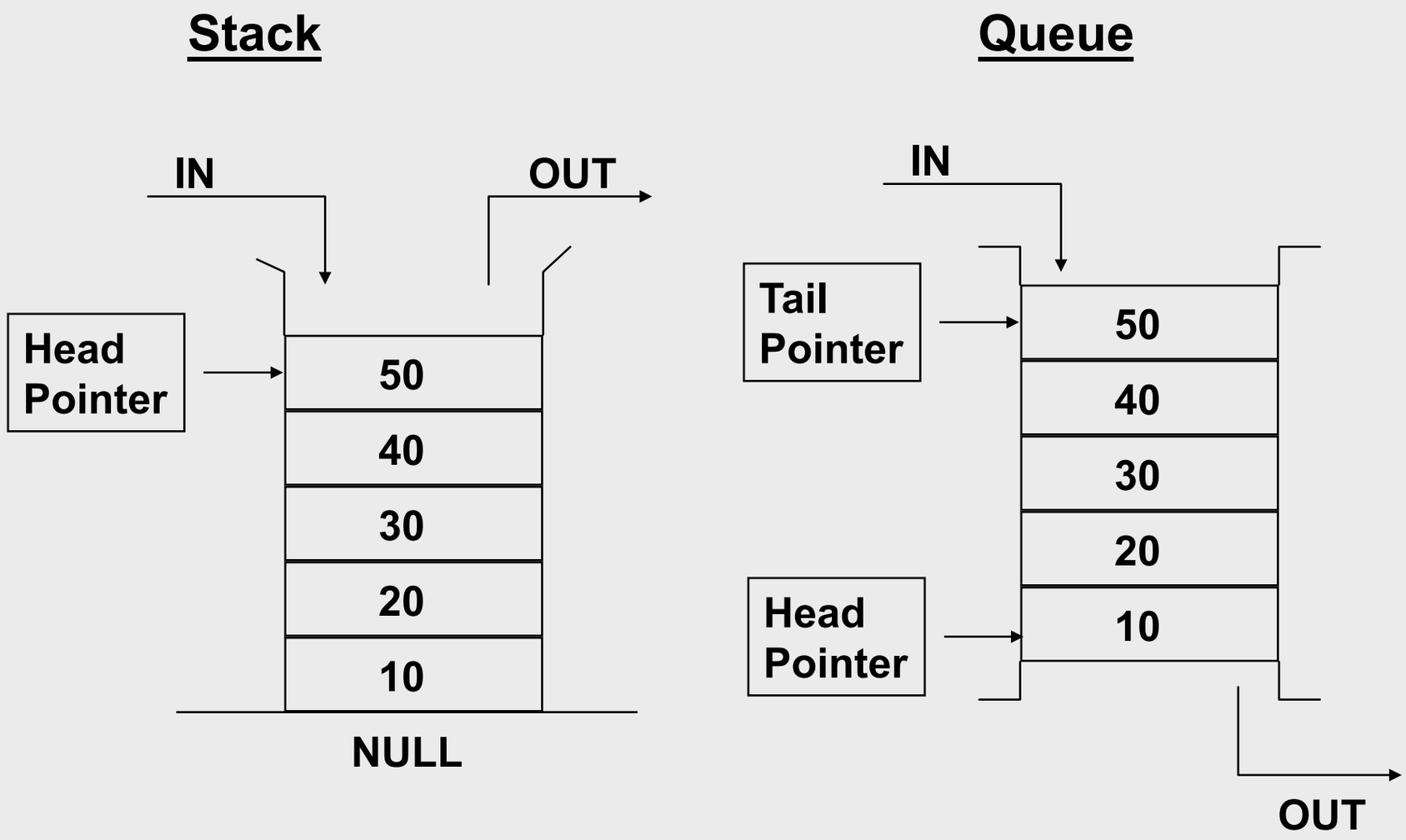
- Queue
  - Similar to a supermarket checkout line
  - First-in, first-out (FIFO)
    - Nodes are removed only from the head
    - Nodes are inserted only at the tail
- Insert and remove operations
  - Enqueue (insert) and dequeue (remove)

Queues

- 1. A queue with some elements already enqueued
- 2. After enqueuing 8, 9, and 2
- 3. After dequeuing 5 and 3



Stack vs. Queue



Implementation of Queue

- Example: queue/queue.h

```
5  #ifndef QUEUE_H
6  #define QUEUE_H
7
8  #include <stdlib.h>
9  #include "list.h"
10
11 /************************************************************************
12  *  Implement queues as linked lists.                                   *
13  ************************************************************************/
14 typedef List Queue;
15
16 /************************************************************************
17  *  ------------------------- Public Interface -------------------------  *
18  ************************************************************************/
19 #define queue_init list_init
20 #define queue_destroy list_destroy
21 int queue_enqueue(Queue *queue, const void *data);
22 int queue_dequeue(Queue *queue, void **data);
23 #define queue_peek(queue) ((queue)->head == NULL ? NULL : (queue)->head->data)
24 #define queue_size list_size
25 #endif
```

Implementation of Queue

- Example: queue/queue.h

```
5  #ifndef QUEUE_H
6  #define QUEUE_H
7
8  #include <stdlib.h>
9  #include "list.h"
10
11 /************************************************************************
12  *  Implement queues as linked lists.                                   *
13  ************************************************************************/
14 typedef List Queue;
15
16 /************************************************************************
17  *  ------------------------- Public Interface -------------------------  *
18  ************************************************************************/
19 #define queue_init list_init
20 #define queue_destroy list_destroy
21 int queue_enqueue(Queue *queue, const void *data);
22 int queue_dequeue(Queue *queue, void **data);
23 #define queue_peek(queue) ((queue)->head == NULL ? NULL : (queue)->head->data)
24 #define queue_size list_size
25 #endif
```

Implementation of Queue

- Example: queue/queue.h

```
5  #ifndef QUEUE_H
6  #define QUEUE_H
7
8  #include <stdlib.h>
9  #include "list.h"
10
11 /**************************************************************************
12  *  Implement queues as linked lists.                                     *
13  **************************************************************************/
14 typedef List Queue;
15
16 /**************************************************************************
17  *  ------------------------- Public Interface -------------------------  *
18  **************************************************************************/
19 #define queue_init list_init
20 #define queue_destroy list_destroy
21 int queue_enqueue(Queue *queue, const void *data);
22 int queue_dequeue(Queue *queue, void **data);
23 #define queue_peek(queue) ((queue)->head == NULL ? NULL : (queue)->head->data)
24 #define queue_size list_size
25 #endif
```

Implementation of Queue

- Example: queue/queue.h

```
5  #ifndef QUEUE_H
6  #define QUEUE_H
7
8  #include <stdlib.h>
9  #include "list.h"
10
11 /**************************************************************************
12  *  Implement queues as linked lists.                                     *
13  **************************************************************************/
14 typedef List Queue;
15
16 /**************************************************************************
17  *  ------------------------- Public Interface -------------------------  *
18  **************************************************************************/
19 #define queue_init list_init
20 #define queue_destroy list_destroy
21 int queue_enqueue(Queue *queue, const void *data);
22 int queue_dequeue(Queue *queue, void **data);
23 #define queue_peek(queue) ((queue)->head == NULL ? NULL : (queue)->head->data)
24 #define queue_size list_size
25 #endif
```

Implementation of Queue

- Example: queue/queue.h

```
 5 #ifndef QUEUE_H
 6 #define QUEUE_H
 7
 8 #include <stdlib.h>
 9 #include "list.h"
10
11 /**************************************************************************
12  *  Implement queues as linked lists.                                     *
13  **************************************************************************/
14 typedef List Queue;
15
16 /**************************************************************************
17  *  ------------------------- Public Interface -------------------------  *
18  **************************************************************************/
19 #define queue_init list_init
20 #define queue_destroy list_destroy
21 int queue_enqueue(Queue *queue, const void *data);
22 int queue_dequeue(Queue *queue, void **data);
23 #define queue_peek(queue) ((queue)->head == NULL ? NULL : (queue)->head->data)
24 #define queue_size list_size
25 #endif
```

Implementation of Queue

- Example: queue/queue.h

```
5  #ifndef QUEUE_H
6  #define QUEUE_H
7
8  #include <stdlib.h>
9  #include "list.h"
10
11 /************************************************************************
12  *  Implement queues as linked lists.                                   *
13  ************************************************************************/
14 typedef List Queue;
15
16 /************************************************************************
17  *  ------------------------- Public Interface -------------------------  *
18  ************************************************************************/
19 #define queue_init list_init
20 #define queue_destroy list_destroy
21 int queue_enqueue(Queue *queue, const void *data);
22 int queue_dequeue(Queue *queue, void **data);
23 #define queue_peek(queue) ((queue)->head == NULL ? NULL : (queue)->head->data)
24 #define queue_size list_size
25 #endif
```

Implementation of Queue

- Example: queue/queue.c

```c
#include <stdlib.h>
#include "list.h"
#include "queue.h"

/*****************************************************************************
 *   ------------------------------ queue_enqueue ----------------------------
 *****************************************************************************/
int queue_enqueue(Queue *queue, const void *data) {
    return list_ins_next(queue, list_tail(queue), data);
}

/*****************************************************************************
 *   ------------------------------ queue_dequeue ----------------------------
 *****************************************************************************/
int queue_dequeue(Queue *queue, void **data) {
    return list_rem_next(queue, NULL, data);
}
```

Implementation of Queue

- Example: queue/queue.c

```
5  #include <stdlib.h>
6  #include "list.h"
7  #include "queue.h"
8
9  /*****************************************************************************
10  *  ---------------------------- queue_enqueue ----------------------
11  *****************************************************************************/
12 int queue_enqueue(Queue *queue, const void *data) {
13     return list_ins_next(queue, list_tail(queue), data);
14 }
15
16 /*****************************************************************************
17  *  ---------------------------- queue_dequeue ----------------------
18  *****************************************************************************/
19 int queue_dequeue(Queue *queue, void **data) {
20     return list_rem_next(queue, NULL, data);
21 }
```

Implementation of Queue

- Example: queue/test.c

Implementation of Queue

- Example: queue/test.c

```
44      queue_init(&queue, free);
45
46      /************************************************************
47       *  Perform some queue operations.
48       ************************************************************/
49      fprintf(stdout, "Enqueuing 10 elements\n");
50      for (i = 0; i < 10; i++) {
51          if ((data = (int *)malloc(sizeof(int))) == NULL)
52              return 1;
53          *data = i + 1;
54          if (queue_enqueue(&queue, data) != 0)
55              return 1;
56      }
57      print_queue(&queue);
```

Implementation of Queue

- Example: queue/test.c

```
44      queue_init(&queue, free);
45
46      /************************************************************
47       *  Perform some queue operations.
48       ************************************************************/
49      fprintf(stdout, "Enqueuing 10 elements\n");
50      for (i = 0; i < 10; i++) {
51          if ((data = (int *)malloc(sizeof(int))) == NULL)
52              return 1;
53          *data = i + 1;
54          if (queue_enqueue(&queue, data) != 0)
55              return 1;
56      }
57      print_queue(&queue);
```

```
Enqueuing 10 elements
Queue size is 10
queue[000]=001
queue[001]=002
queue[002]=003
queue[003]=004
queue[004]=005
queue[005]=006
queue[006]=007
queue[007]=008
queue[008]=009
queue[009]=010
```

Implementation of Queue

- Example: queue/test.c

Implementation of Queue

- Example: queue/test.c

```
59      fprintf(stdout, "Dequeuing 5 elements\n");
60      for (i = 0; i < 5; i++) {
61          if (queue_dequeue(&queue, (void **)&data) == 0)
62              free(data);
63          else
64              return 1;
65      }
66      print_queue(&queue);
```

Implementation of Queue

- Example: queue/test.c

```
59      fprintf(stdout, "Dequeuing 5 elements\n");
60      for (i = 0; i < 5; i++) {
61          if (queue_dequeue(&queue, (void **)&data) == 0)
62              free(data);
63          else
64              return 1;
65      }
66      print_queue(&queue);
```

```
Dequeuing 5 elements
Queue size is 5
queue[000]=006
queue[001]=007
queue[002]=008
queue[003]=009
queue[004]=010
```

Implementation of Queue

- Example: queue/test.c

Implementation of Queue

- Example: queue/test.c

```
68      fprintf(stdout, "Enqueuing 100 and 200\n");
69      if ((data = (int *)malloc(sizeof(int))) == NULL)
70          return 1;
71      *data = 100;
72      if (queue_enqueue(&queue, data) != 0)
73          return 1;
74
75      if ((data = (int *)malloc(sizeof(int))) == NULL)
76          return 1;
77      *data = 200;
78      if (queue_enqueue(&queue, data) != 0)
79          return 1;
80      print_queue(&queue);
```

Implementation of Queue

- Example: queue/test.c

```
68      fprintf(stdout, "Enqueuing 100 and 200\n");
69      if ((data = (int *)malloc(sizeof(int))) == NULL)
70          return 1;
71      *data = 100;
72      if (queue_enqueue(&queue, data) != 0)
73          return 1;
74
75      if ((data = (int *)malloc(sizeof(int))) == NULL)
76          return 1;
77      *data = 200;
78      if (queue_enqueue(&queue, data) != 0)
79          return 1;
80      print_queue(&queue);
```

```
Enqueuing 100 and 200
Queue size is 7
queue[000]=006
queue[001]=007
queue[002]=008
queue[003]=009
queue[004]=010
queue[005]=100
queue[006]=200
```

Implementation of Queue

- Example: queue/test.c

Implementation of Queue

- Example: queue/test.c

```
82      if ((data = queue_peek(&queue)) != NULL)
83          fprintf(stdout, "Peeking at the head element...Value=%03d\n", *data);
84      else
85          fprintf(stdout, "Peeking at the head element...Value=NULL\n");
86
87      print_queue(&queue);
88      fprintf(stdout, "Dequeuing all elements\n");
89      while (queue_size(&queue) > 0) {
90          if (queue_dequeue(&queue, (void **)&data) == 0)
91              free(data);
92      }
93      if ((data = queue_peek(&queue)) != NULL)
94          fprintf(stdout, "Peeking at an empty queue...Value=%03d\n", *data);
95      else
96          fprintf(stdout, "Peeking at an empty queue...Value=NULL\n");
97
98      /*************************************************************************
99       *  Destroy the queue.
100      *************************************************************************
101
102     fprintf(stdout, "Destroying the queue\n");
103     queue_destroy(&queue);
```

Implementation of Queue

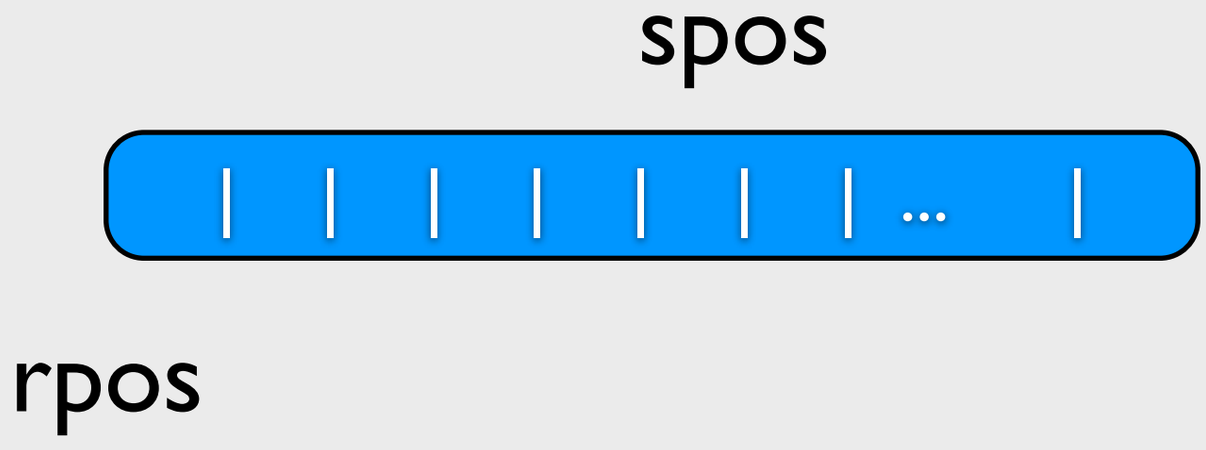
- Example: queue/test.c

```
82      if ((data = queue_peek(&queue)) != NULL)
83          fprintf(stdout, "Peeking at the head element...Value=%03d\n", *data);
84      else
85          fprintf(stdout, "Peeking at the head element...Value=NULL\n");
86
87      print_queue(&queue);
88      fprintf(stdout, "Dequeuing all elements\n");
89      while (queue_size(&queue) > 0) {
90          if (queue_dequeue(&queue, (void **)&data) == 0)
91              free(data);
92      }
93      if ((data = queue_peek(&queue)) != NULL)
94          fprintf(stdout, "Peeking at an empty queue...Value=%03d\n", *data);
95      else
96          fprintf(stdout, "Peeking at an empty queue...Value=NULL\n");
97
98      /**************************************************************************
99       *  Destroy the queue.
100      **************************************************************************
101
102     fprintf(stdout, "Destroying the queue\n");
103     queue_destroy(&queue);
```

```
Peeking at the head element...Value=006
Queue size is 7
queue[000]=006
queue[001]=007
queue[002]=008
queue[003]=009
queue[004]=010
queue[005]=100
queue[006]=200
Dequeuing all elements
Peeking at an empty queue...Value=NULL
Destroying the queue
```

Use Array for Queue

- queue_array
  - an array: p[ ]
  - use two indices: spos, rpos



Use Array for Queue

- Example: queue_array/queue_array.c

```
13 void enqueue(void) {
14     char s[256], *p;
15
16     do {
17         printf("spos %d: ", spos+1);
18         gets(s);
19         if(*s==0) {
20             break;
21         }
22         p = (char *) malloc(strlen(s)+1);
23         if(!p) {
24             printf("Out of memory.\n");
25             return;
26         }
27         strcpy(p, s);
28         if(*s) {
29             push(p);
30         }
31     } while(*s);
32 }
```

```
50 void push(char *q) {
51     if(spos==MAX) {
52         printf("List Full\n");
53         return;
54     }
55     p[spos] = q;
56     spos++;
57 }
```

Use Array for Queue

- Example: queue_array/queue_array.c

```
41 void dequeue(void) {
42     char *p;
43
44     if((p=pop())==NULL) {
45         return;
46     }
47     printf("%s\n", p);
48 }
```

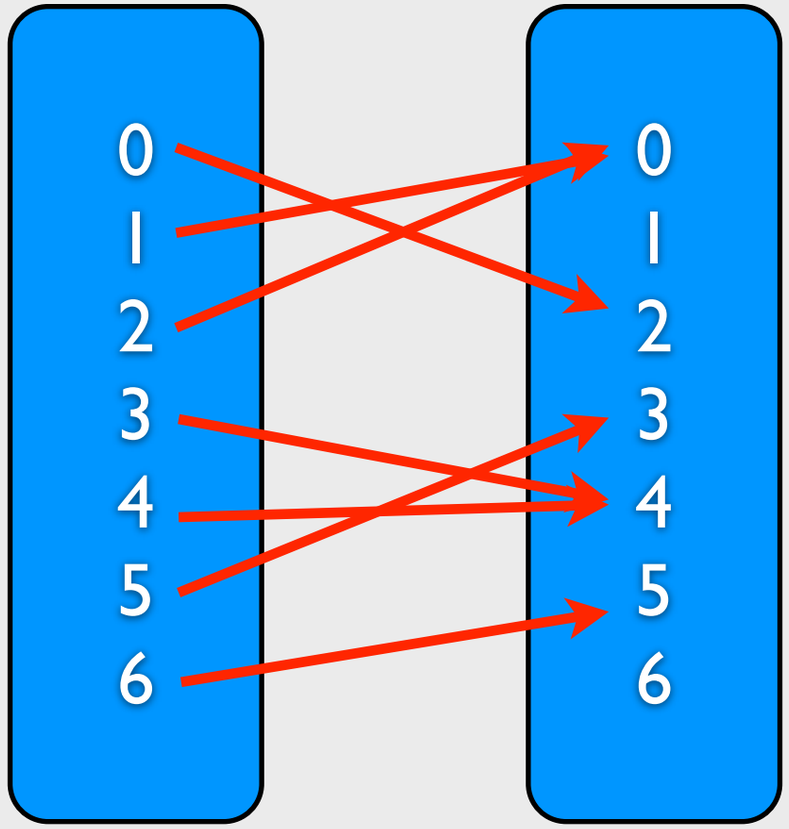
```
59 char *pop(void) {
60     if(rpos==spos) {
61         printf("No more.\n");
62         return NULL;
63     }
64     rpos++;
65     return p[rpos-1];
66 }
```

Example: Find 1to1 Function

- Given a Set: *S = {0, 1, 2, 3, ..., n}*, and a function from *S* to *S f: S -> S*. Please write a program that can find out a subset *S'* such that *f: S' -> S'* is a 1-to-1 and onto function and the the number of *S'* is maximal.
- For example
  - S = {0, 1, 2, 3, 4, 5, 6}
  - f: f(0) = 2; f(1)=f(2)=0; f(3)=f(4)=4; f(5)=3; f(6)=5

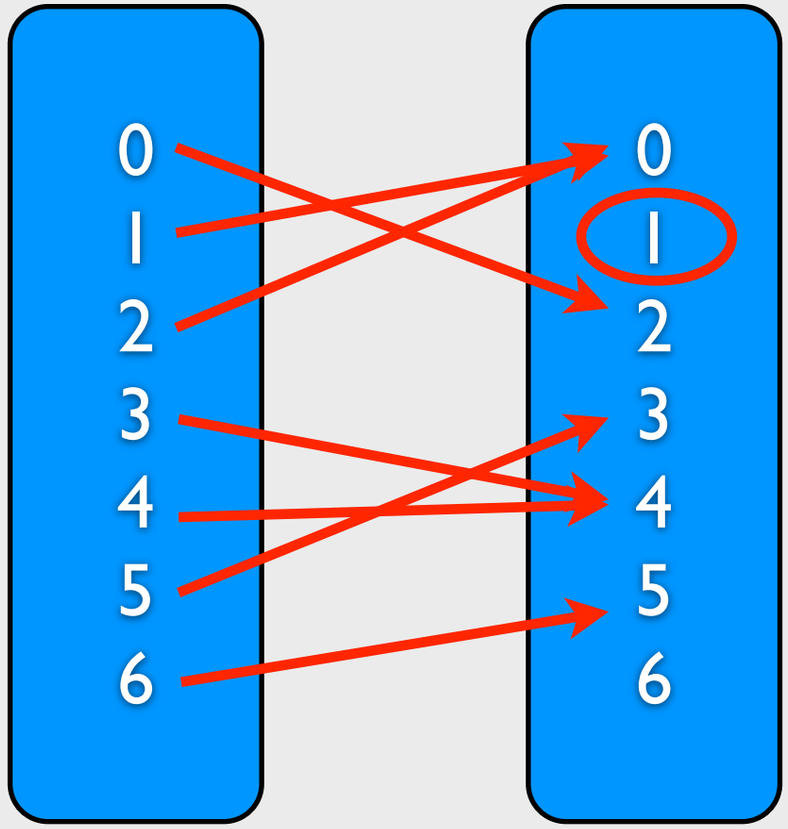
Example: Find 1to1 Function

- For example
  - S = {0, 1, 2, 3, 4, 5, 6}
  - f: f(0) = 2; f(1)=f(2)=0; f(3)=f(4)=4; f(5)=3; f(6)=5



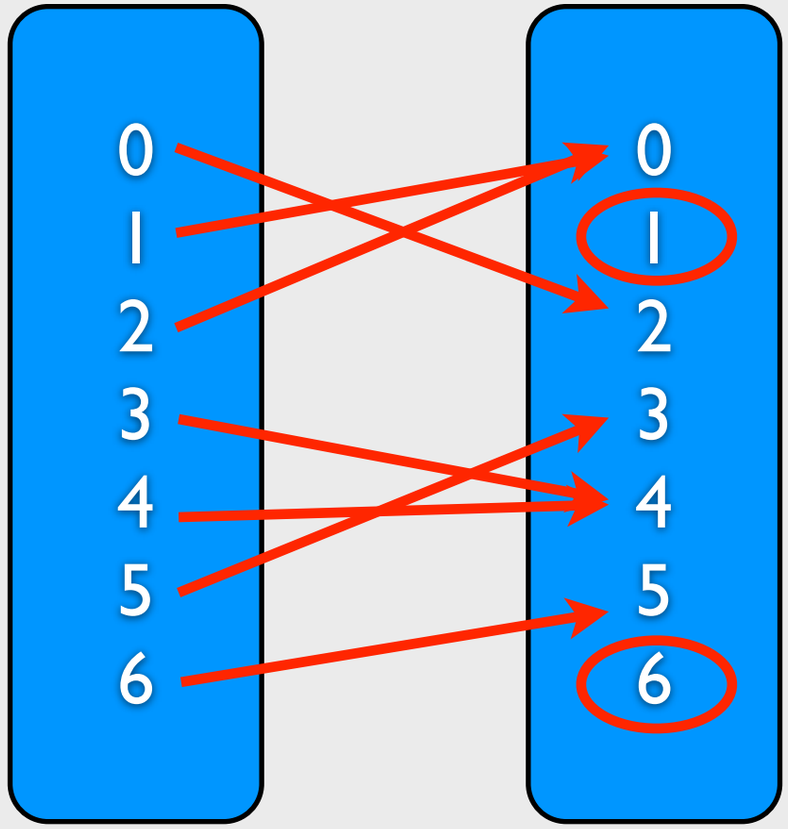
Example: Find 1to1 Function

- For example
  - S = {0, 1, 2, 3, 4, 5, 6}
  - f: f(0) = 2; f(1)=f(2)=0; f(3)=f(4)=4; f(5)=3; f(6)=5



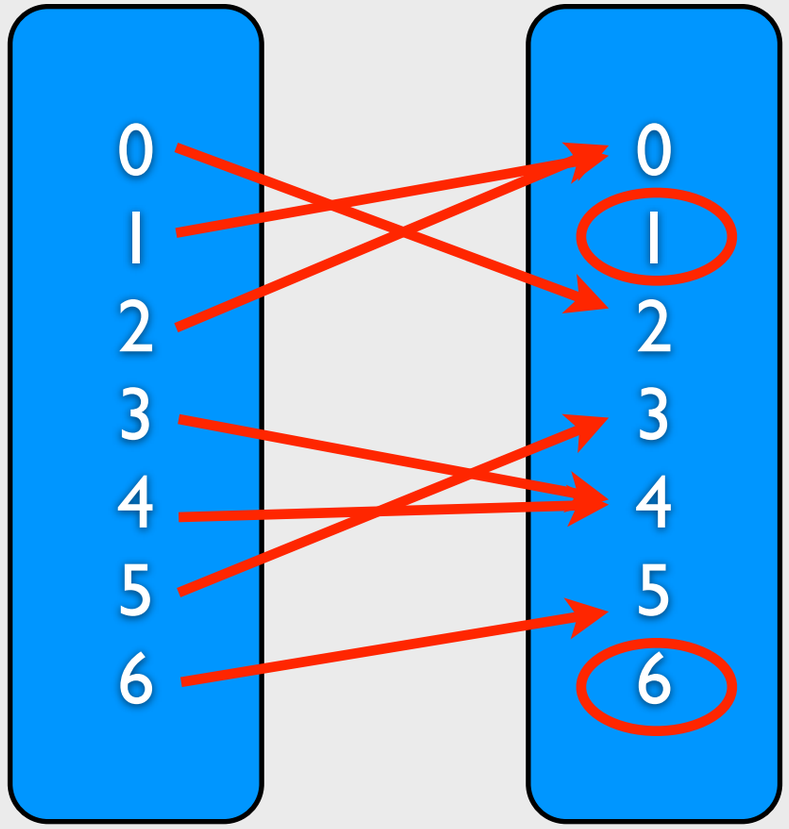
Example: Find 1to1 Function

- For example
  - S = {0, 1, 2, 3, 4, 5, 6}
  - f: f(0) = 2; f(1)=f(2)=0; f(3)=f(4)=4; f(5)=3; f(6)=5

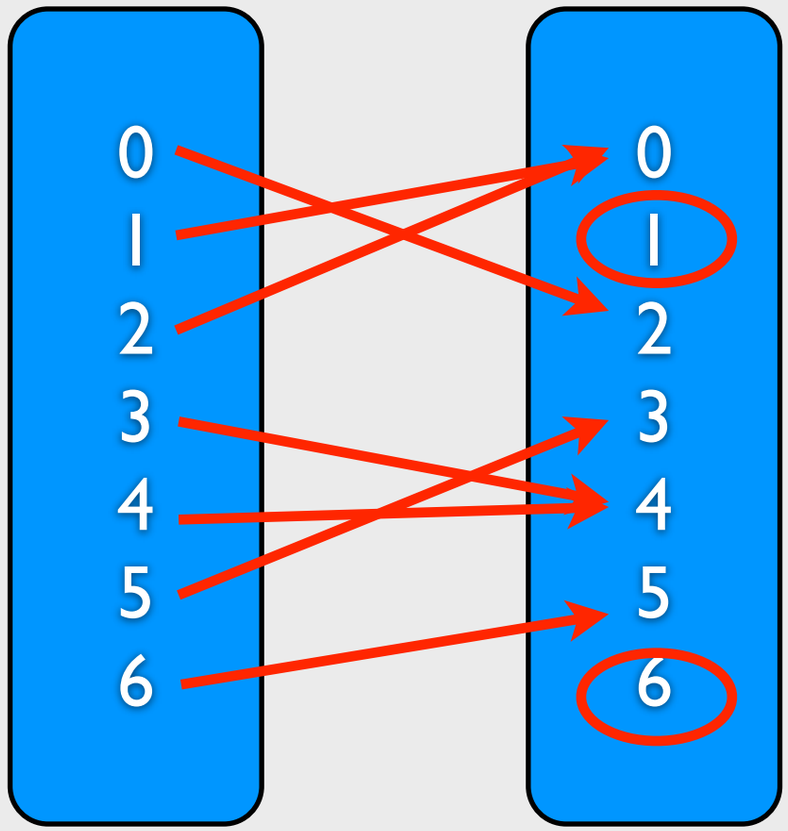


Example: Find 1to1 Function

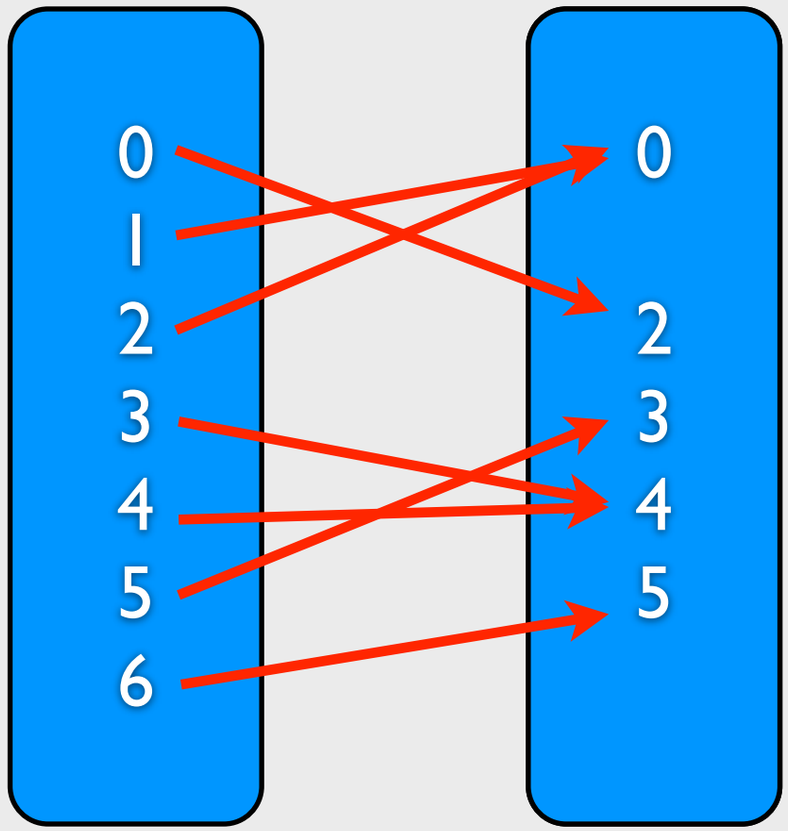
- Possible S’
  - S’ = {4}, then f: S’ -> S’ is 1-1 and onto
  - S’ = {0,2,4}, then f: S’ -> S’ is also 1-1 and onto



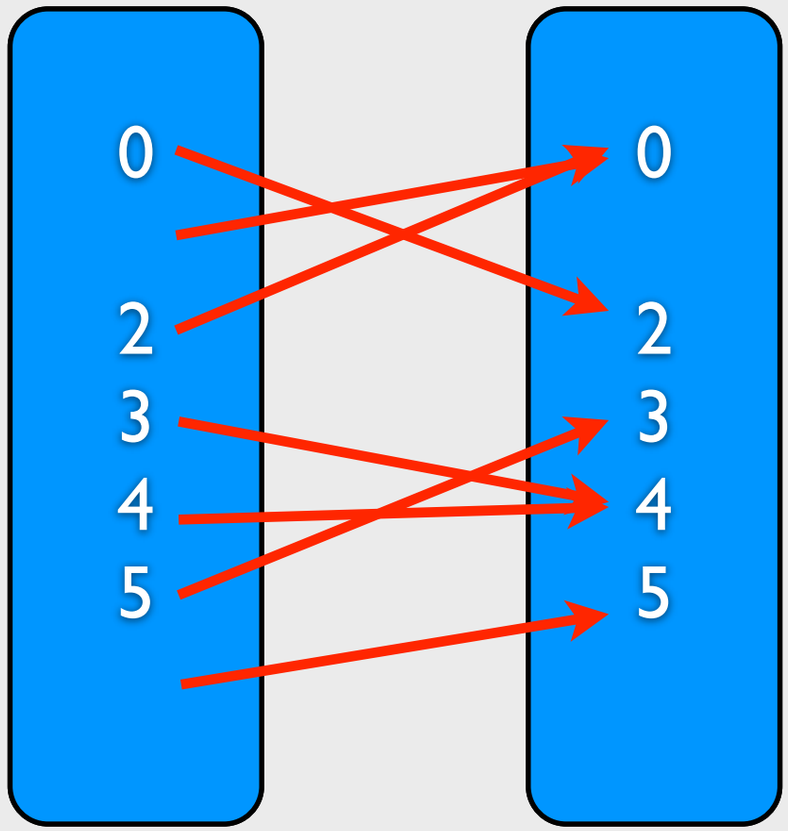
How to find the S’



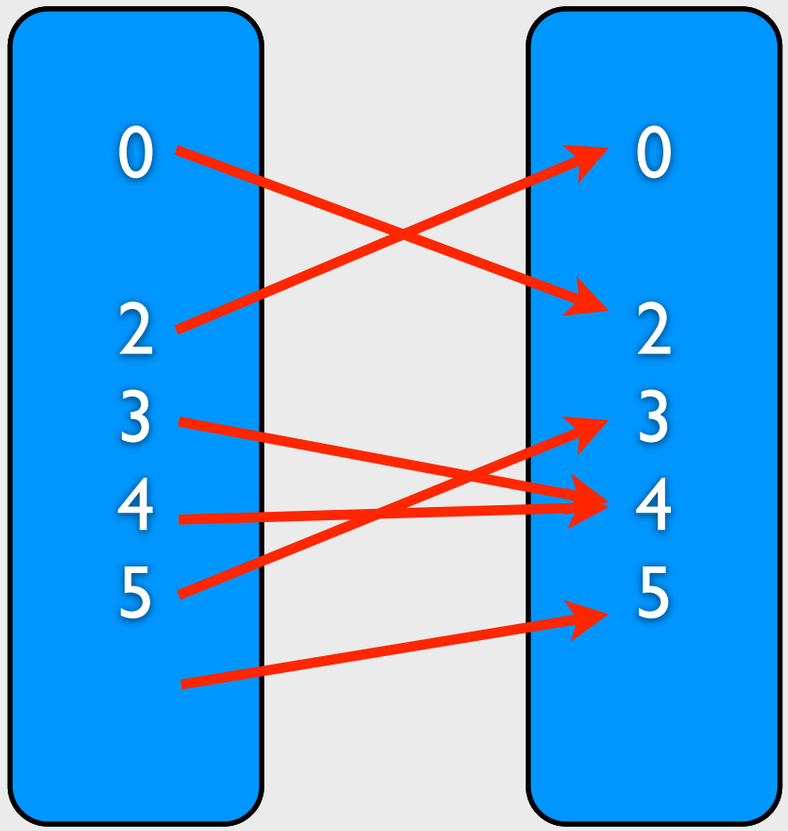
How to find the S’



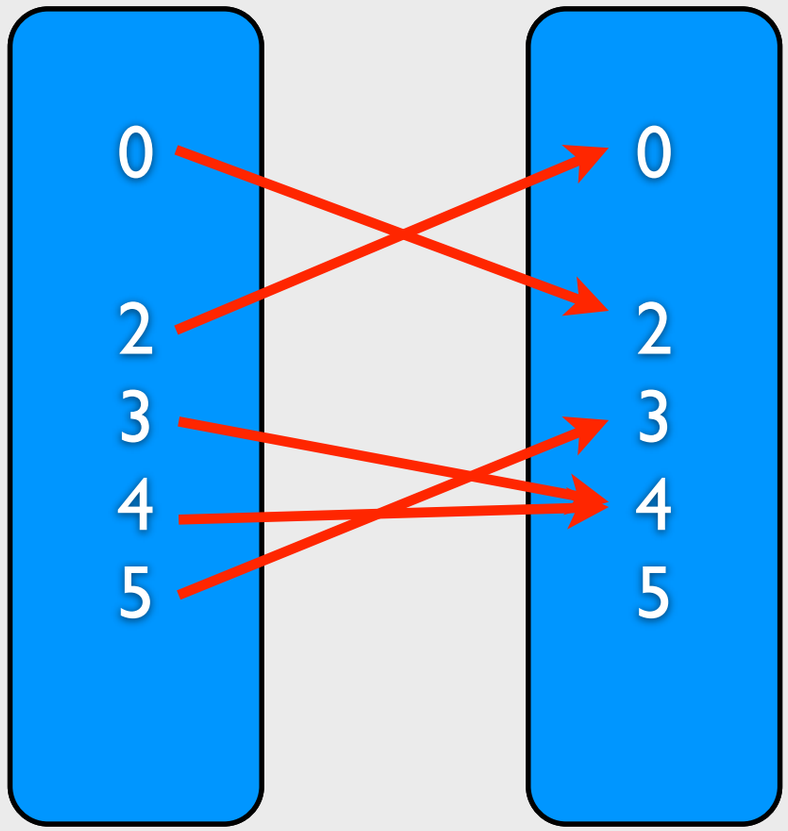
How to find the S’



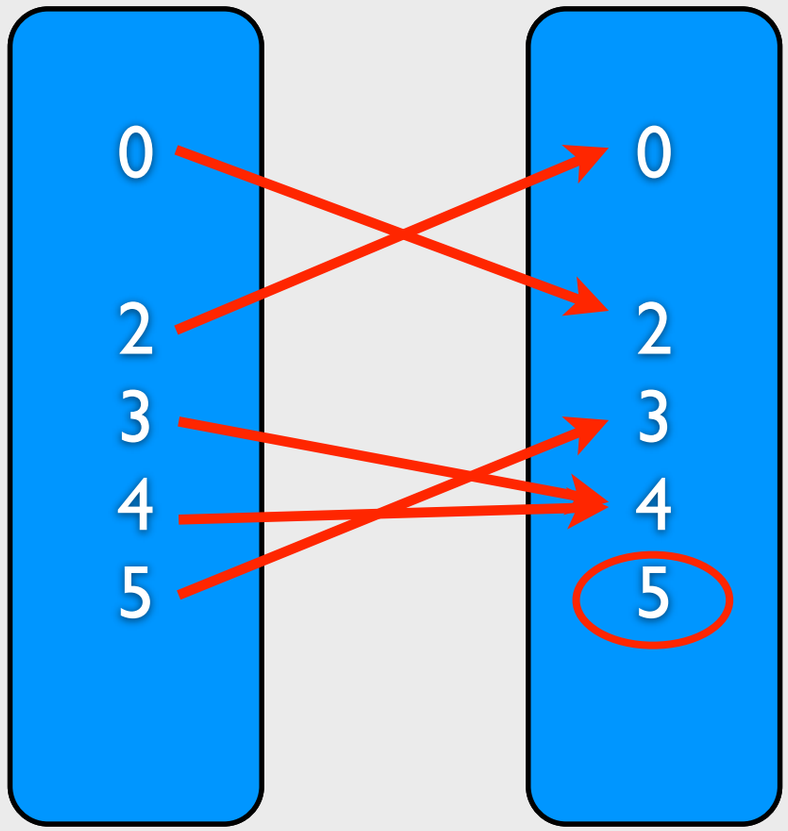
How to find the S’



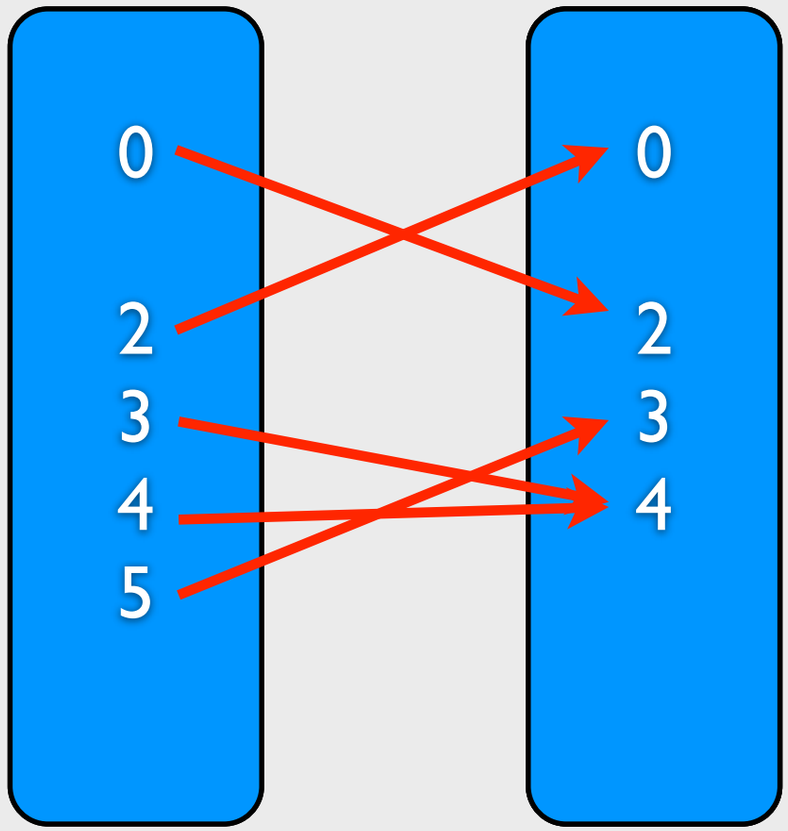
How to find the S’



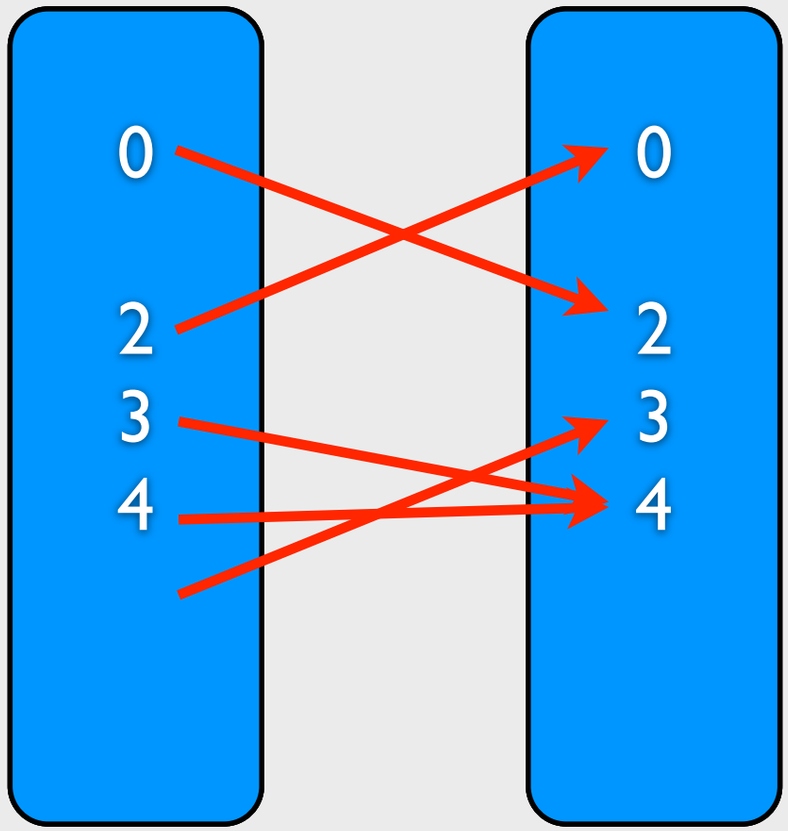
How to find the S’



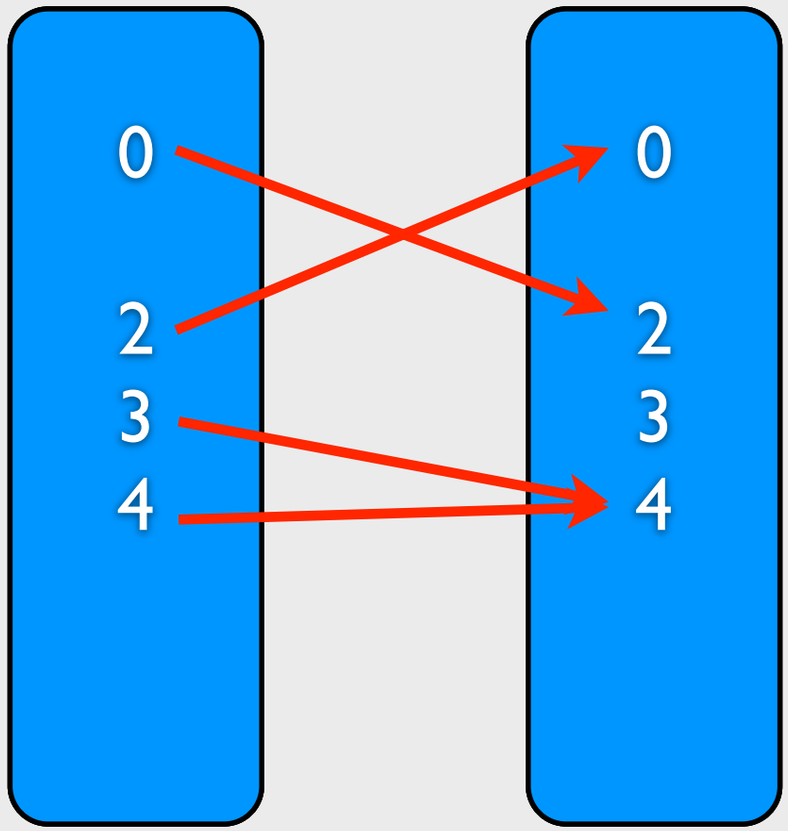
How to find the S’



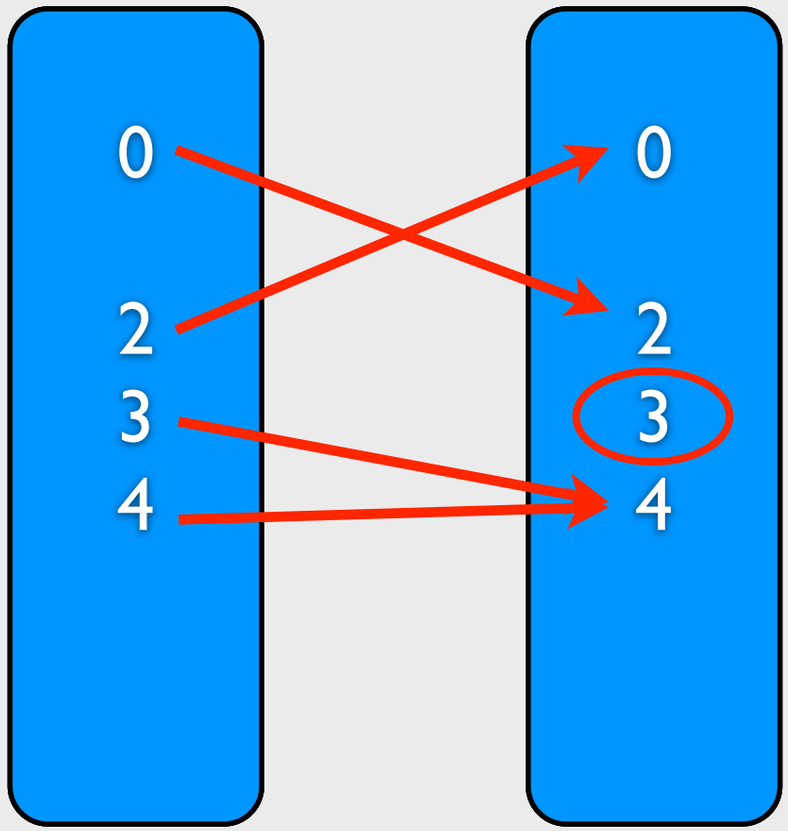
How to find the S’



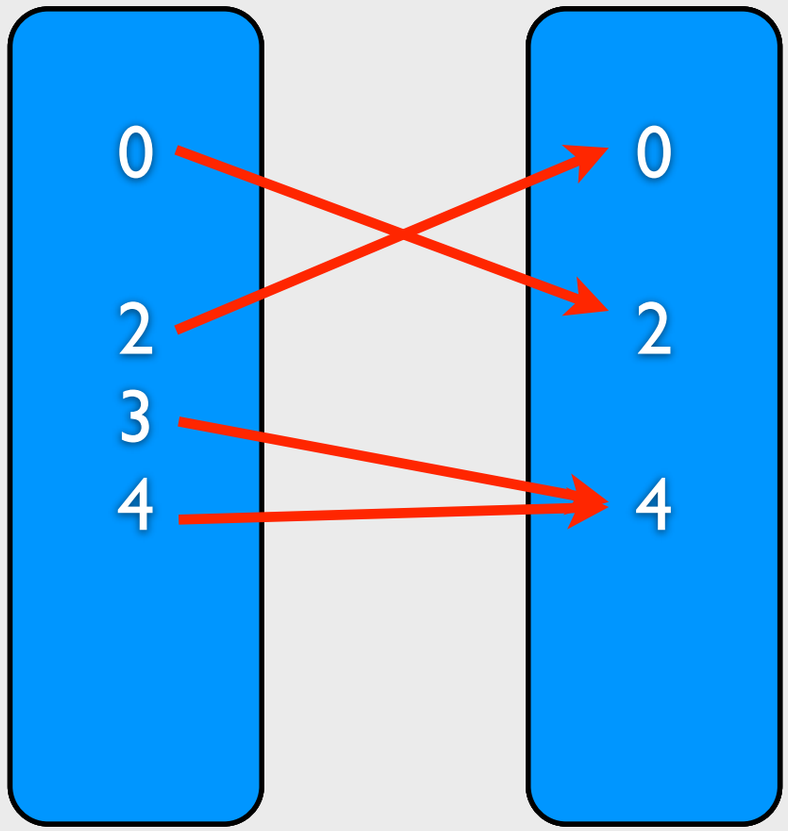
How to find the S’



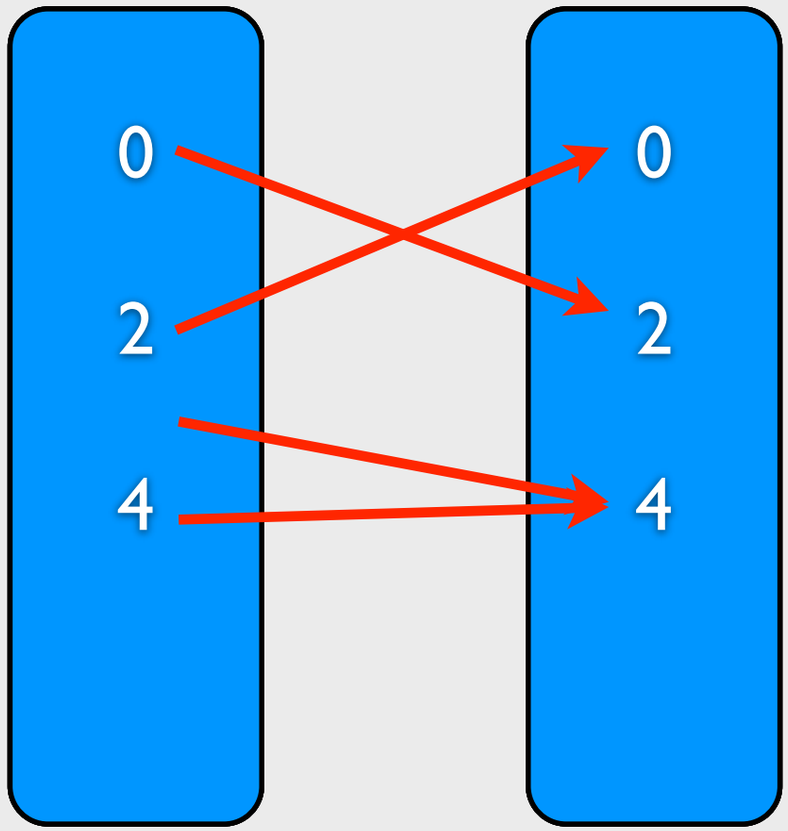
How to find the S’



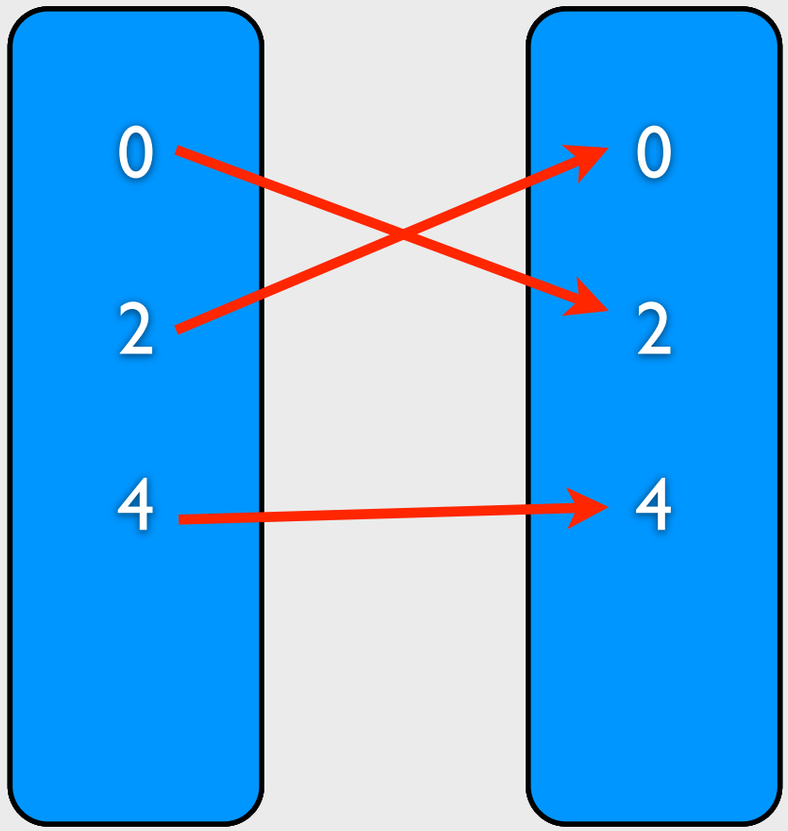
How to find the S’



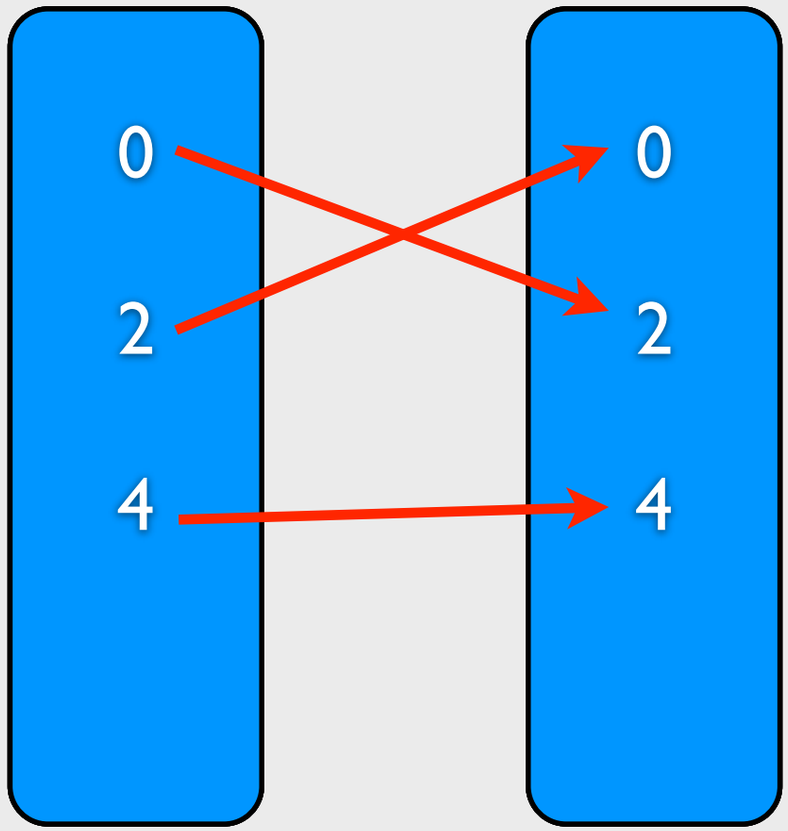
How to find the S’



How to find the S’



How to find the S’



So, S’ = {0, 2, 4}

Main Idea

- Use a queue to maintain those numbers without any mappings
- Then, iterate these numbers, and add new such numbers
- Finally, find out the S’ set, which is the maximal 1to1 and onto subset

Example: 1to1/1to1.c

```c
int main(void)
{
     int  funct_table[7] = { 2, 0, 0, 4, 4, 3, 5};
     int  n = sizeof(funct_table)/sizeof(int);
     int  status[sizeof(funct_table)/sizeof(int)];
     int  counter[sizeof(funct_table)/sizeof(int)];
     int  i;

     printf("\nOne-To-One Function Construction Program");
     printf("\n========================================\n");
     printf("\nDomain    Range    Status");
     printf("\n------    -----    ------");

     find_one_to_one(funct_table, status, counter, n);

     for (i = 0; i < n; i++) {
          printf("\n%4d%10d", i, funct_table[i]);
          if (status[i] == SAVED)
               printf("        SAVED");
          else
               printf("       DELETED");
     }

     printf("\n\nConstructed New 1-1 Function\n");
     printf("\nDomain    Range");
     printf("\n------    -----");
     for (i = 0; i < n; i++)
          if (status[i] == SAVED)
               printf("\n%4d%10d", i, funct_table[i]);
     printf("\n");
     return 0;
```

Example: 1to1/1to1.c

```c
void find_one_to_one(int  funct[], int status[], int counter[], int n)
{
     int  queue[QUEUE_SIZE];  /* we need a queue          */
     int  head, tail;         /* queue pointers           */
     int  i, j;

     for (i = 0; i < n; i++) { /* initialization          */
          counter[i] = 0;     /* size of inverse-images   */
          status[i]  = SAVED; /* assume all are SAVED     */
     }

     for (i = 0; i < n; i++)  /* count inverse-image size */
          counter[funct[i]]++;

     for (tail = -1, i = 0; i < n; i++) /* put all i such */
          if (counter[i] == 0) /* that counter[i]=0 to Q  */
               queue[++tail] = i;

     head = 0;                /* main loop. start from H  */
     while (head <= tail) {   /* if there have elements   */
          j = queue[head++];  /* get it and put it to j   */
          status[j] = DELETED;/* delete it. no inv-image  */
          if (--counter[funct[j]] == 0)
               queue[++tail] = funct[j];
     }
}
```

Example: 1to1/1to1.c

```
21 void find_one_to_one(int  funct[], int status[], int counter[], int n)
22 {
23      int  queue[QUEUE_SIZE];  /* we need a queue          */
24      int  head, tail;         /* queue pointers           */
25      int  i, j;
26
27      for (i = 0; i < n; i++) { /* initialization          */
28           counter[i] = 0;     /* size of inverse-images   */
29           status[i]  = SAVED; /* assume all are SAVED     */
30      }
31
32      for (i = 0; i < n; i++)  /* count inverse-image size */
33           counter[funct[i]]++;
34
35      for (tail = -1, i = 0; i < n; i++) /* put all i such */
36           if (counter[i] == 0) /* that counter[i]=0 to Q  */
37                queue[++tail] = i;
38
39      head = 0;                /* main loop. start from H  */
40      while (head <= tail) {   /* if there have elements   */
41           j = queue[head++];  /* get it and put it to j   */
42           status[j] = DELETED;/* delete it. no inv-image  */
43           if (--counter[funct[j]] == 0)
44                queue[++tail] = funct[j];
45      }
46 }
```

Think over 1to1 example

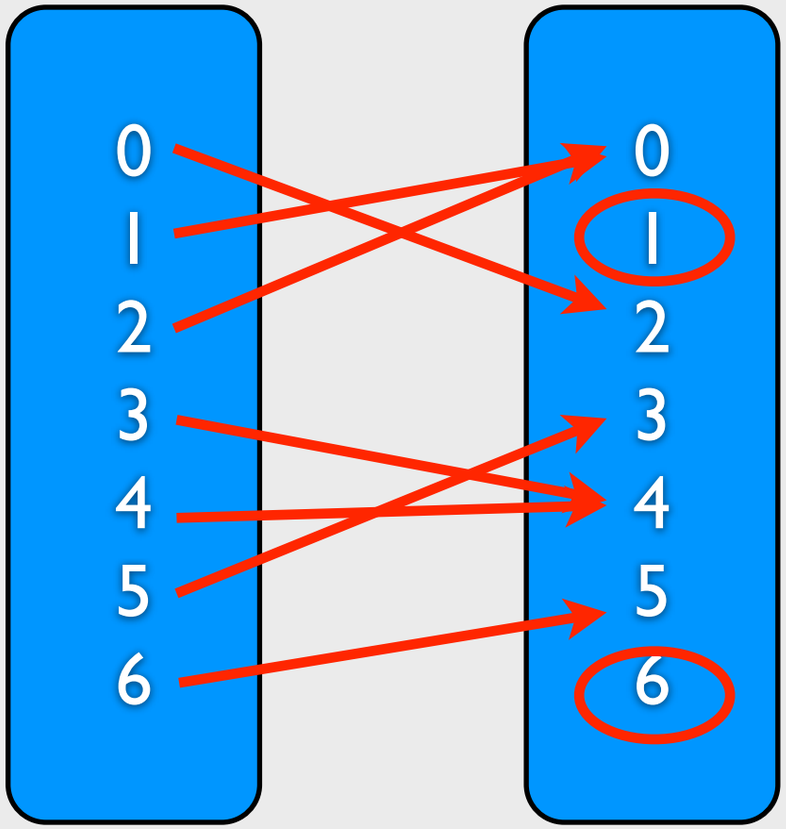
- Why we can find out the S’ when the queue is empty?
- What if we replace the queue as stack?
- What is the worst case? How many times will be executed?

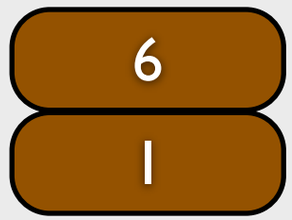
Think over 1to1 example

- Why we can find out the S’ when the queue is empty?
- That is because the queue is to store the numbers without any mappings. So, when we delete these numbers, the remaining numbers form the S’ set.

Think over 1to1 example

- What if we replace the queue as stack?

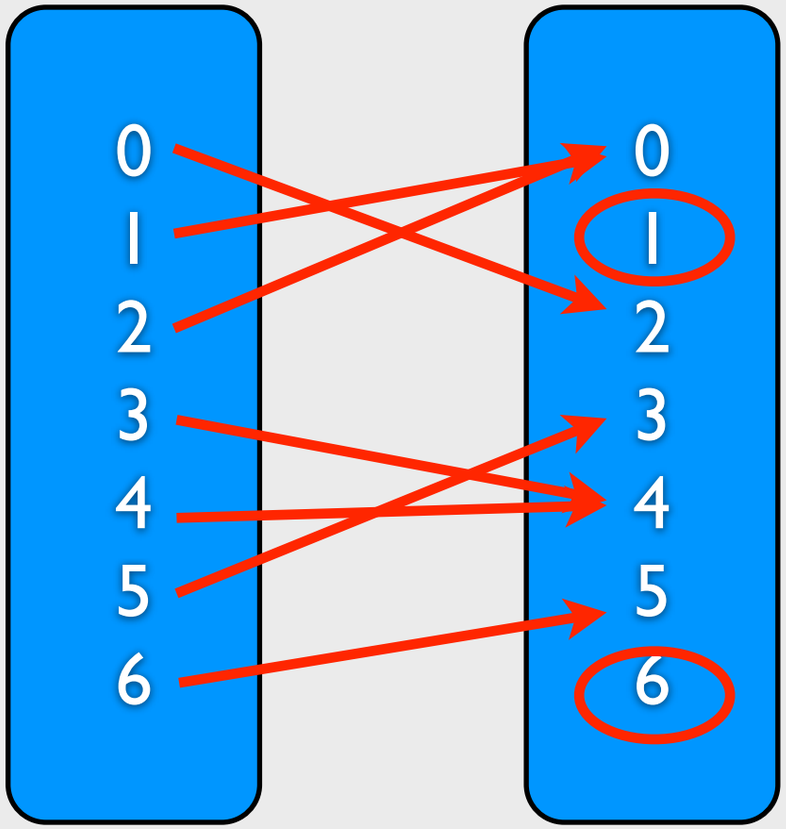




stack

Think over 1to1 example

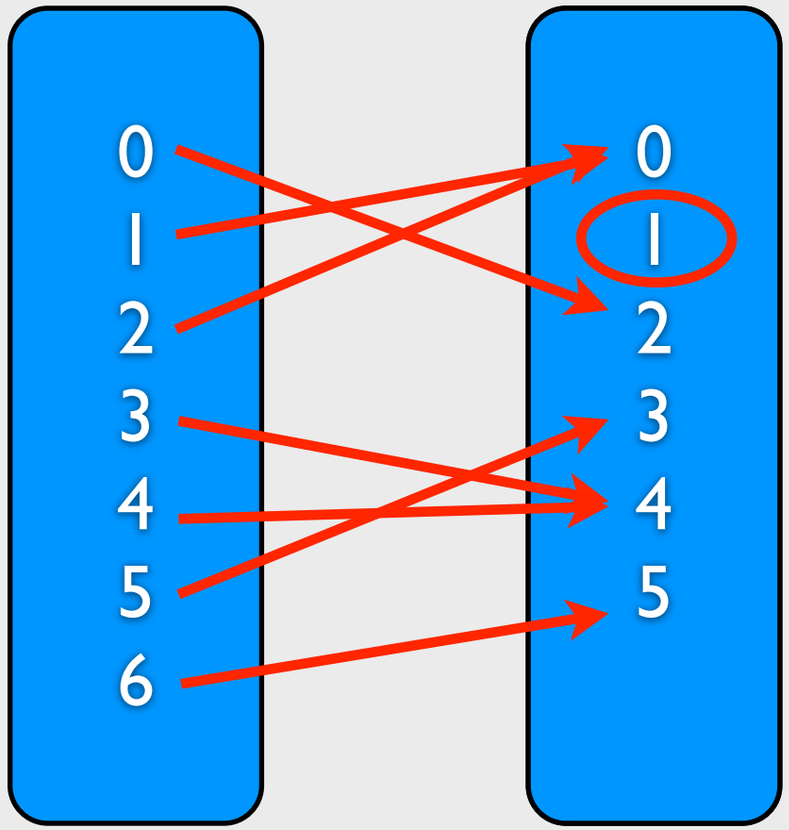
- What if we replace the queue as stack?



stack

Think over 1to1 example

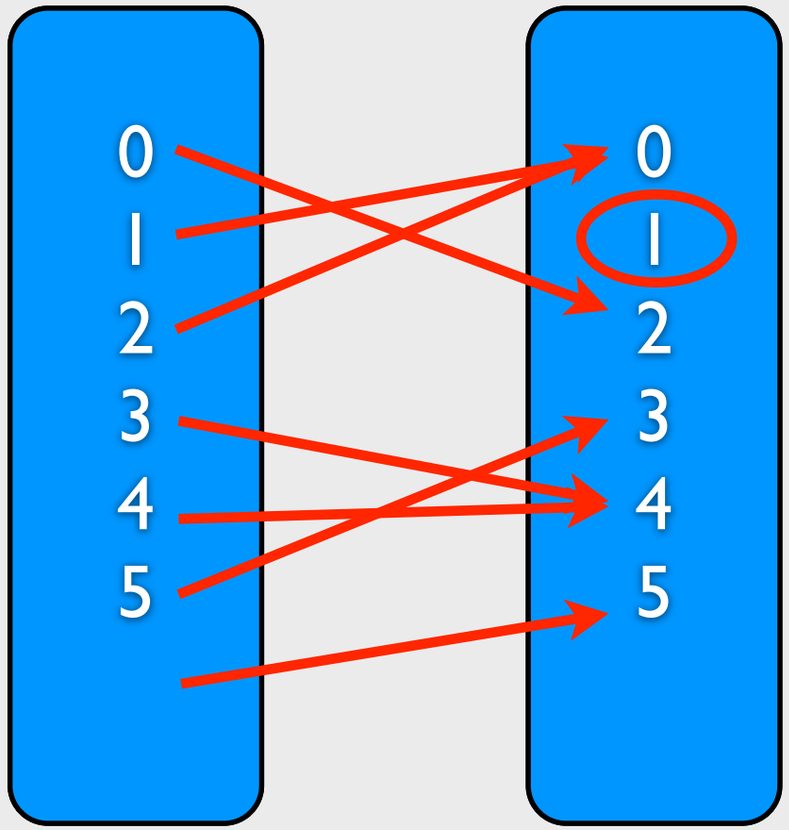
- What if we replace the queue as stack?



stack

Think over 1to1 example

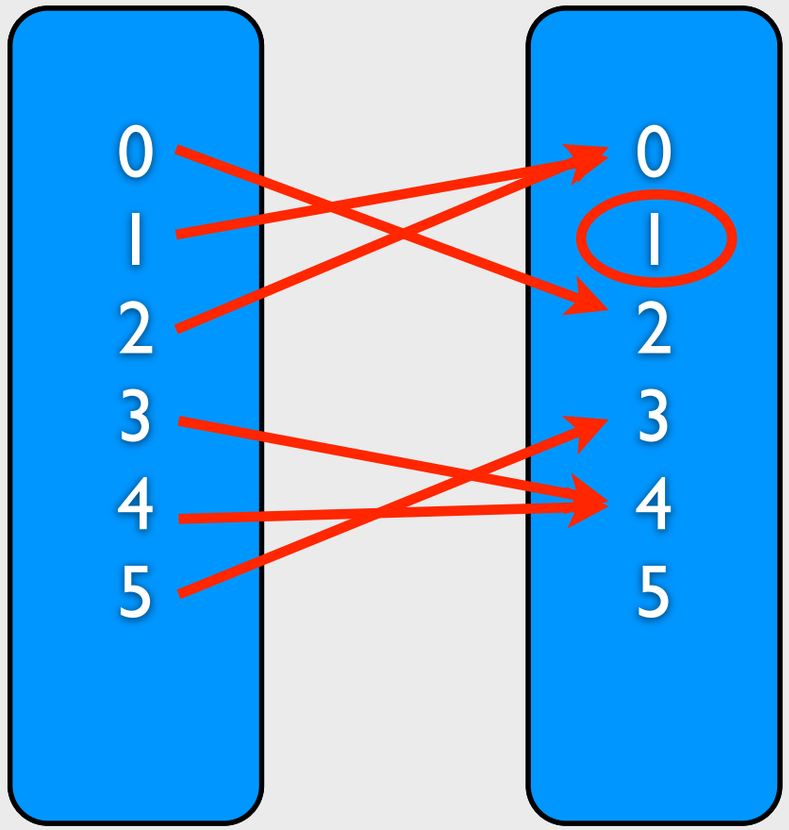
- What if we replace the queue as stack?



stack

Think over 1to1 example

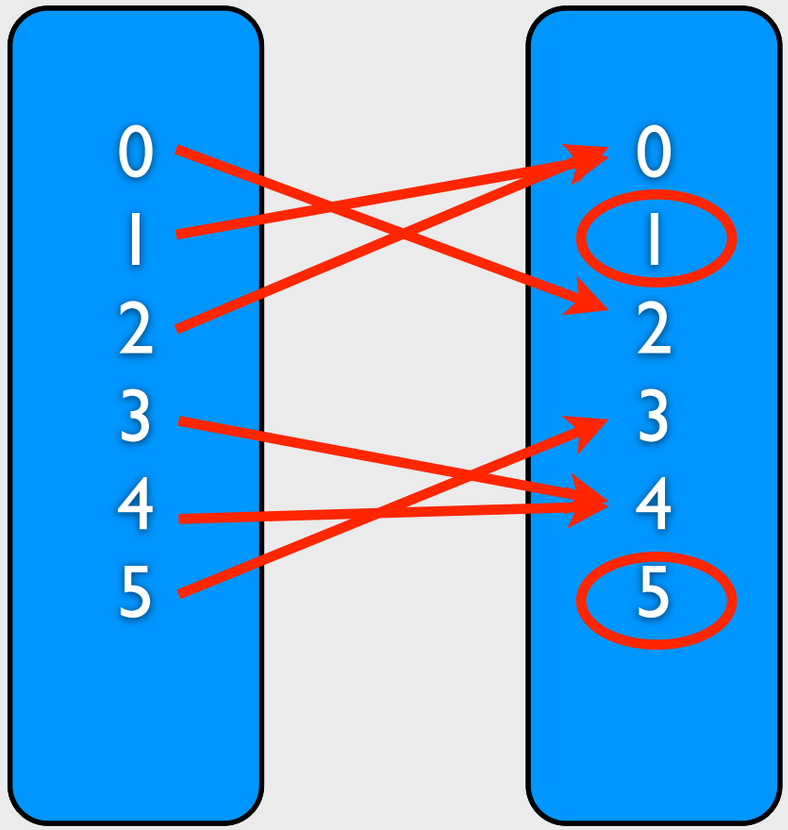
- What if we replace the queue as stack?



stack

Think over 1to1 example

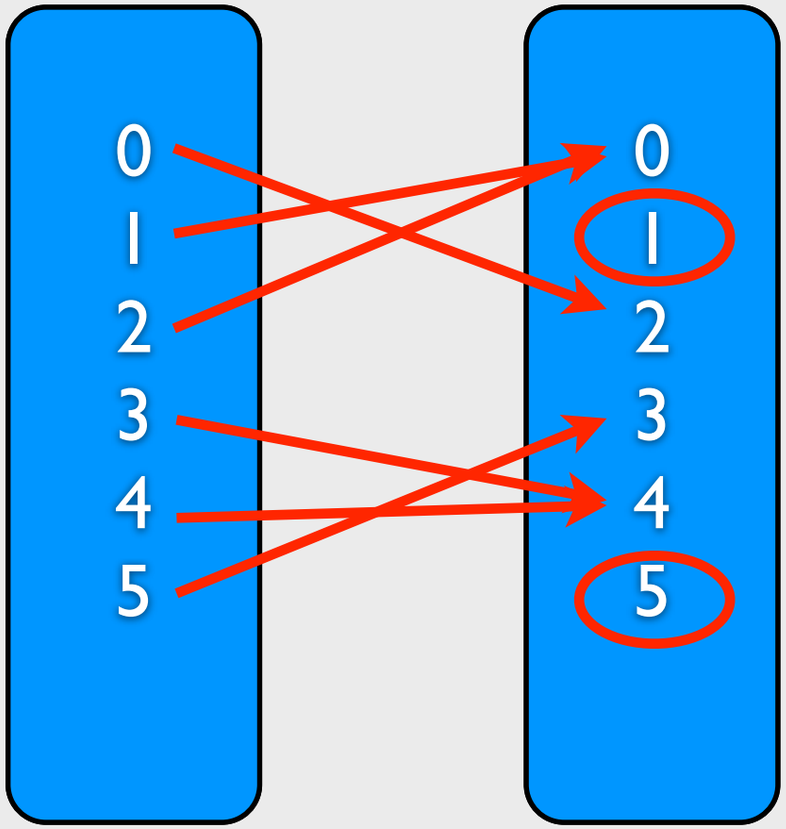
- What if we replace the queue as stack?

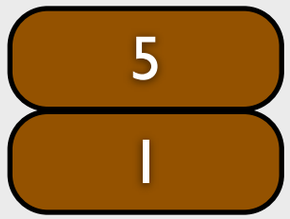


stack

Think over 1to1 example

- What if we replace the queue as stack?

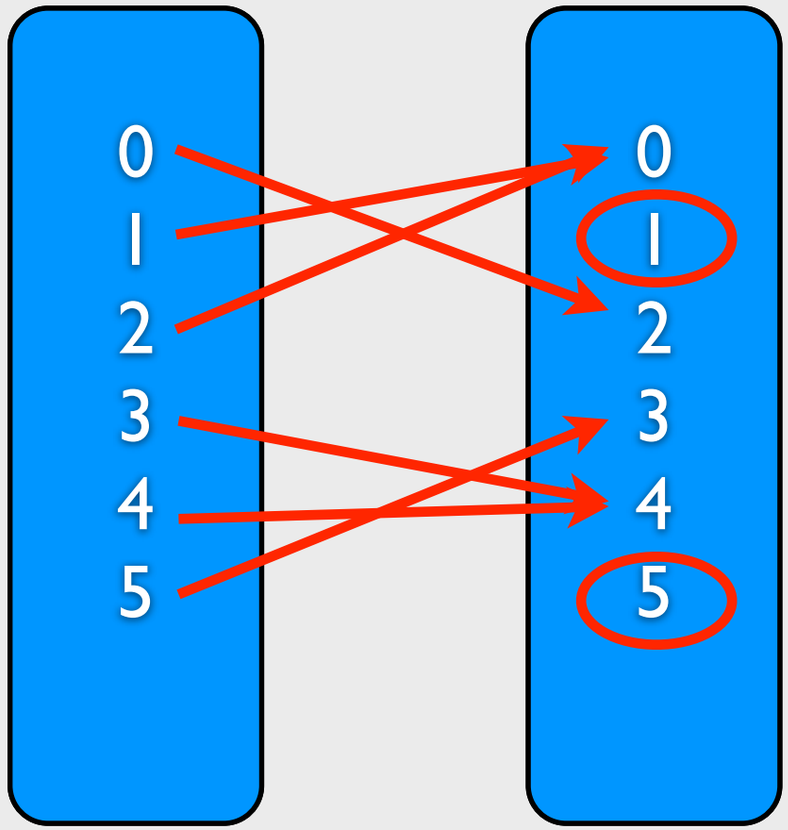




stack

Think over 1to1 example

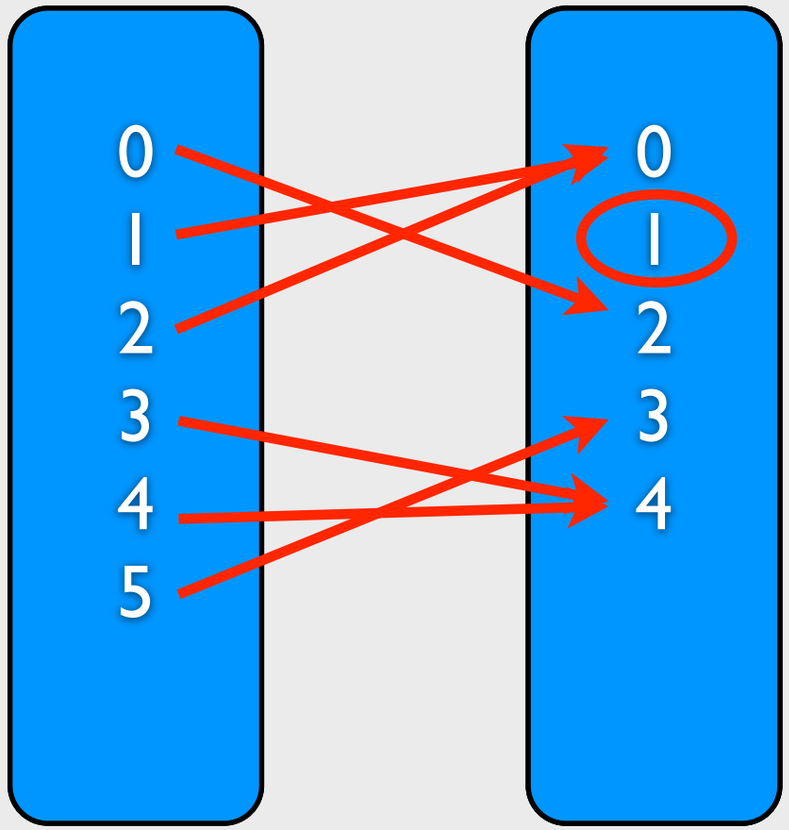
- What if we replace the queue as stack?



stack

Think over 1to1 example

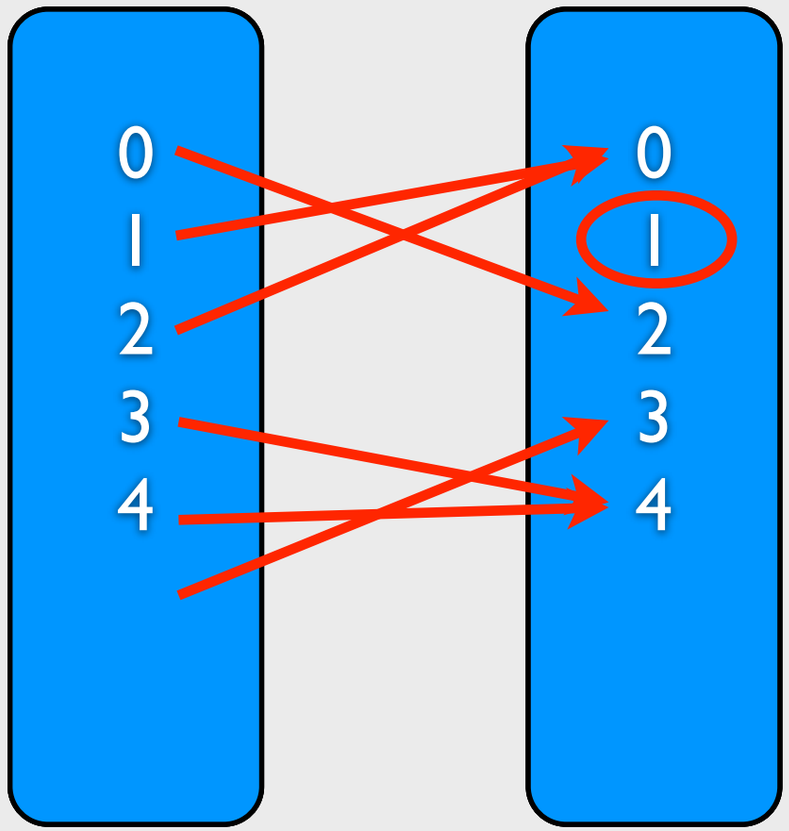
- What if we replace the queue as stack?



stack

Think over 1to1 example

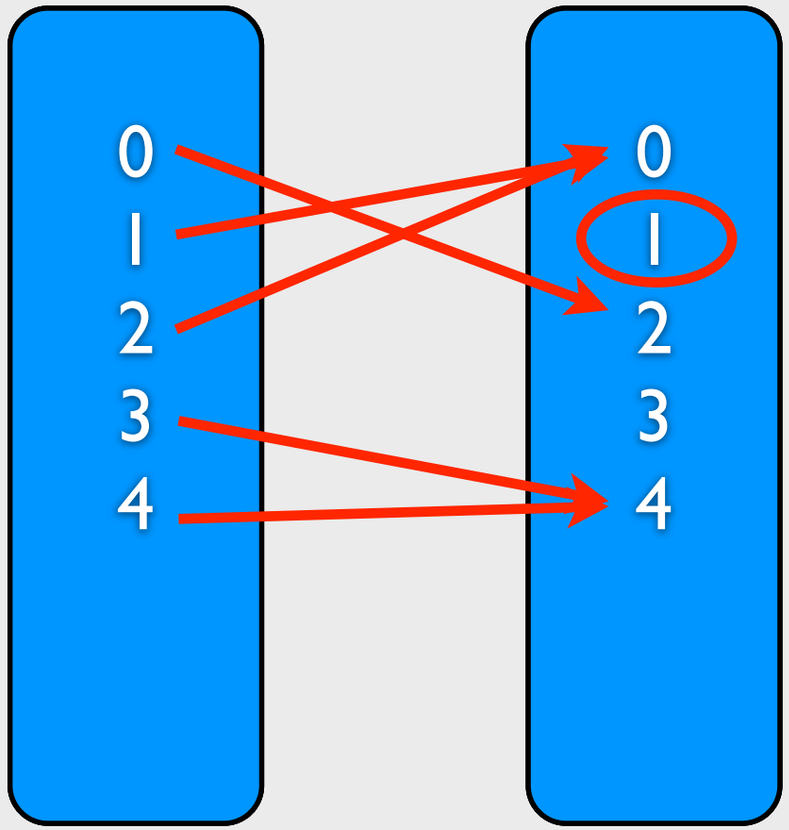
- What if we replace the queue as stack?



stack

Think over 1to1 example

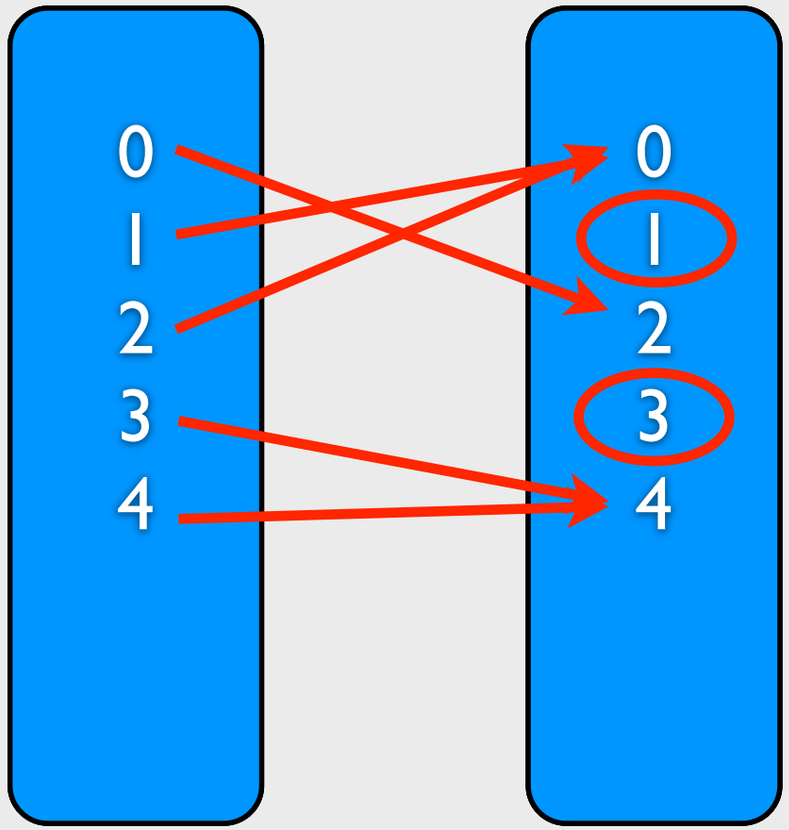
- What if we replace the queue as stack?



stack

Think over 1to1 example

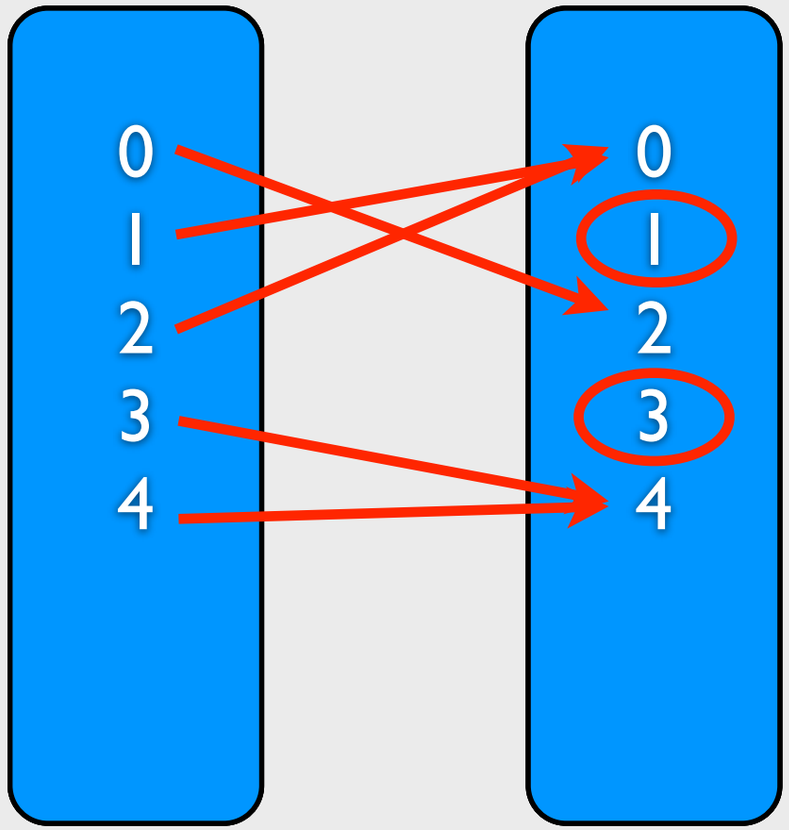
- What if we replace the queue as stack?

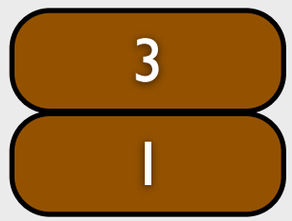


stack

Think over 1to1 example

- What if we replace the queue as stack?

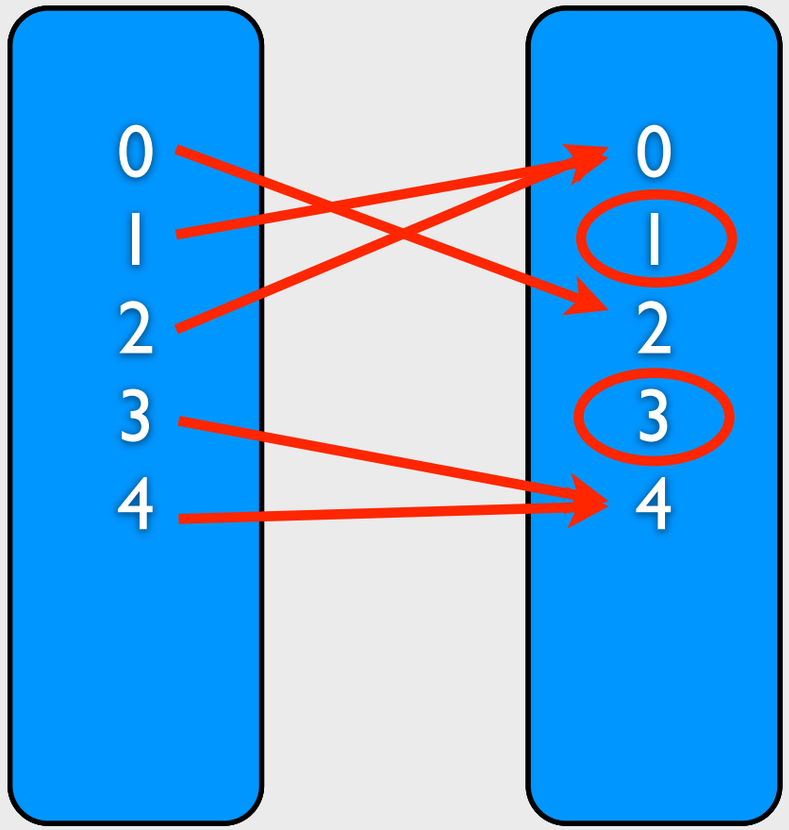




stack

Think over 1to1 example

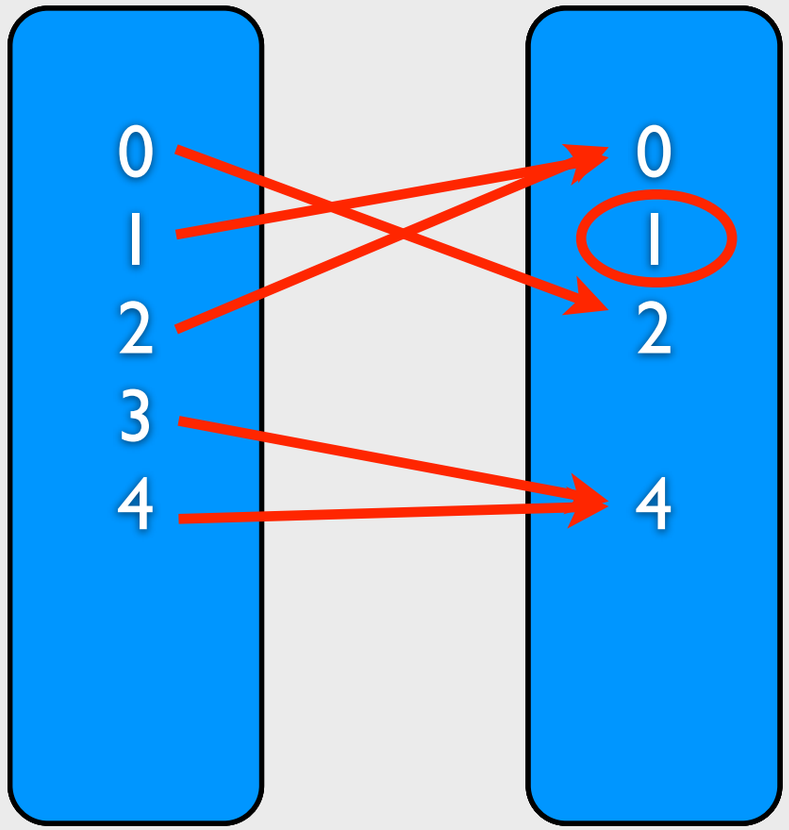
- What if we replace the queue as stack?



stack

Think over 1to1 example

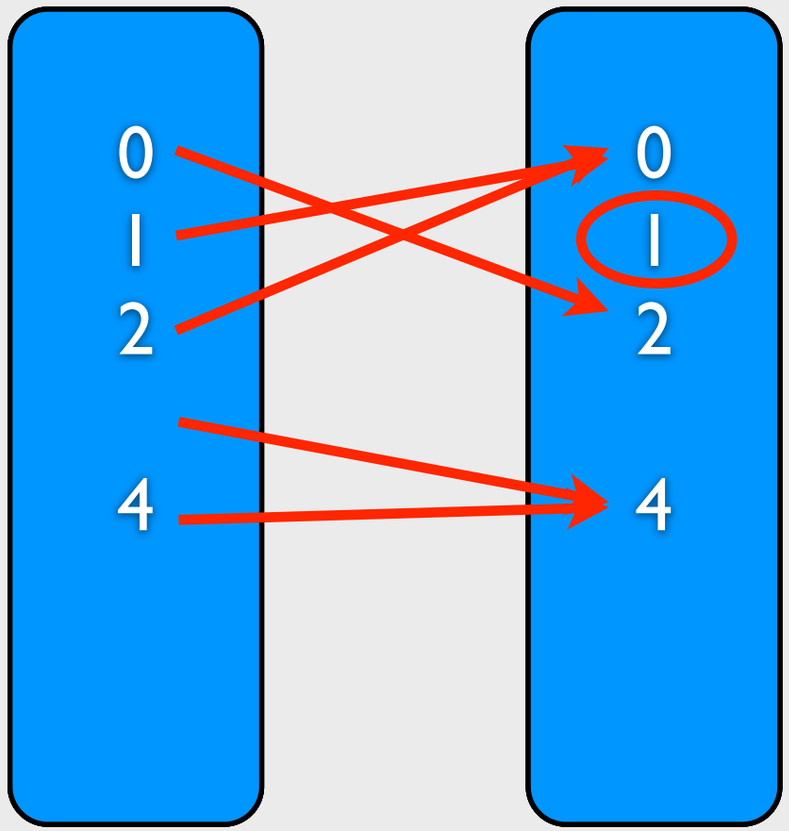
- What if we replace the queue as stack?



stack

Think over 1to1 example

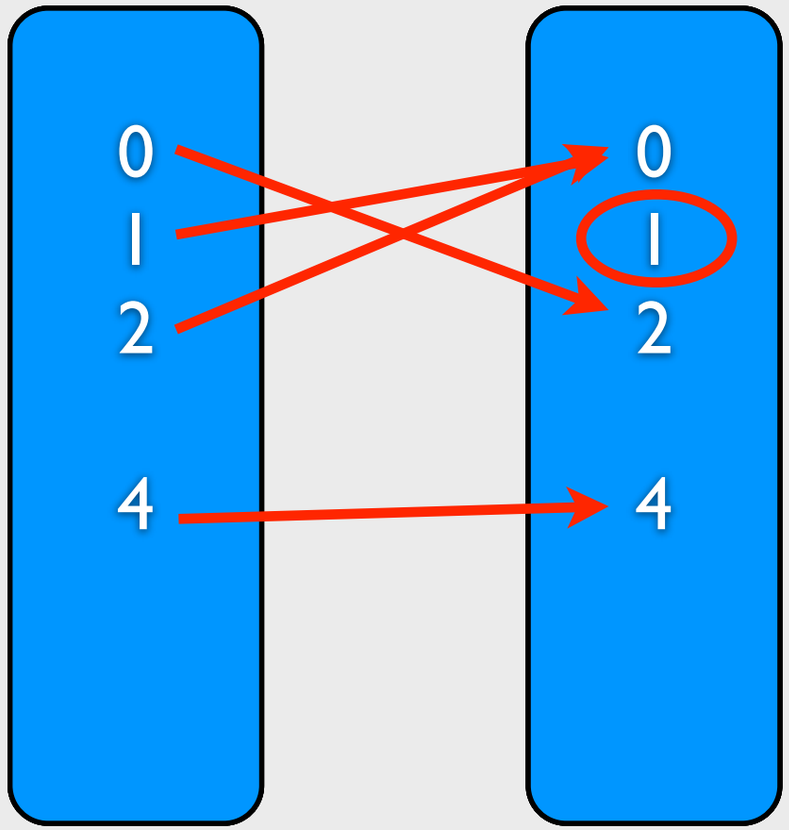
- What if we replace the queue as stack?



stack

Think over 1to1 example

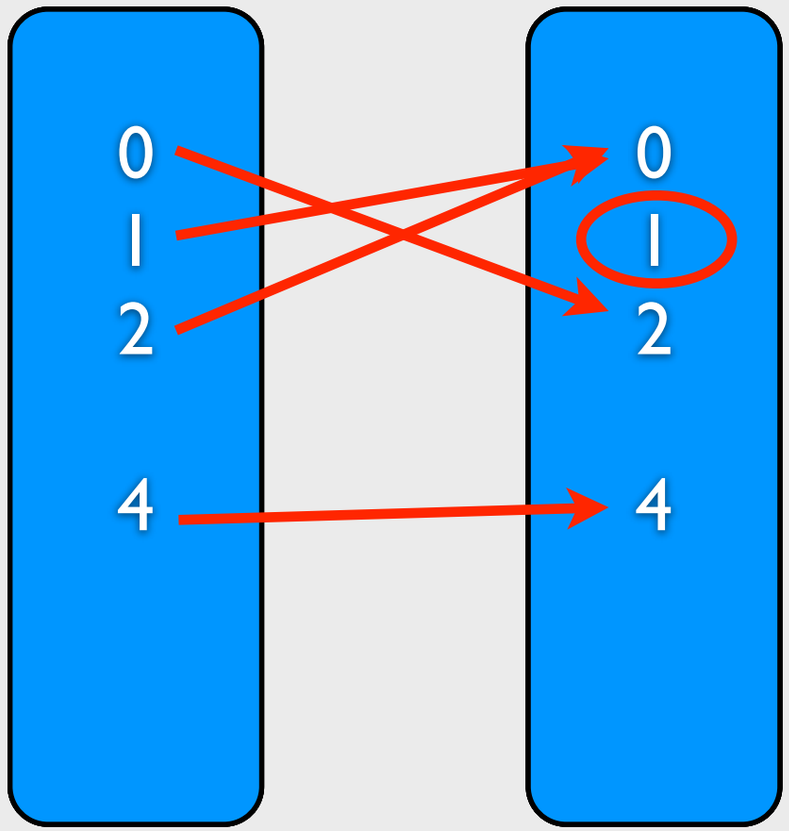
- What if we replace the queue as stack?



stack

Think over 1to1 example

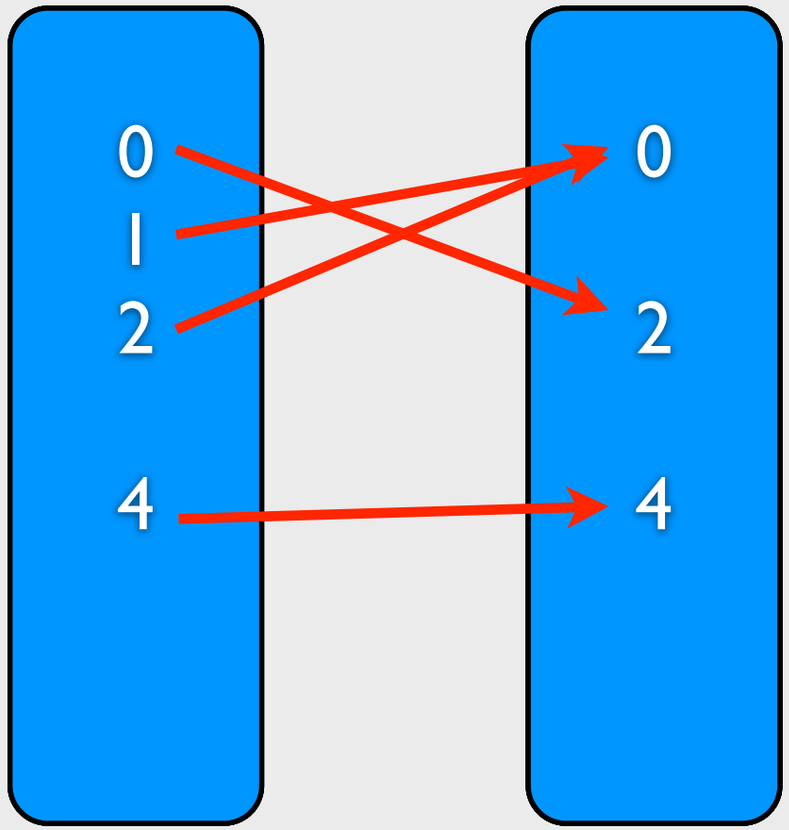
- What if we replace the queue as stack?



stack

Think over 1to1 example

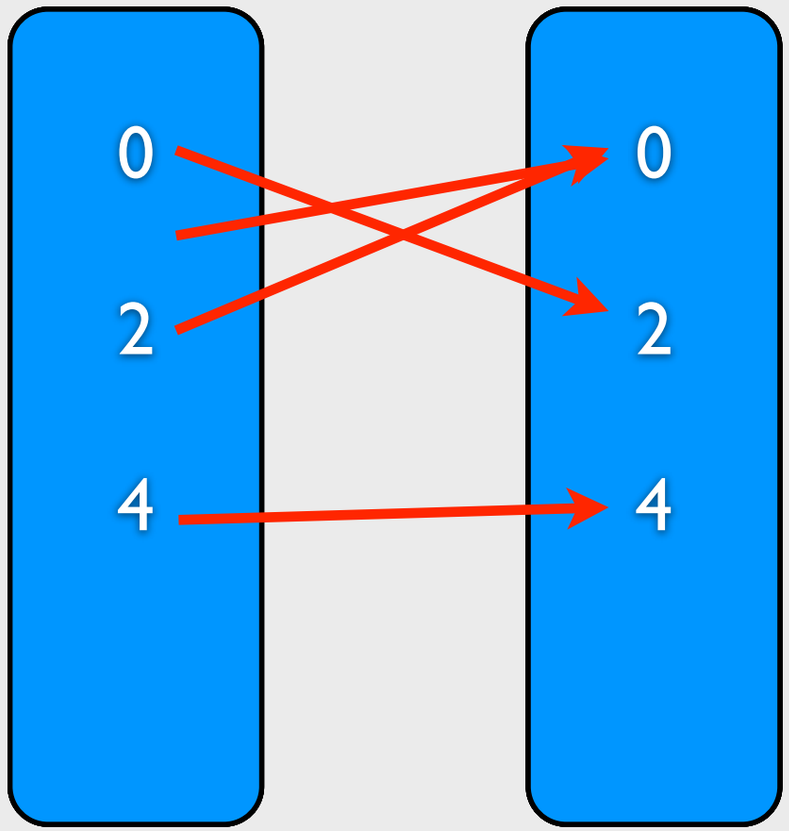
- What if we replace the queue as stack?



stack

Think over 1to1 example

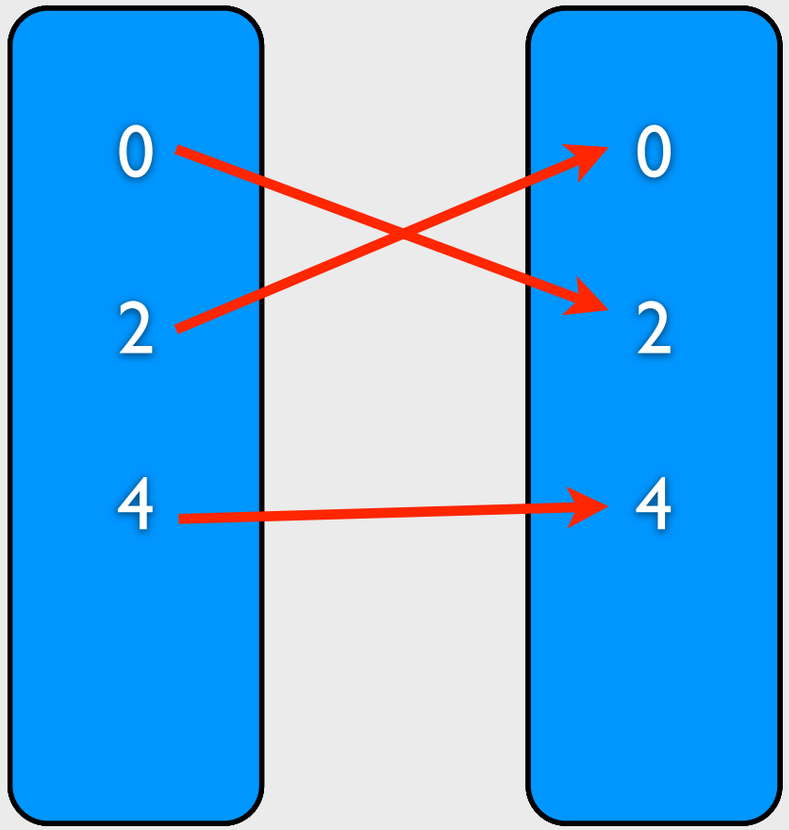
- What if we replace the queue as stack?



stack

Think over 1to1 example

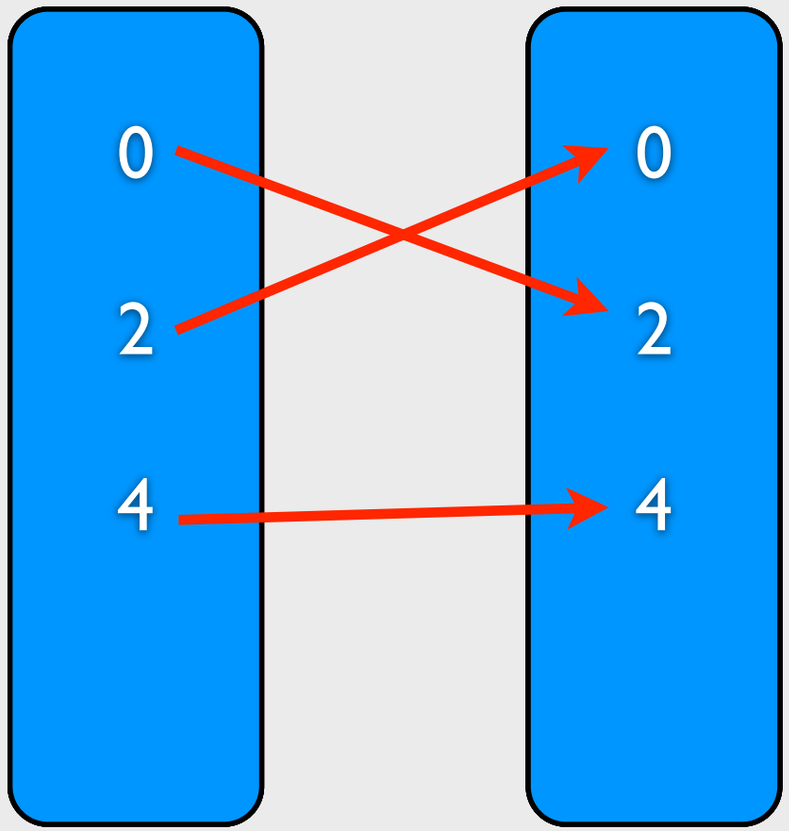
- What if we replace the queue as stack?



stack

Think over 1to1 example

- What if we replace the queue as stack?



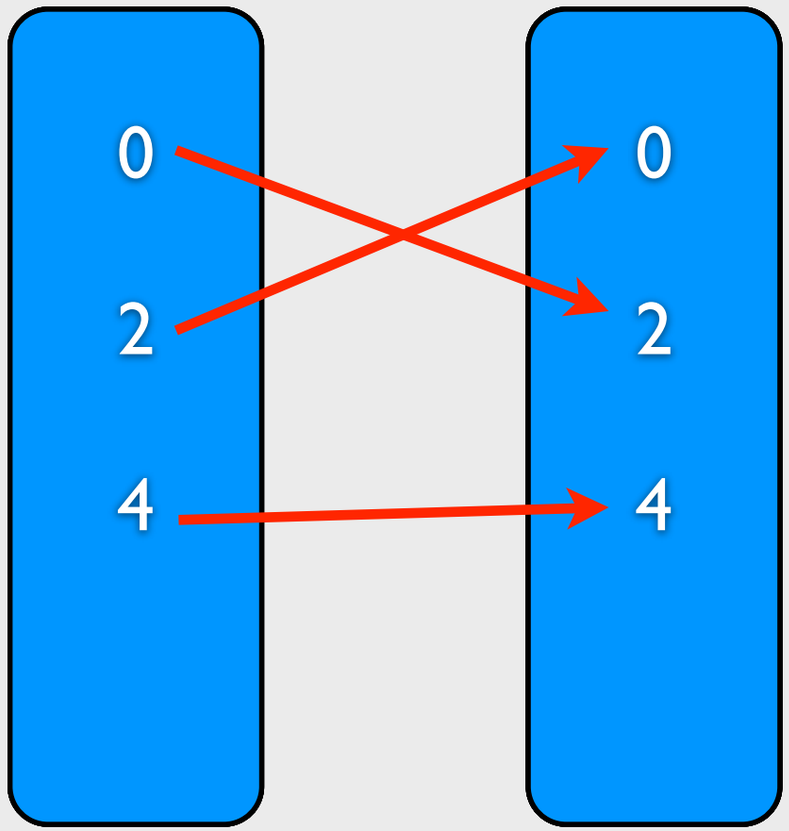
S' = {0, 2, 4}

stack

Think over 1to1 example

- What if we replace the queue as stack?

So, the result is the same!!



S' = {0, 2, 4}

stack

Think over 1to1 example

- What is the worst case? How many times will be executed?
  - the worst case: |S'| = 0
  - corresponding time complexity: O(n)

Vim Tips

- Vim Plugin Manager
  - Vundle
  - Useful links
    - http://blog.chh.tw/posts/vim-vundle/
    - http://blogger.gtwang.org/2014/04/vundle-vim-bundle-plugin-manager.html