Programming Practice

2018-11-15

Week 11

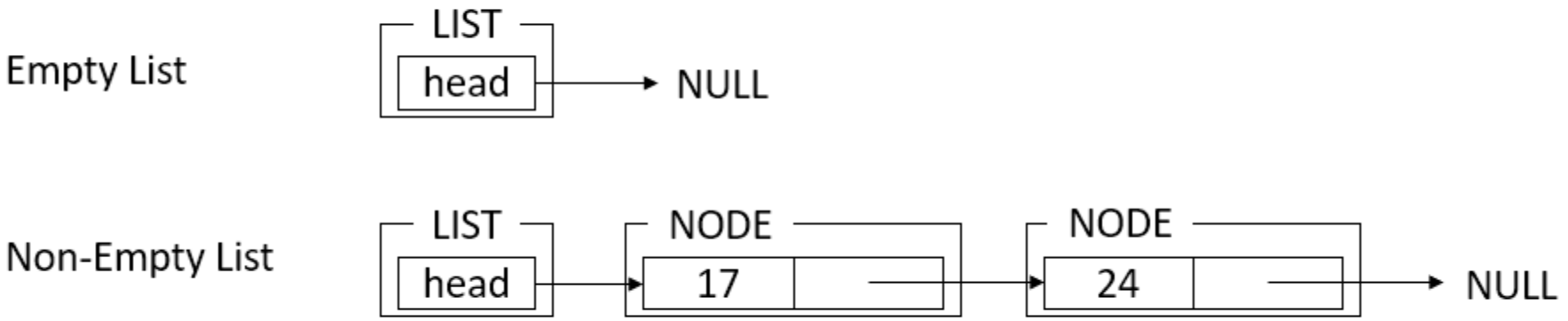
Notice

For this week

- Please use either Lab computer(Linux) or Martini server(Putty/SSH).
- Not on local Mac.

Practice Lecture

Empty vs Nonempty List



Dealing with list, you must care whether the list is empty or not!!

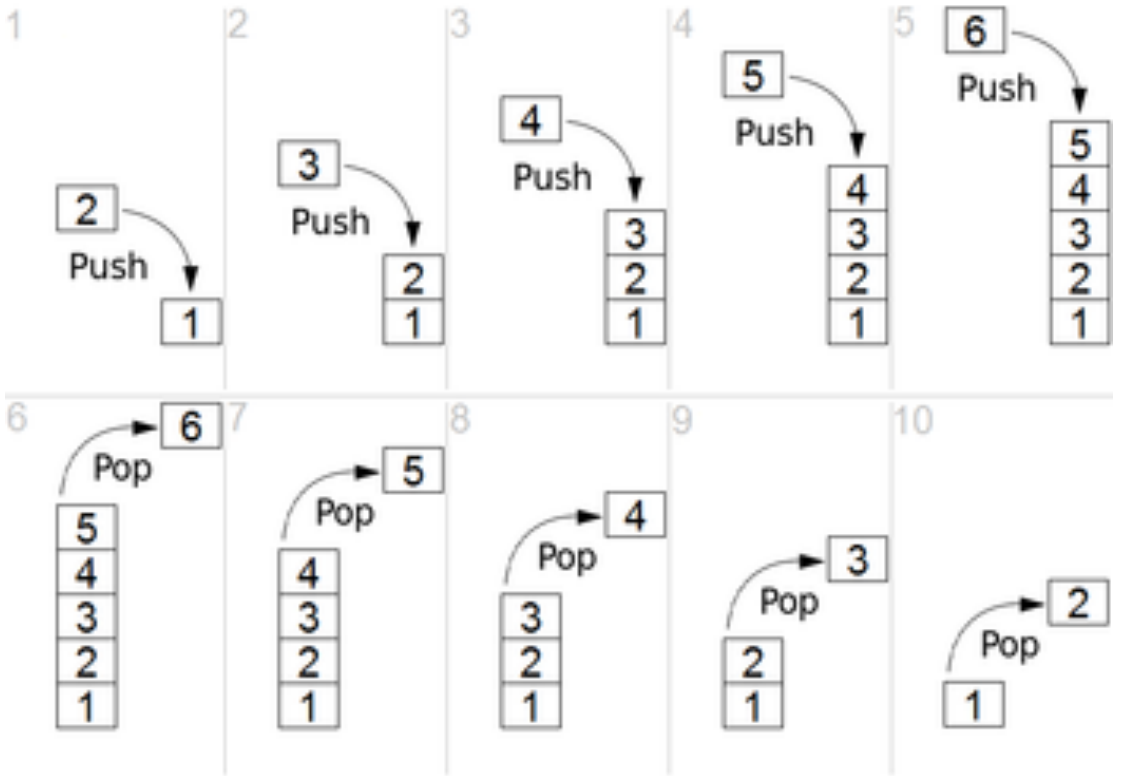
Stack

- This week, we'll practice creating Stack & basic functions.

- Two principal operations
  - Push element into stack
  - Pop element from stack

Stack

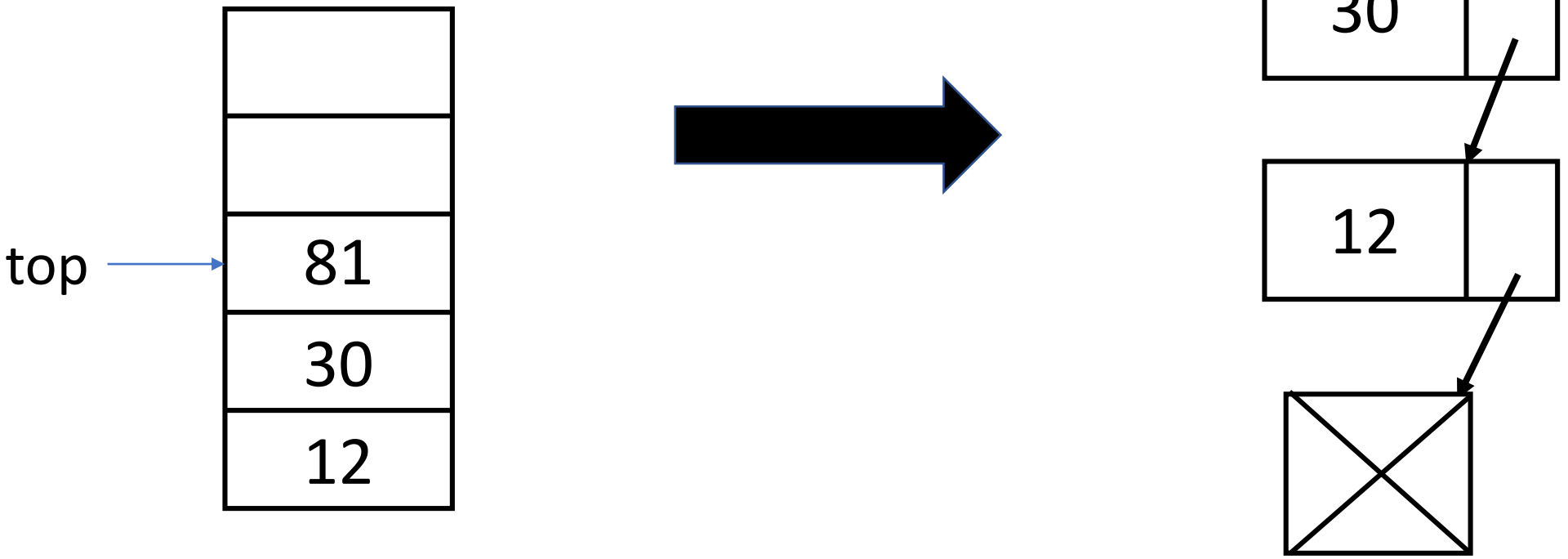
- LIFO(Last-In-First-Out)
  - Push at the top of the stack
  - Pop from the top of the stack



reference: https://en.wikipedia.org/wiki/Stack_(abstract_data_type)

Stack

- Construct stack based on linked list.



struct NODE & struct STACK

- ```
  typedef struct NODE{
      int value;
      NODE *next;
  }NODE;
  ```

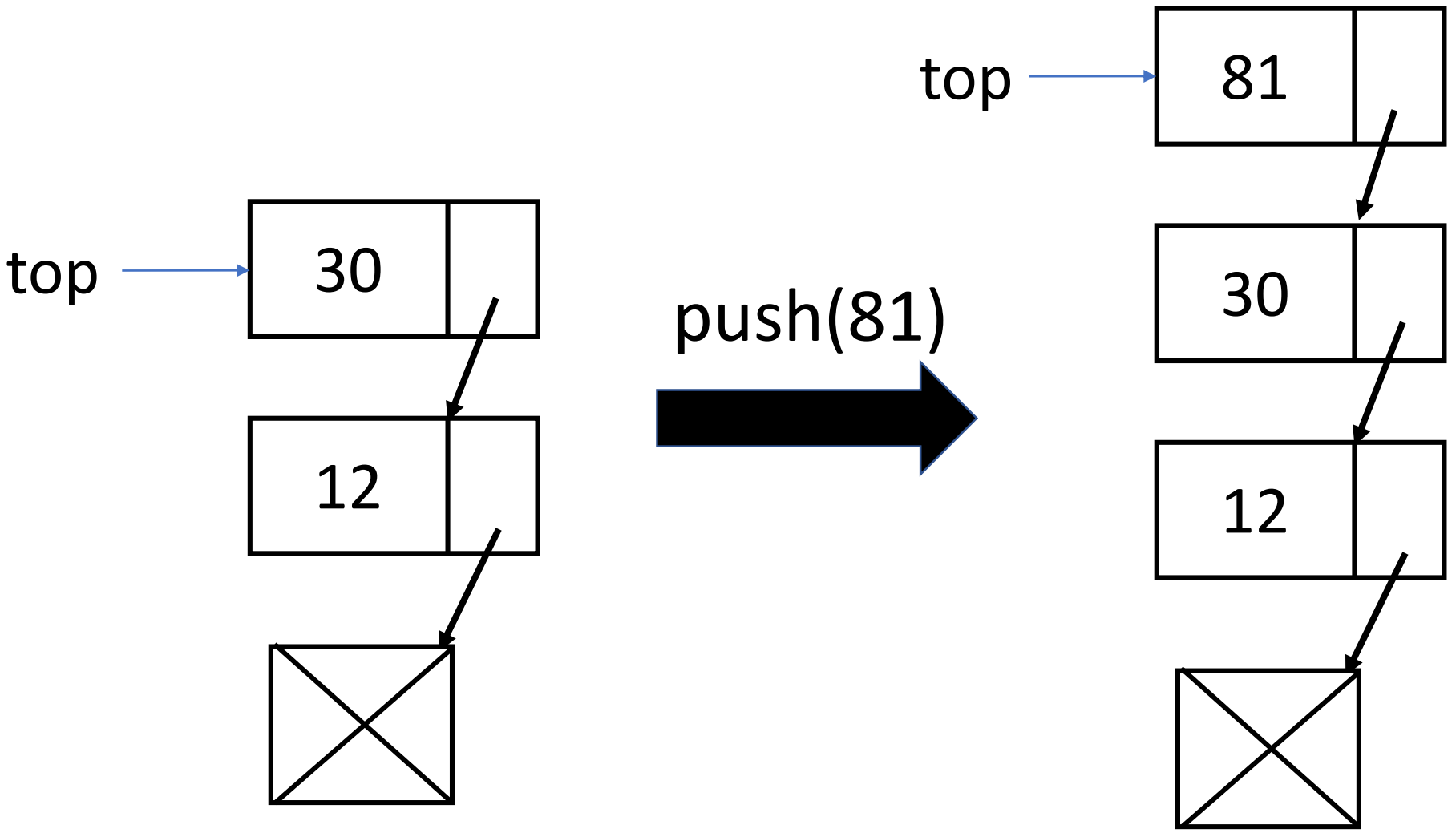
- ```
  typedef struct STACK{
      NODE *top;
  }STACK;
  ```

Stack

- Push Operation
  - Always insert at the top -> Don’t have to care about empty stack

  1) Generate new node with the pushed value.
  2) Make the node point the top of stack.
  3) Make top point the new node.

Stack



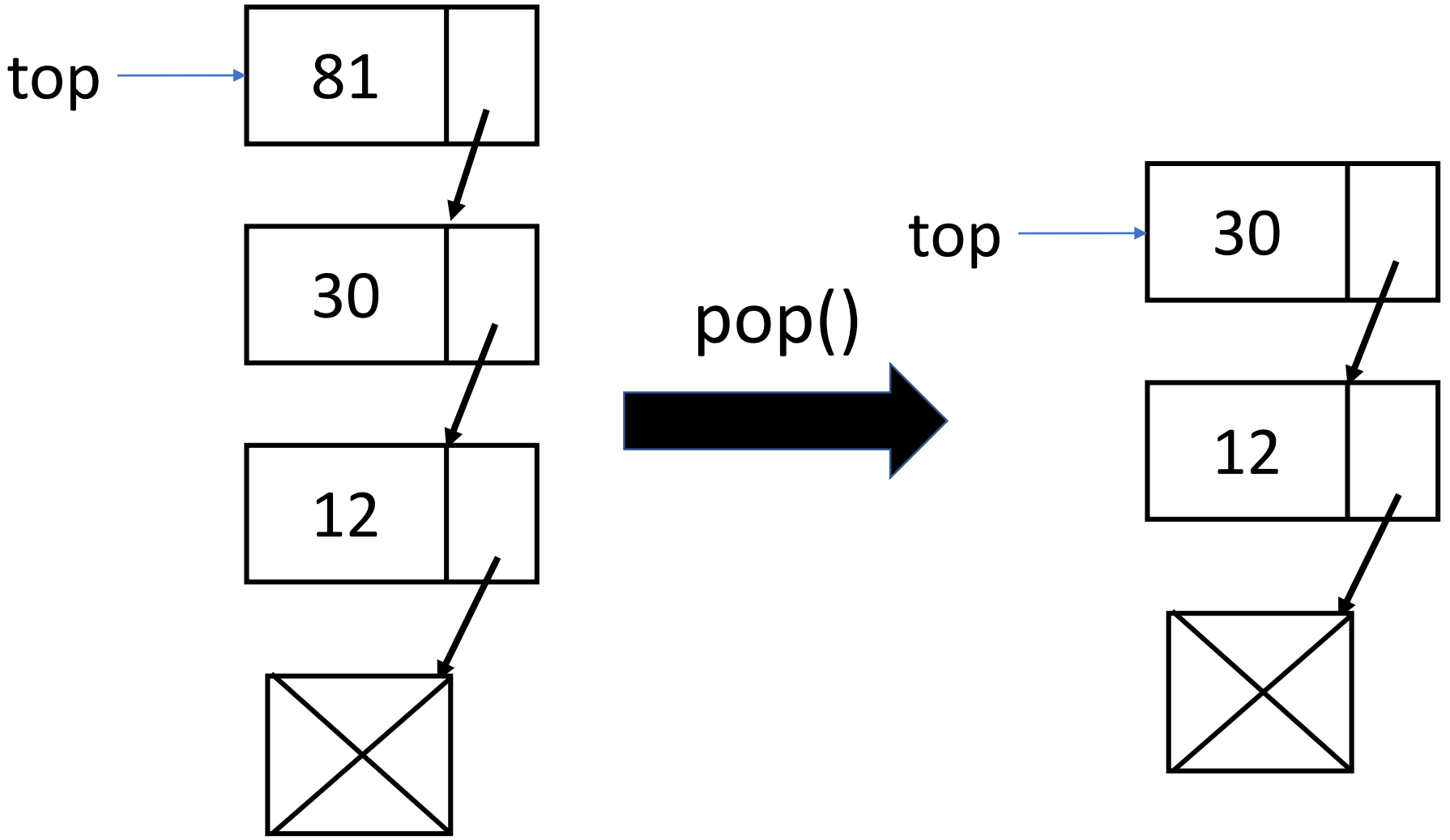
Stack

- Pop operation
  - Always delete at the top. Cannot delete if stack is empty. (Underflow)
  - Empty stack ⇔ top points NULL

    You should check whether the stack is empty or not before pop operation in general.

  1) Make top point the second element. It may be NULL.
  2) Remove the top element. Use free() function.
  3) Return the popped value.

Stack



Your job is to write

- push and pop functions

</> source code

```
1  void push(STACK *myStack, int value){
2      /* TODO */
3  }
4
5  int pop(STACK *myStack){
6      /* TODO */
7  }
```

Postfix Expression

- Three types of arithmetic expression
  - Prefix: * + 1 2 3
  - Infix: (1 + 2) * 3
  - Postfix: 1 2 + 3 *
- Only infix expression needs parentheses.

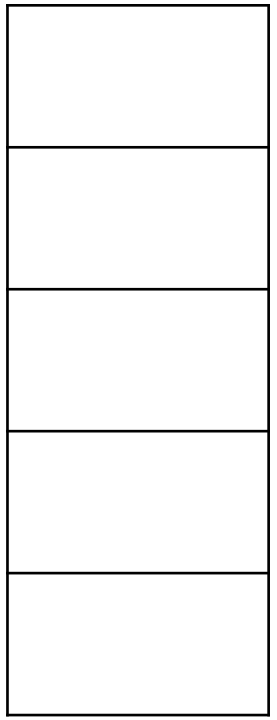
Postfix Expression

- How to calculate → Use stack
  - If you meet operand → push into stack
  - If you meet operator → pop twice, operate, push back into stack

Postfix Expression

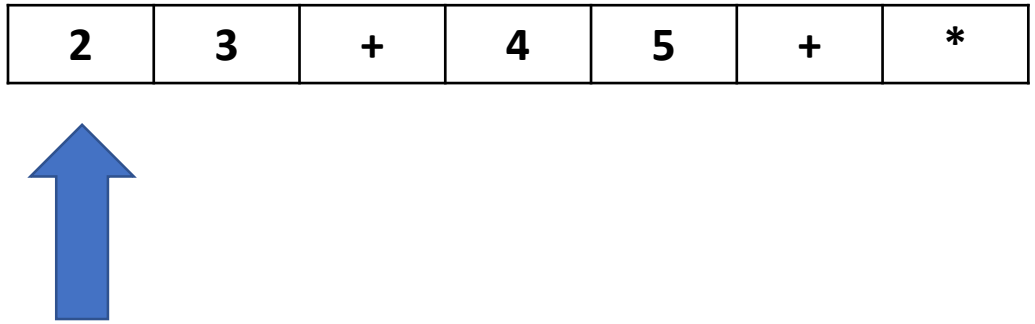
- Example: Calculate “2 3 + 4 5 + *” (Equal to (2+3)*(4+5))

2	3	+	4	5	+	*

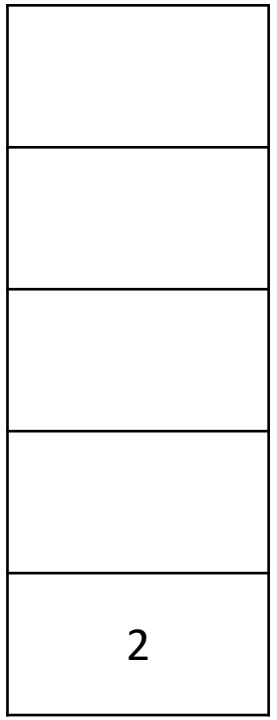


Operand Stack

Postfix Expression

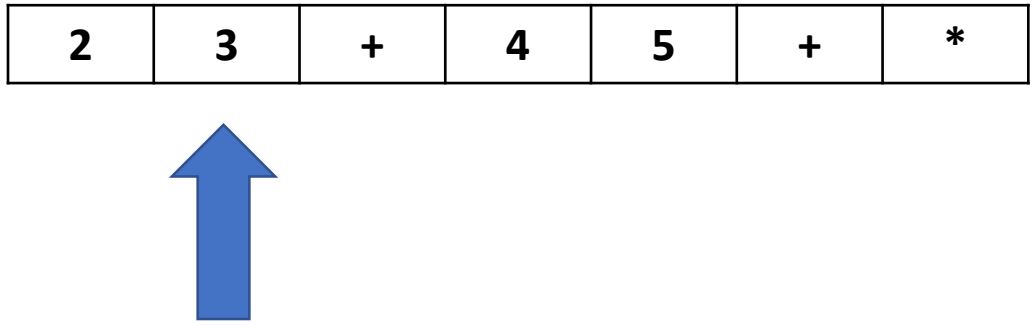


push 2

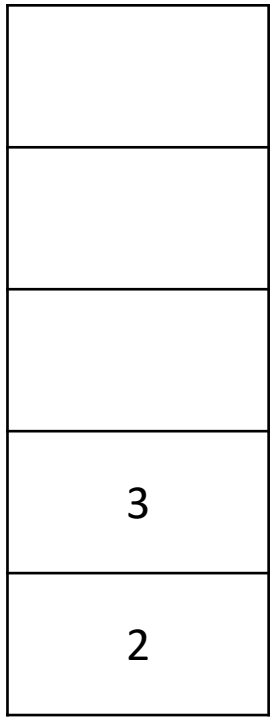


Operand Stack

Postfix Expression

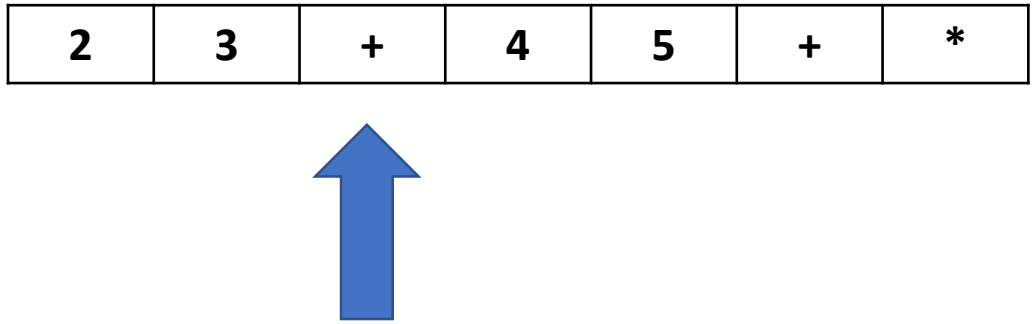


push 3

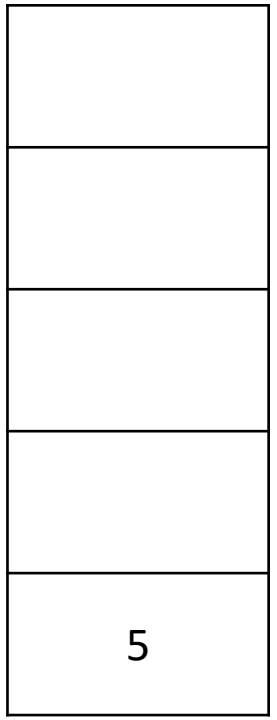


Operand Stack

Postfix Expression

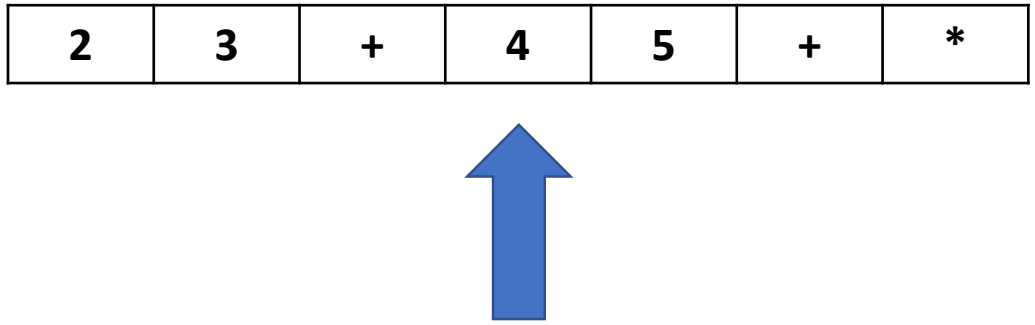


pop->3
pop->2
2+3 = 5
push 5

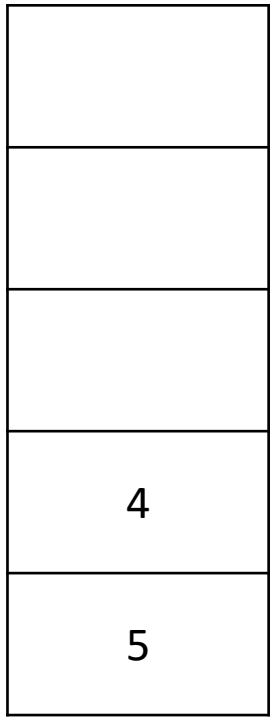


Operand Stack

Postfix Expression

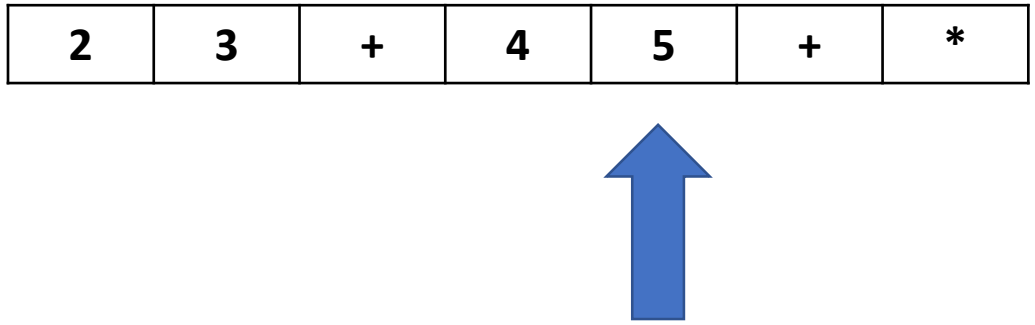


push 4

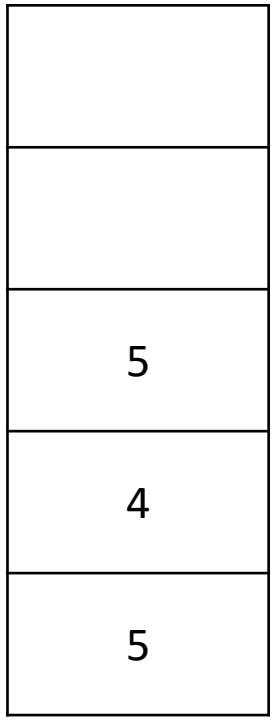


Operand Stack

Postfix Expression

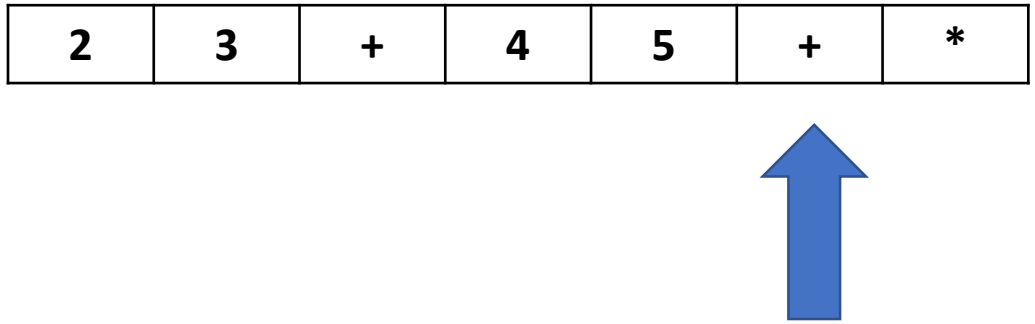


push 5

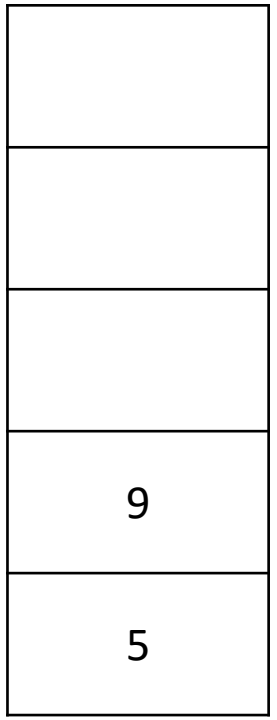


Operand Stack

Postfix Expression

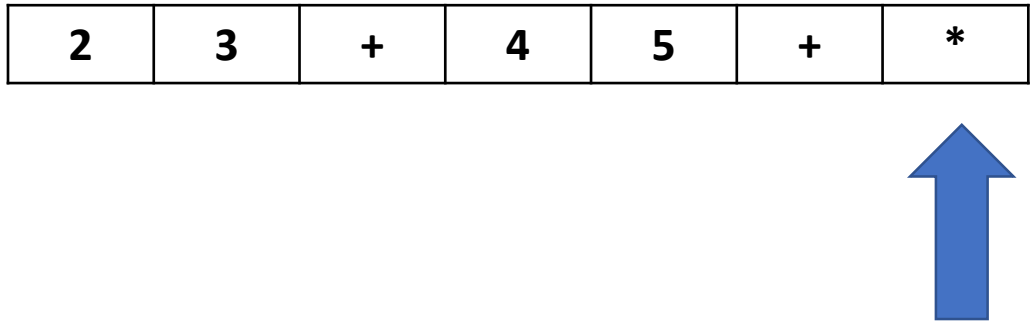


pop->5
pop->4
4+5 = 9
push 9

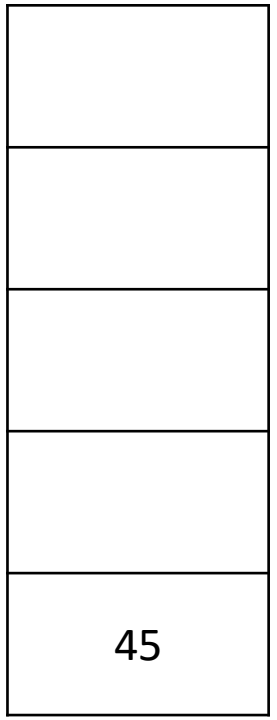


Operand Stack

Postfix Expression

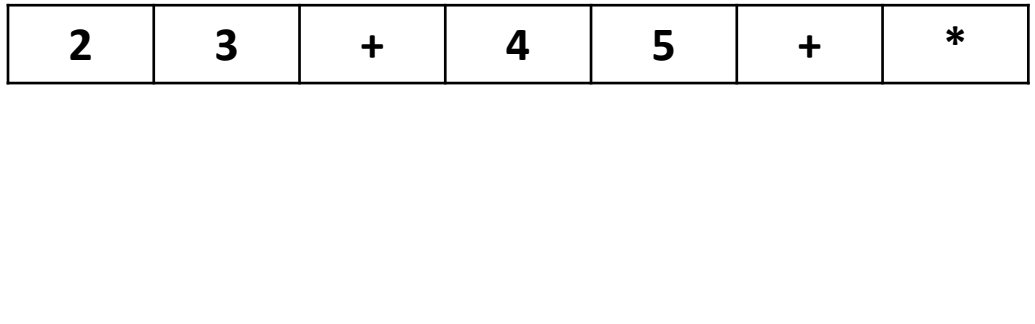


pop->9
pop->5
5*9 = 45
push 45



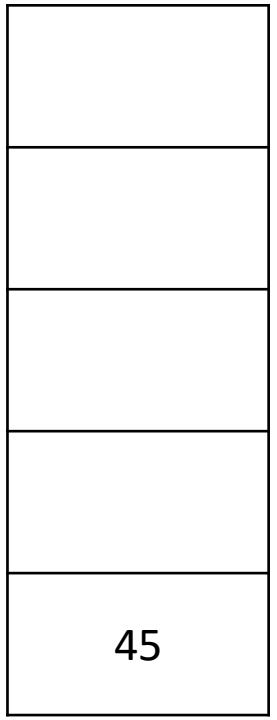
Operand Stack

Postfix Expression



DONE

result: 45



Operand Stack

Postfix Expression

- When is the expression valid?
  - No Underflow
  - Stack must have only one value at the end.

Homework Problems

1. Stack Push
2. Stack Pop
3. Postfix Expression

Problem. **1**

Stack Push

Submit your code file as **`push.c`**

Description

Push a *value* to the stack *myStack*.

(This means you have to create a new node when this function is called.)

Function prototype:

```
void push(STACK *myStack, int value);
```

Arguments:

`STACK *myStack` : Points to the stack to do the push operation to.

`int value` : The value for the new node to hold.

Problem. **2**

Stack Pop

Submit your code file as **`pop.c`**

Description

Pop the top value from the stack *myStack*.

Generally, you must check if stack is empty before pop operation. But in this problem you don’t have to care about it.

Function prototype:

```
int pop(STACK *myStack);
```

Arguments:

`STACK *myStack` : Points to the stack on which to do the pop operation.

Return:

Return popped value.

main()

- Will also be provided:

```c
int main(){
    STACK *myStack = (STACK *) malloc(sizeof(STACK));
    myStack->top = NULL;

    while(1){
        printf("Current: ");              // Print current state of stack
        print(myStack);

        char operation[5];           // Which type of operation
        scanf("%s", operation);

        if(strcmp(operation, "q") == 0){
            break;
        }
        else if(strcmp(operation, "push") == 0){
            int value;
            scanf("%d", &value);
            push(myStack, value);
        }
        else if(strcmp(operation, "pop") == 0){
            pop(myStack);
        }

        scanf("%*c");                // Flush '\n' at end of line
    }

    printf("Final: ");
    print(myStack);

    return 0;
}
```

Problem 1 & 2

- Submit code containing just the function (push/pop).
- Don't include `main()` in your submissions.

How this main() works

Continuously receives input, and does the requested operation.

There are 2 types of request. **`push, pop`**

Formats are as follows.

`push (value)`

`pop`

Example)

`push 7`	→	Push 7 -> [7]
`push 3`	→	Push 3 -> [3;7]
`pop`	→	Pop -> [7]
`push 5`	→	Push -> [5;7]
`q`	→	Finish with '**q**'

Skeleton Code

File Name	Description
main.c	Reads input & prints output. It will call your push, pop functions.
stack.h	Header file. Your functions should fit with the prototypes declared in this file.
push.c	TODO :: Implement **push** function.
pop.c	TODO :: Implement **pop** function.
TA_dir/	Directory that contains object files of TA's implementation.
Makefile	Compile macros are defined here.
testin.txt	Example test input. You may change test input if you want.

Makefile

- Build automation tool. Makes compile & running programs easier.

- You don’t have to understand it.
- It will help you to compile and test your program.

- You can simply open this file with **`vim Makefile`**
- If you are not familiar with this, Do NOT modify it.

How to use

- Four commands available

```
> make push
> make pop
```

Only Compilation

```
> make pushtest
> make poptest
```

Compile & Test

Problem. **3**

Postfix Expression

Description

Given postfix expression, calculate and print the result of expression.

Only valid expressions will be given.

Given operands are always positive integer.
NOTE: (integer) / (integer) = (integer). ex) 3/2 = 1

You don’t have to care about division by zero or integer overflow.

The character length of entire expression is less than or equal to 10^6.

Input

Valid postfix expression is given.

Output

Print out the result of calculation.

Sample

```
[Input]
2 3 + 4 5 + *
[Output]
45


[Input]
155 3 2 1 + * /
[Output]
17
```

Integer Parsing

- How to change string to integer?
- char str[5] = “1023”;

Integer Parsing

- First, do yourself

```
ret = 0
for i = start ~ end{
        ret = ret * 10
        ret = ret + (str[i]-’0’)
}
```

Integer Parsing

- Use atoi function
- ret = atoi(str)
- available only if str is valid (cannot parse -3$2, 3382+2, and so on)