Data Structures and Algorithms
(ES221)

Stacks

Dr. Zubair Ahmad

Stacks

Known as LIFO Lists

Last in, First Out

Basic operations

Push

Pop

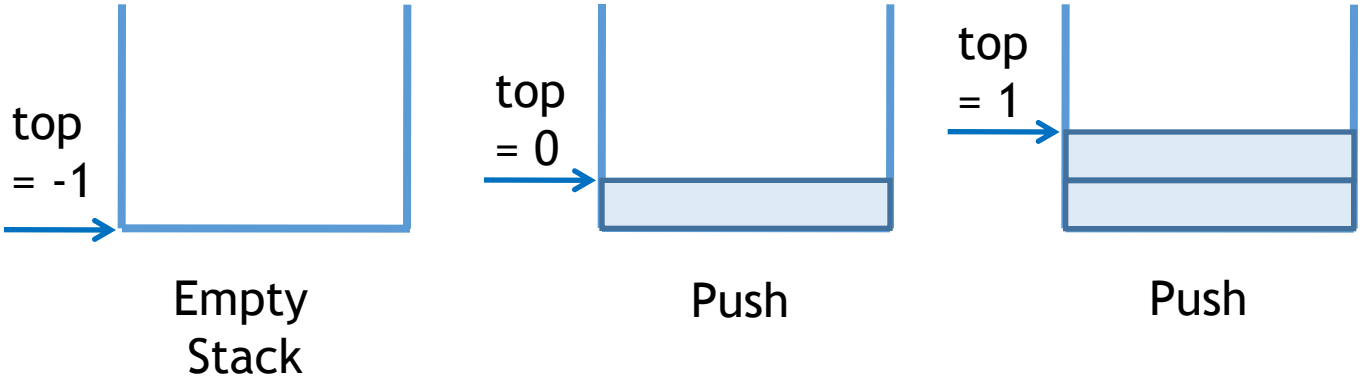
Top

- Implementation
  - Array implementation
  - Linked list implementation

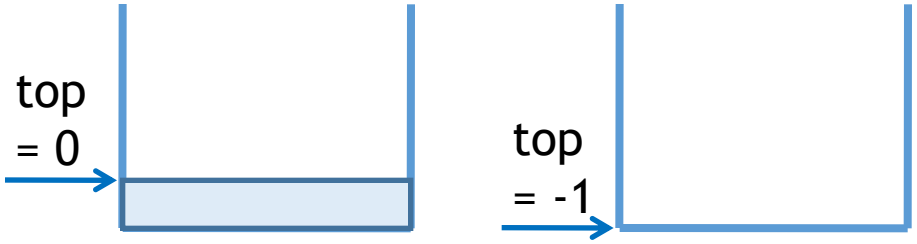
Stack operations

- Push
  - Add an element/data at the top of the stack
- Pop
  - Remove and element/data from the top of the stack
- Top
  - Read the contents of the top element in the stack

Stack operations



Empty Stack | Push | Push



Pop | Pop

Stack ADT: Array Implmentation

- Array implementation
  - static array implementation
  - dynamic array implementation
- dynamic linked list implementation

Stack ADT:
static Array Implementation

- Maximum Size of the stack is fixed
- When the stack is empty
  - top = -1
- Push operation
  - top = top+1
  - Stack[top] = newData
- Pop operation
  - topData=Stack[top];
  - top =top-1;
- Complexity: O(1)

Stack ADT:
static Array Implementation

- Stack Declaration

```
const int N = 5;
int stack[N];
int top;
```

- Read Operation

```
int ReadTop(){
  if(top == -1){
      cout<<"\nError: Stack is Empty\n";
      return -1;
      }
  else{
      cout<<"\nThe top elmenet is\n"<<stack[top];
      return stack[top];
     }
}
```

Stack ADT:
static Array Implementation

- Push Operation

```
void Push(int newData){
   if(top == N-1)
        cout<<"\nError: Stack is Full\n";
   else{
        top=top+1;
        stack[top] = newData;
        cout<<"\npushing "<< newData <<" at the    top\n";
        }
    }
```

Stack ADT:
static Array Implementation

- Pop Operation

```
int Pop(){
    int topData;
    if(top == -1){
        cout<<"\nError: Stack is Empty\n";
        return -1;
        }
    else {
        topData = stack[top];
        cout<<"\nremoving "<< stack[top] <<" from the top\n";
        top = top-1;
        return topData;
     }
}
```

Stack ADT:
static Array Implementation

```
void main(){
        ReadTop();
        Pop();
        Push(1);
        Push(2);
        Push(3);
        Push(4);
        Push(5);
        Push(6);
        Pop();
        ReadTop();
        Pop();
        ReadTop();
        getch();
        }
```

Stack ADT:
Linked List Implementation

- Stack Declaration

```
//const int N=5;
  struct stack{
      int data;
      stack*next;
      };
  stack *topPtr = NULL;
```

- Read Operation

```
int ReadTop() {
    if(topPtr == NULL){
        cout<<"\nError: Stack is Empty\n";
        return -1;
        }
    else{
         cout<<"\nThe top elmenet is\n  " <<topPtr->data;
         return topPtr->data;
         }
    }
```

Stack ADT:
Linked List Implementation

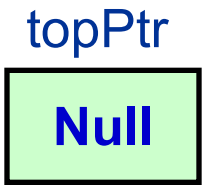
- Push Operation

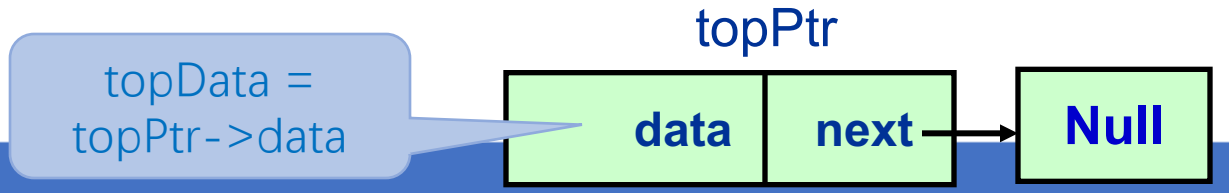
```
void Push(int newData){
//        if(top==N-1)
//          cout<<"\nError: Stack is Full\n";
//  else{
    stack *ptr = new stack;
    ptr->next = topPtr;
    topPtr=ptr;
    topPtr->data = newData;
    cout<<"\npushing "<< newData <<" at the top\n";
    // }
}
```

Stack ADT:
Linked List Implementation

- Pop Operation

```
int Pop(){
    int topData;
    if(topPtr == NULL){
        cout<<"\nError: Stack is Empty\n";
        return -1;
        }
    else {
        topData = topPtr->data;
        cout<<"\nremoving "<< topPtr->data <<" from the top\n";
        topPtr = topPtr->next;
        return topData;
        }
    }
```





Stacks Application: Balancing Symbols

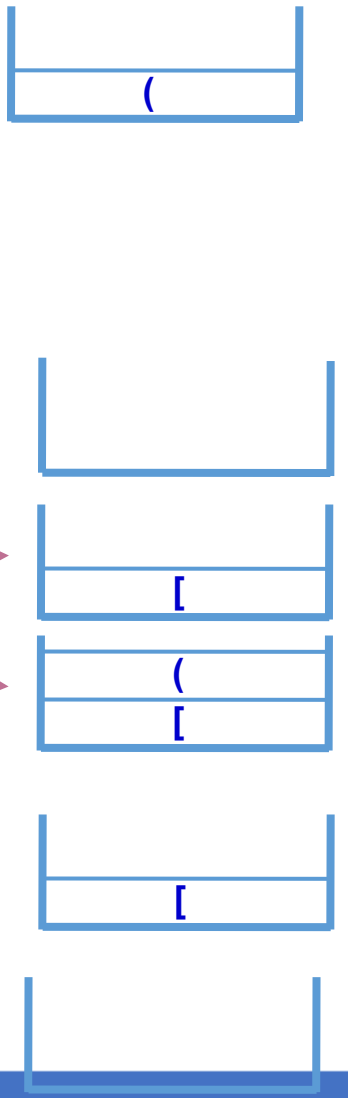
- A lack of a certain symbol (such as a mssing brace or a comment starter) will cause the compiler to spill out a hundred lines of diagnostics without identifying the real error.
- Solution?
  - Check whether everything is balanced?
  - Every right brace, bracket, parenthesis and etc must correspond to its left counter part

Stack used for Balancing Symbols

- Make an empty ***STACK***
- Read characters until the end of file
  - If the character is an opening symbol
    - Push it onto the ***STACK***
  - If the character is a closing symbol
    - If the ***STACK*** is empty
      - Report error
    - Else
      - Pop the ***STACK***
      - If the symbol popped is not the corresponding symbol
        - Report error
- If the ***STACK*** is not empty in the end
  - Report error

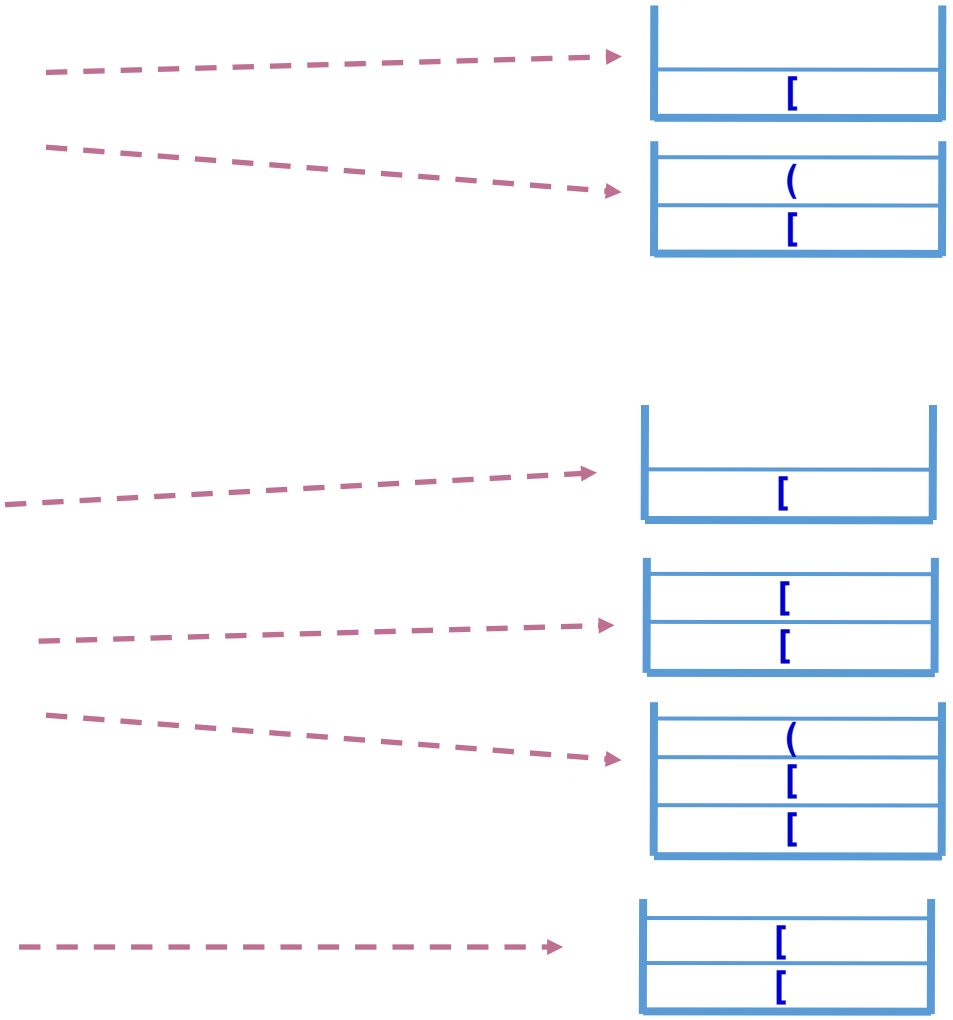
EXAMPLE: STACKS USED FOR BALANCING SYMBOLS

- **(a + b * c) / [(a – b) ^ c]**
- Read → **(** → Push **(**
- Read → **a**
- Read → **+**
- Read → **b**
- Read → *****
- Read → **c**
- Read → **)** → Pop
- Read → **/**
- Read → **[** → Push **[**
- Read → **(** → Push **(**
- Read → **a**
- Read → **-**
- Read → **b**
- Read → **)** → Pop
- Read → **^**
- Read → c
- Read →] → Pop



EXAMPLE: STACKS USED FOR BALANCING SYMBOLS

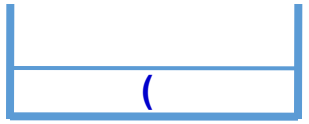
- **[(a - b / c) / [(a – b) * c**
- Read → **[** → Push **[**
- Read → **(** → Push **(**
- Read → **a**
- Read → **-**
- Read → **b**
- Read → **/**
- Read → **c**
- Read → **)** → Pop
- Read → **/**
- Read → **[** → Push **[**
- Read → **(** → Push **(**
- Read → **a**
- Read → **-**
- Read → **b**
- Read → **)** → Pop
- Read → *****
- Read → c



Stack Not Empty Error!

EXAMPLE: STACKS USED FOR BALANCING SYMBOLS

- **(a - b * c)] / [[(a – b) ^ c]**
- Read → **(** → Push **(**
- Read → **a**
- Read → **-**
- Read → **b**
- Read → *****
- Read → **c**
- Read → **)** → Pop
- Read → **]** → Pop



Stack Empty
No matching opening symbol
Error!

Postfix and Infix expression

Infix Expression

(5 + 3) * 8

Parentheses determine precedence.

Equivalent Postfix Expression

5 3 + 8 *

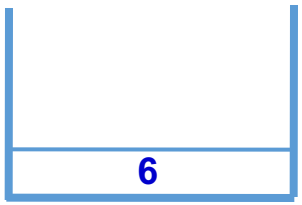
No parentheses needed, evaluated left to right using a stack.

STACK USED TO EVALUATE A POSTFIX EXPRESSION

1. Make an empty ***STACK***
2. Add a right parenthesis ‘**)**’ at the end of expression ***P***
3. Scan ***P*** from left to right and **repeat** the following steps for each element of ***P*** until the sentinel ‘**)**’ is encountered
   - If an operand is encountered
     - Push the operand into the ***STACK***
   - Else if an operator ® is encountered
     (a) Remove the top two elements ***A*** and ***B*** (***A*** **= top**, and **B = next to top**)of the ***STACK***
     (b) Evaluate ***B*** ® ***A***
     (c) Push the result of (b) into ***STACK***
   - End if
- End **repeat**
- Set the ***Result*** equal to the top element of the ***STACK***
- Exit

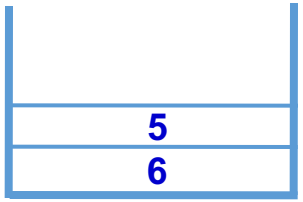
EXAMPLE: STACK USED TO EVALUATE A POSTFIX EXPRESSION

- 6523+8*+3+* → 6523+8*+3+*)



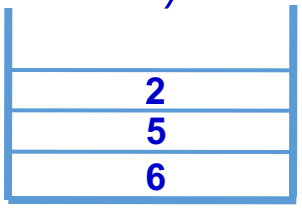
Read :→ **6**

→ Push **6**



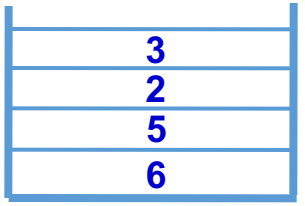
Read :→ **5**

→ Push **5**



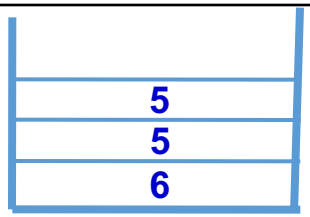
Read :→ **2**

→ Push **2**



Read :→ **3**

→ Push **3**

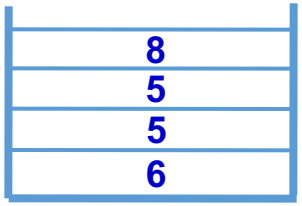


Read :→ **+**

→ Pop **3**, Pop **2**

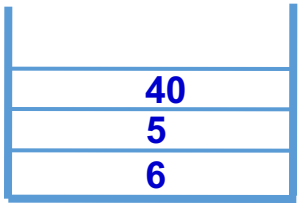
2 + 3 = 5

Push **5**



Read :→ **8**

→ Push **8**

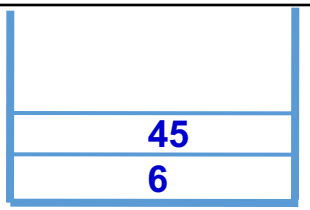


Read :→ *

→ Pop **8**, Pop **5**

5 * 8 = 40

Push **40**

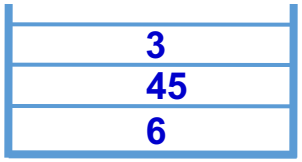


Read :→ **+**

→ Pop **40**, Pop **5**

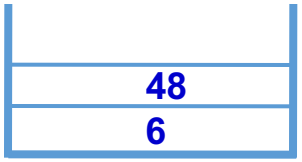
5 + 40 = 45

Push **45**



Read :→ **3**

→ Push **3**

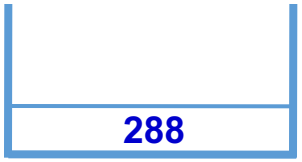


Read :→ **+**

→ Pop **3**, Pop **45**

45 + 3 = 48

Push **48**

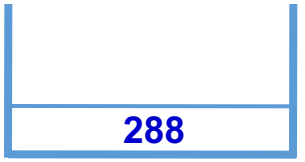


Read :→ *

→ Pop **48**, Pop **6**

6 * 48 = 288

Push **288**



Read :→ **)**

Result = **STACK[top]**

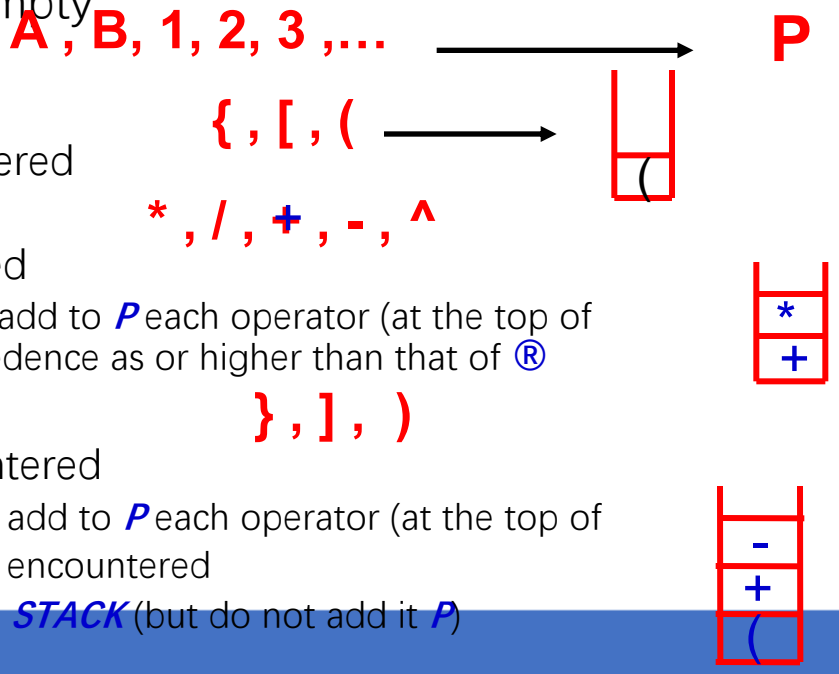
= 288

STACK USED TO TRANSFORM AN INFIX EXPRESSION INTO POSTFIX EXPRESSION

Shunting-yard algorithm

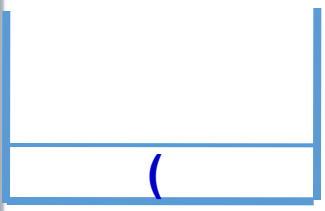
Suppose the infix expression ***Q*** has to be converted into the postfix expression ***P***

1. Make an empty ***STACK***
2. Push the left parenthesis '(' into the ***STACK*** and add the right parenthesis ')' at the end of expression ***Q***
3. Scan ***Q*** from left to right and **repeat** the following steps for each element of ***Q*** until the ***STACK*** is empty
   - If an operand is encountered
     - Add it to ***P***
   - Else if a left parenthesis is encountered
     - Push it into the ***STACK***
   - Else if an operator ® is encountered
     (a) Repeatedly pop from ***STACK*** and add to ***P*** each operator (at the top of ***STACK***) which has the same precedence as or higher than that of ®
     (b) Push ® into the ***STACK***
   - Else if a right parenthesis is encountered
     (a) Repeatedly pop from ***STACK*** and add to ***P*** each operator (at the top of ***STACK***) until a left parenthesis is encountered
     (b) Remove the left parenthesis from ***STACK*** (but do not add it ***P***)
   - End if



STACK EXAMPLE: INFIX EXPRESSION INTO POSTFIX EXPRESSION

6*[5+ (2+3)*8+3] → (6 * [5 + (2 + 3) * 8 + 3])



Read :→ (

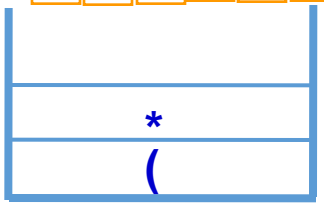
→ Push (

OUTPUT:

Read :→ **6**

Add **6** to ***P***

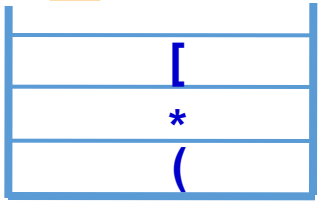
OUTPUT: **6**



Read :→ *

→ Push *

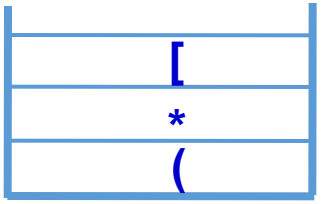
OUTPUT: **6**



Read :→ [

→ Push [

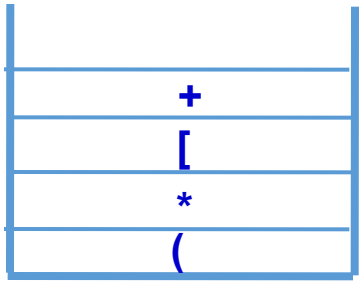
OUTPUT: **6**



Read :→ **5**

Add **5** to ***P***

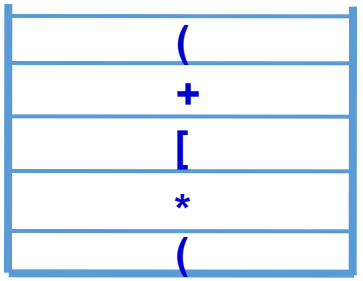
OUTPUT: **65**



Read :→ **+**

→ Push **+**

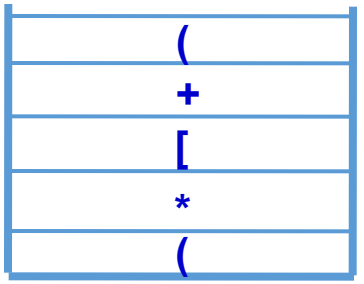
OUTPUT: **65**



Read :→ **(**

→ Push **(**

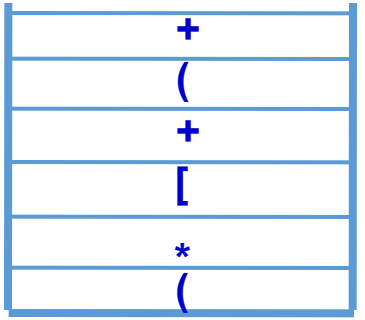
OUTPUT: **65**



Read :→ **2**

Add **2** to ***P***

OUTPUT: **652**



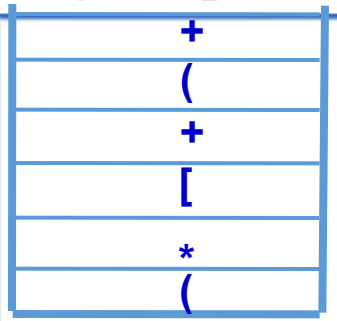
Read :→ **+**

→ Push **+**

OUTPUT: **652**

STACK INFIX TO POSTFIX EXPRESSION

(6 * [5 + (2 + 3) * 8 + 3]) → 6523+8*+3+*

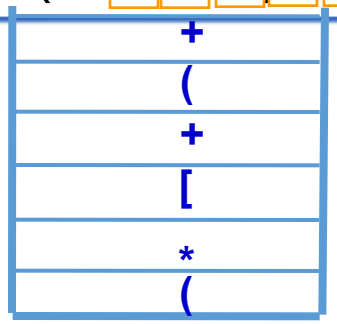


Read :→ **+**

→ Push **+**

OUTPUT:

652

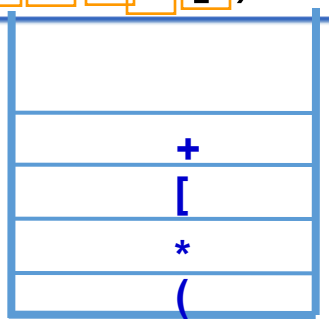


Read :→ **3**

Add **3** to ***P***

OUTPUT:

6523



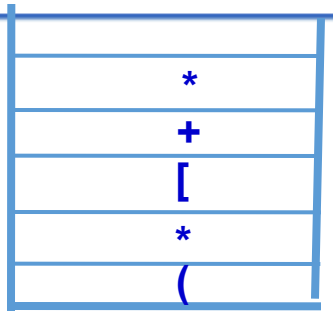
Read :→ **)**

→ Pop **+**, Add **+** to ***P***

Pop **(**

OUTPUT:

6523+

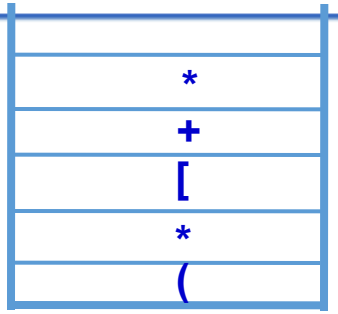


Read :→ ***

→ Push ***

OUTPUT:

6523+

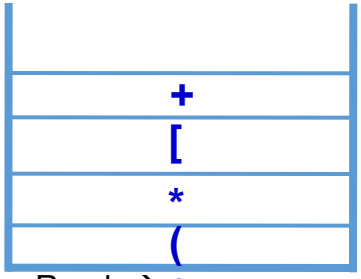


Read :→ **8**

Add **8** to ***P***

OUTPUT:

6523+8



Read :→ **+**

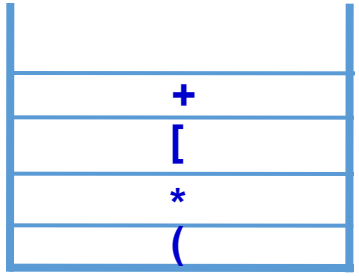
→ Pop ***, Add *** to ***P***

→ Pop **+**, Add **+** to ***P***

Push **+**

OUTPUT:

6523+8*+

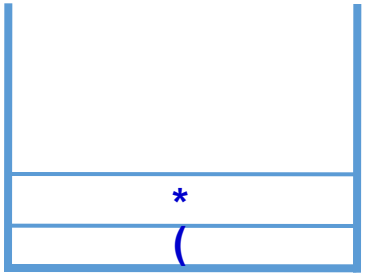


Read :→ **3**

Add **3** to ***P***

OUTPUT:

6523+8*+3



Read :→]

→ Pop **+**, Add **+** to ***P***

Pop **[**

OUTPUT:

6523+8*+3+

Read :→)

→ Pop ***, Add *** to ***P***

Pop **(**

OUTPUT:

6523+8*+3+*

Questions?

zahmaad.github.io