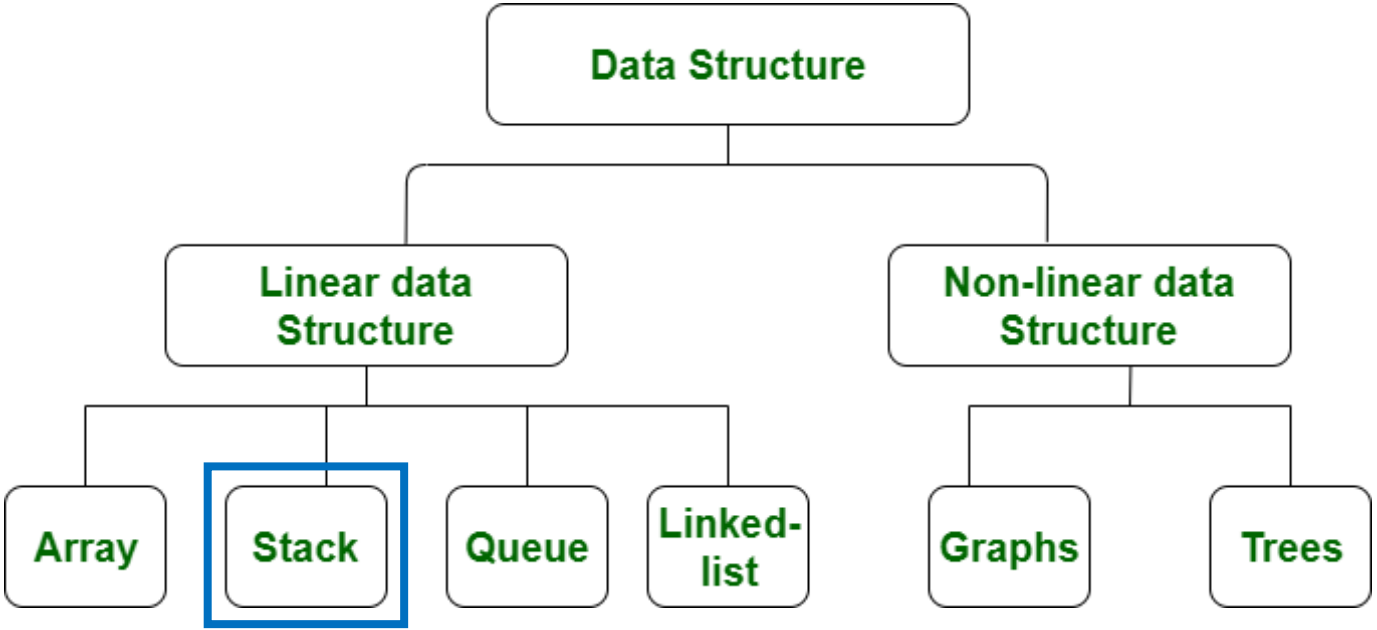
STACK

Teaching Team
Algorithm and Data Structure
Information Technology Dept.


Data Structure
Linear data Structure
Non-linear data Structure
Array
Stack
Queue
Linked-list
Graphs
Trees

Linear Data Structure Definition

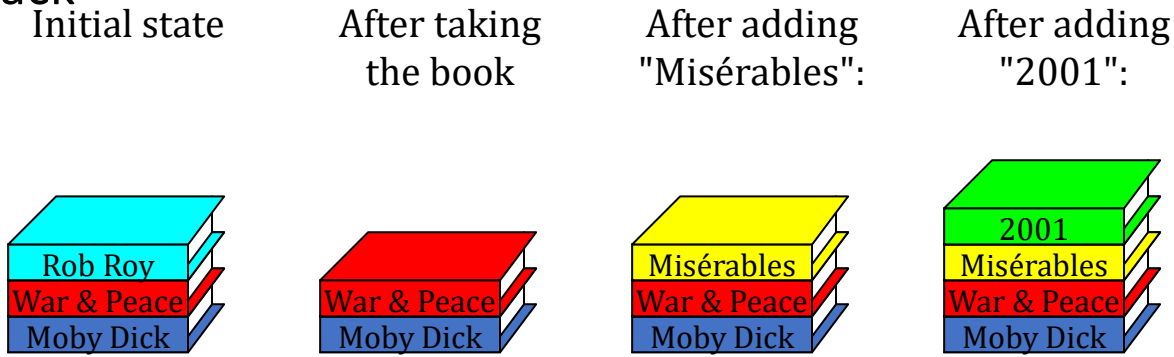
- All data arranged sequentially or linearly. Each element is managed to each other by the element before and after it.
- All data can be traversed in one run
- Each element is accessed or placed at a contiguous memory address

Stack Definition

- Stack is dynamic data structure concept that follows LIFO (Last In First Out) principle.
- The last item to be inserted into a stack is the first item that will go out from stack
- The first item inserted into a stack, then it will be the last item that will go out from stack
- Example: there is a stack of CD (compact disc). The CD at the top of the stack is the first item to be moved when you require a CD from that stack.

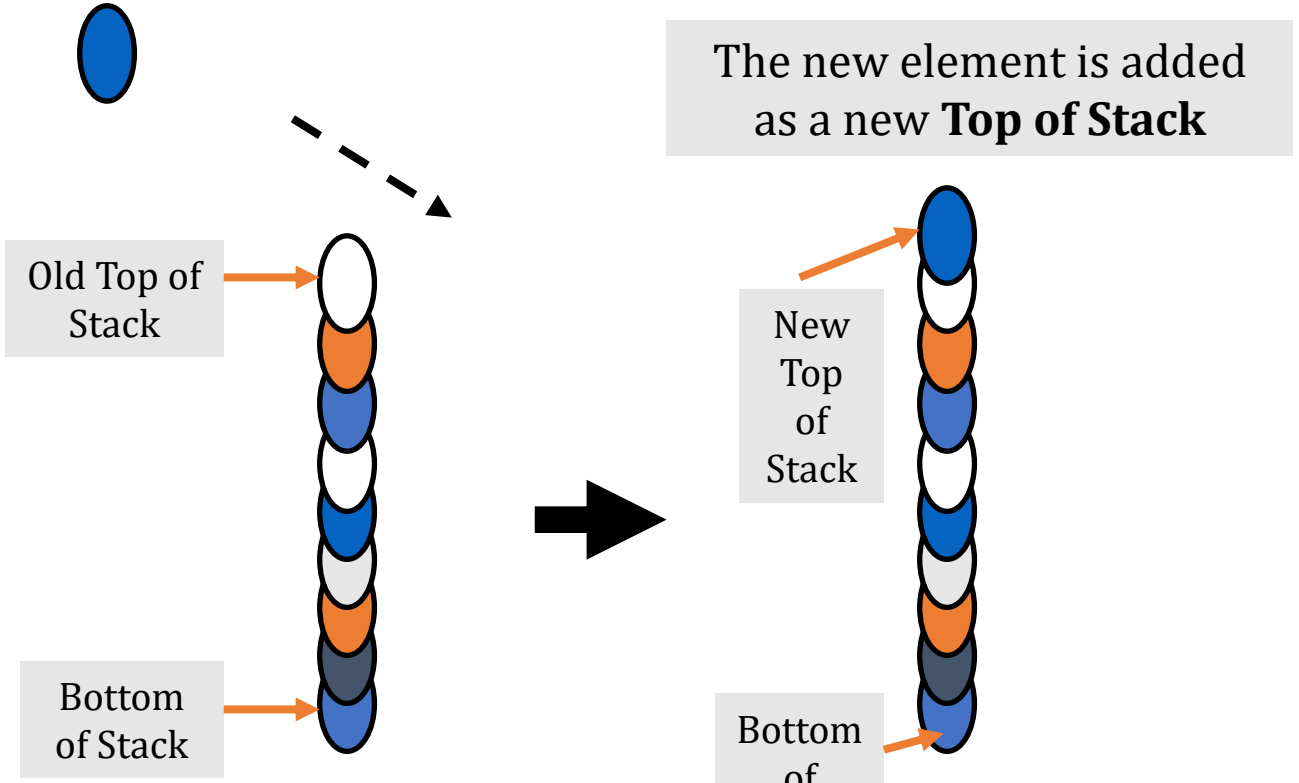
Stack Concept

- An arrangement of data collections where data can be added and deleted. This process is always done at the end of the data which is at the top position, which is called the **top of stack**
- The object that **last entered** the stack will be the **first object to exit** the stack



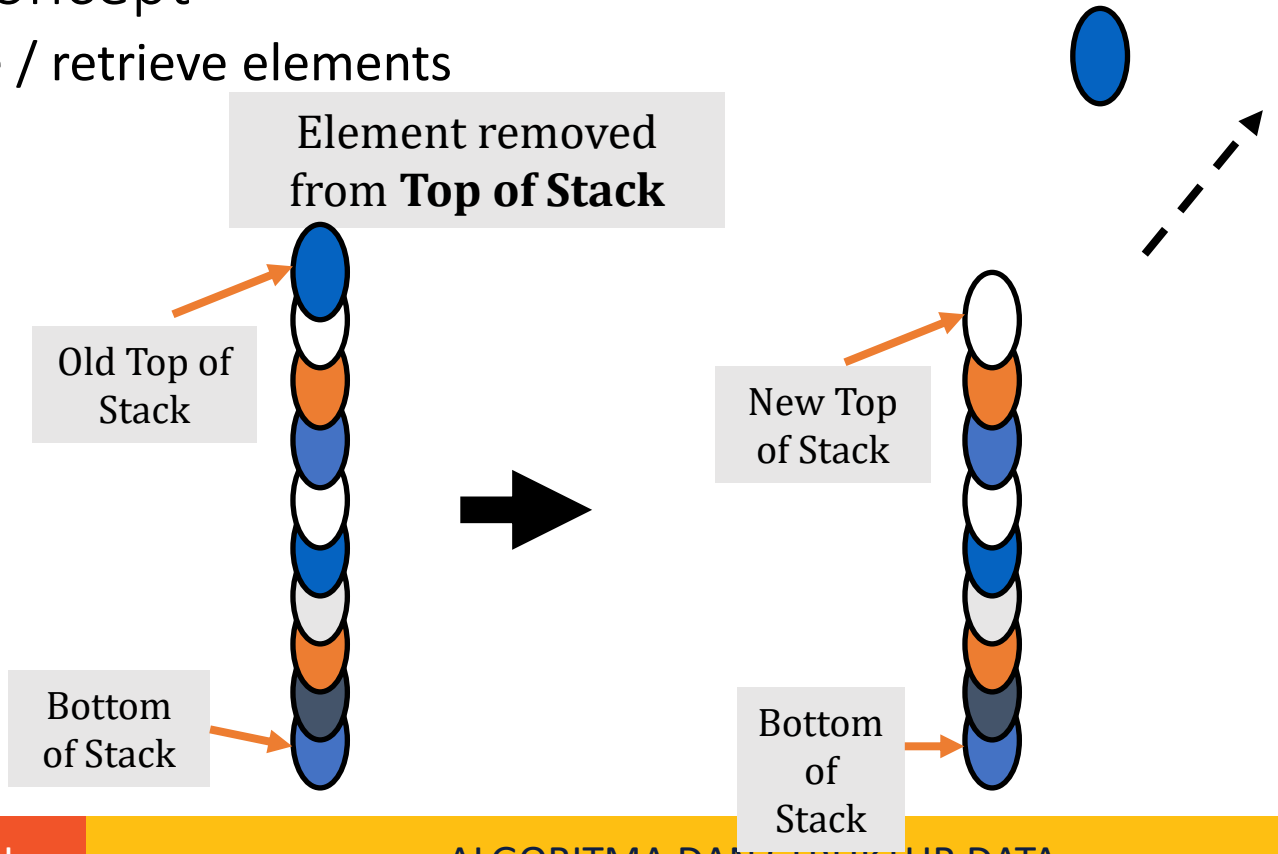
Stack Concept

- Add elements



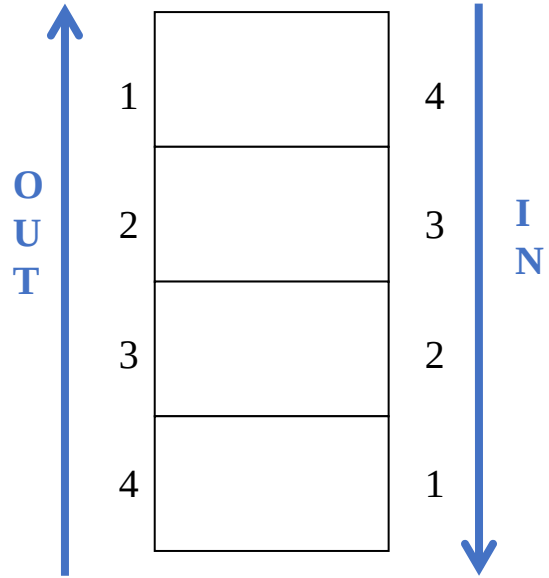
Stack Concept

- Remove / retrieve elements



Stack Operation

- **IsFull**: check whether the stack is full
- **IsEmpty**: check whether the stack is empty
- **Push**: add elements to the stack on the top stack
- **Pop**: retrieve elements on the stack on the top stack
- **Peek**: check the top element
- **Print**: display all elements in the stack
- **Clear**: empty the stack



Stack Declaration

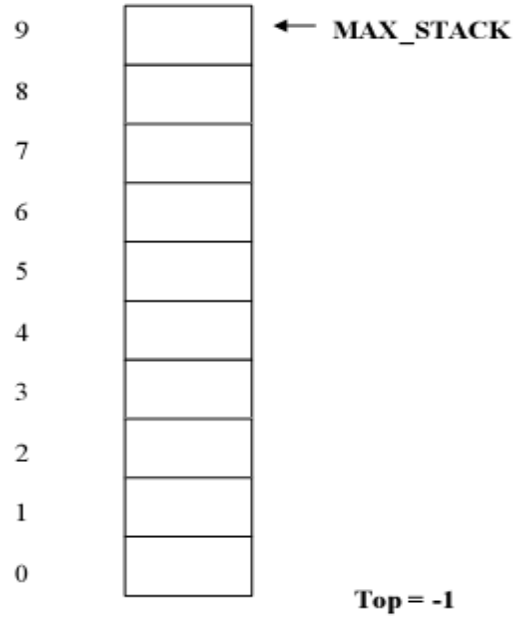
- The first process to do is to declare or prepare a place for the stack
- Steps:
  - Class declaration
  - Array declaration
  - Declaration of top pointer

```
public class Stack {
    int data[];
    int size;
    int top;
}
```

Stack initialization

- Initially fill **top** with -1 because the array starts at 0, which means that the data stack is currently **EMPTY**
- **Top** is a marker variable in the stack that shows the top element of the current data stack. Top will save the index of array where the top data is located.
- The **Top of Stack** will always move until it reaches the **MAX of STACK** (size) which causes a **FULL** stack!

Stack initialization



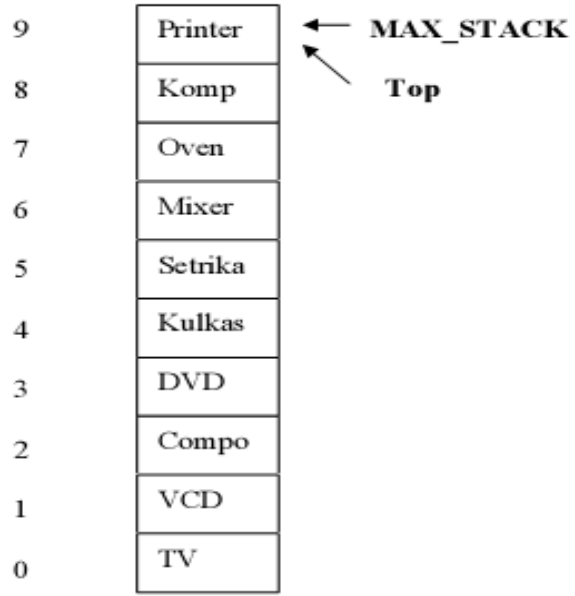

```
public Stack(int size) {
    this.size = size;
    data = new int[size];
    top = -1;
}
```

- Illustration of Stack during initialization

IsFull function

- To check whether the stack is **full** by checking the **top** of stack
- If the top of stack is the same as **size - 1**, then it's **full**
- If the top of stack is still smaller than **size - 1**, then it's not full

IsFull function




```
public boolean IsFull() {
    if (top == size - 1) {
        return true;
    } else {
        return false;
    }
}
```

- Illustration of stack in Full condition

IsEmpty Function

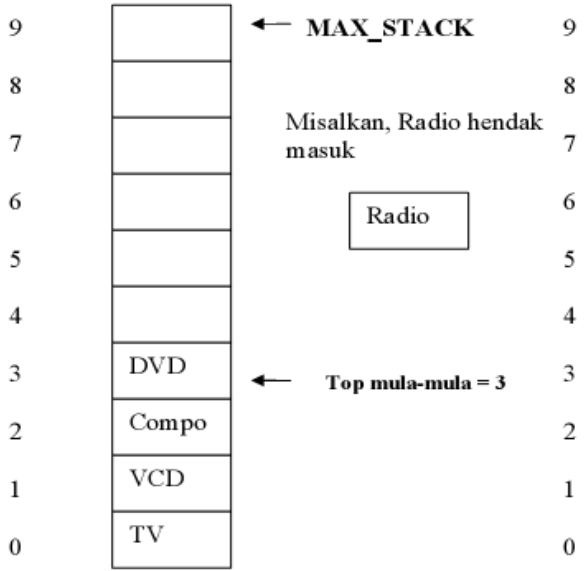
- To check whether the Stack is still **empty**
- By checking the **top** of stack, if it's still -1 then it means the data stack is still empty!

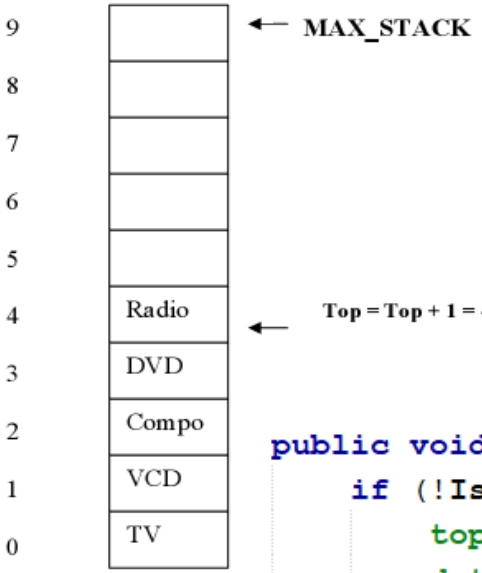
```
public boolean IsEmpty() {
    if (top == -1) {
        return true;
    } else {
        return false;
    }

}
```

Push Function

- To insert elements into the data stack. The new data will always be entered in the **top** of the stack (which is designated by the top of the stack)
- If the **data is not full**,
  - Add one (increment) **top** of stack value first every time there is an addition to the stack data array.
  - Fill new data into the stack based on the index of **top** of stack that has been previously incremented.
- If not, output "Full"

Push Function



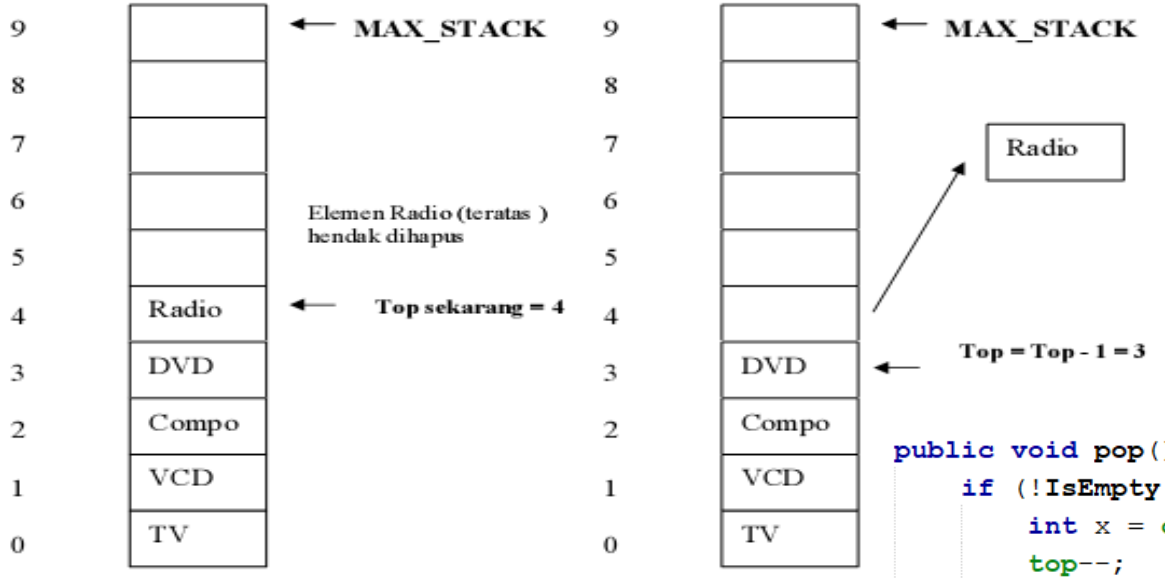



```
public void push(int dt) {
    if (!IsFull()) {
        top++;
        data[top] = dt;
    } else {
        System.out.println("Isi stack penuh!");
    }
}
```

Pop Function

- To retrieve the data stack that is located at the top (data pointed to by the top of stack)
- **Show the value of the top element** of the stack by accessing its index according to the top of the stack, then **decrementing** the value of the **top** of stack is done so that the number of stack elements is reduced

Pop Function




```
public void pop() {
    if (!IsEmpty()) {
        int x = data[top];
        top--;
        System.out.println("Data yang keluar: " + x);
    } else {
        System.out.println("Stack masih kosong");
    }
}
```

Peek Function

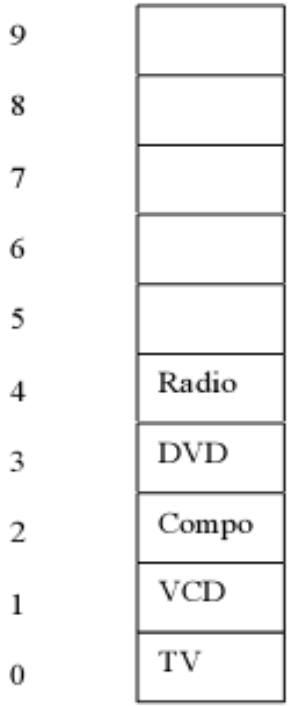
- To access the element pointed to by the top of stack, the element that was last added
- This operation is different from pop because it is not accompanied by deletion of data, but only accessing (returning) data only

```
public void peek() {
    System.out.println("Elemen teratas: " + data[top]);
}
```

Print Function

- To display all data stack elements
- By looping all array values in reverse, because we have to access from the highest array index first and then to the smaller indexes

Print Function



Jika kita ingin mengeprint elemen stack, kita harus membaca elemen array dari indeks yang ada isinya tertinggi, yaitu dari Top downto 0

Sehingga kita akan menampilkan elemen:
Radio, DVD, Compo, VCD, dan TV

```
public void print() {
    System.out.println("Isi stack: ");
    for (int i = top; i >= 0; i--) {
        System.out.println(data[i] + " ");
    }
    System.out.println("");
}
```

Clear Function

- To empty the stack by removing all stack elements

```
public void clear() {
    if (!IsEmpty()) {
        for (int i = top; i >= 0; i--) {
            top--;
        }
        System.out.println("Stack sudah dikosongkan");
    } else {
        System.out.println("Gagal! Stack masih kosong");
    }
}
```

Postfix Expressions

Expressions

Application of the stack in the field of arithmetic is the writing of mathematical expressions, which consist of three types:

- **Infix notation** with characteristics:
  - Operators are between operands: 3 + 4 * 2
  - Brackets are preferred: (3 + 4) * 2
- **Prefix notation**: operator is written **before** two operands
- **Postfix notation**: operator is written **after** two operands
- Ex:
  - 3 + 4 * 2 → + 3 * 4 2 → 3 4 2 * +
  - (3 + 4) * 2 → * + 3 4 2 → 3 4 + 2 *

  Infix Prefix Postfix

Postfix Expressions

- Typically, mathematical expressions are written using infix notation, but postfix notation is a notation used by a computer compilation engine to simplify the coding process

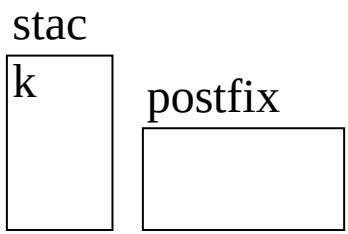
Arithmetic Operator Degrees

Arithmetic operator degree order:

- Pow ^
- Multiplication * is equivalent to division / and modulo%
- Addition + equivalent to subtraction -
- Open parenthesis (and close parenthesis)

Algorithm of Infix to Postfix Conversion

- Scan **problem** from the first character. The problem is notated as infix
- If it's operand, then enter to **postfix**
- If "(", then push to the **stack**
- If ")", then pop the contents of the **stack** until found the sign "(", then add to **postfix**, while the sign "(" is not added to postfix
- If **operator**, then:
  - If **stack** is still empty, then push the operator into **stack**
  - If the precedence of operator > precedence of operator at the top of **stack**
    - Push operator into **stack**
  - While the precedence of operator <= precedence of operator at the top of stack
    - Pop operator from top of **stack** and input it to **posfix**
    - After finished, push operator into **stack**
- If all of the equations has been read, pop all the contents of the stack and push to postfix accordance with the order



Case Study 1

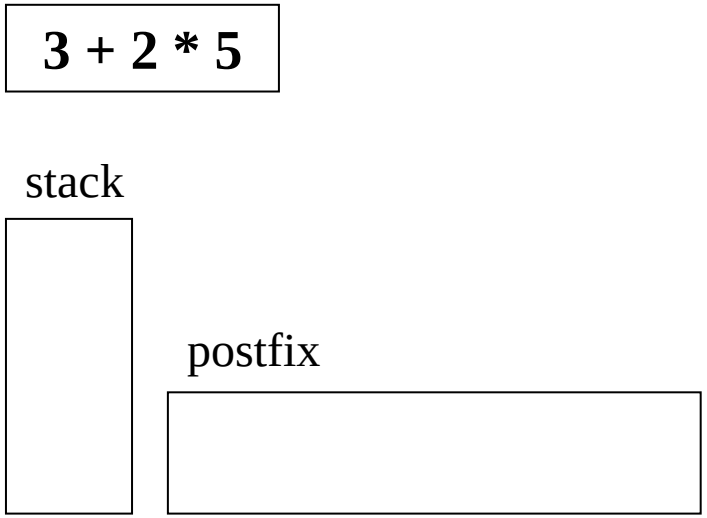
- For example there is an equation:

  3 + 2 * 5

- The above operation is called **infix** notation, the infix notation must be **changed to postfix** notation

Case Study 1

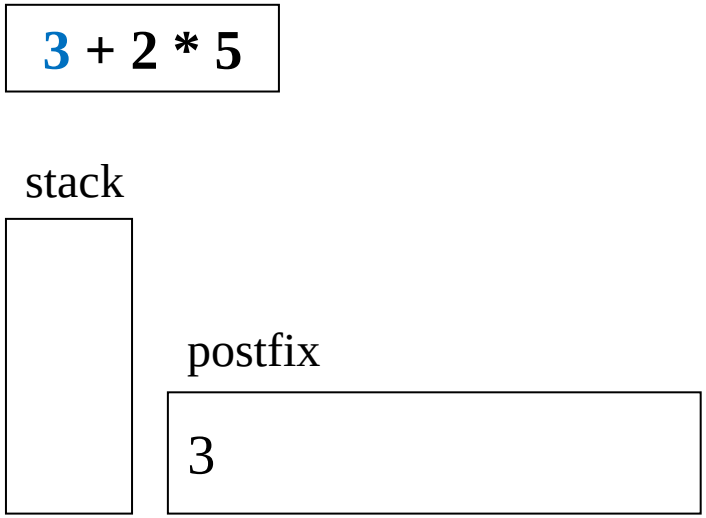
- Read the equation from left to right



Case Study 1

- Step 1: Operand 3

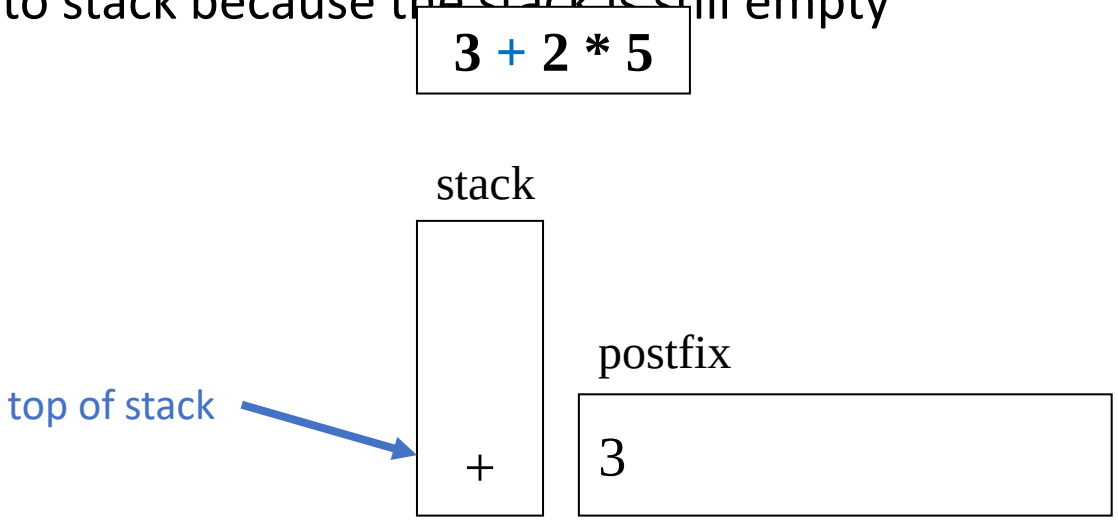
Enter postfix



Case Study 1

- Step 2: Operator +

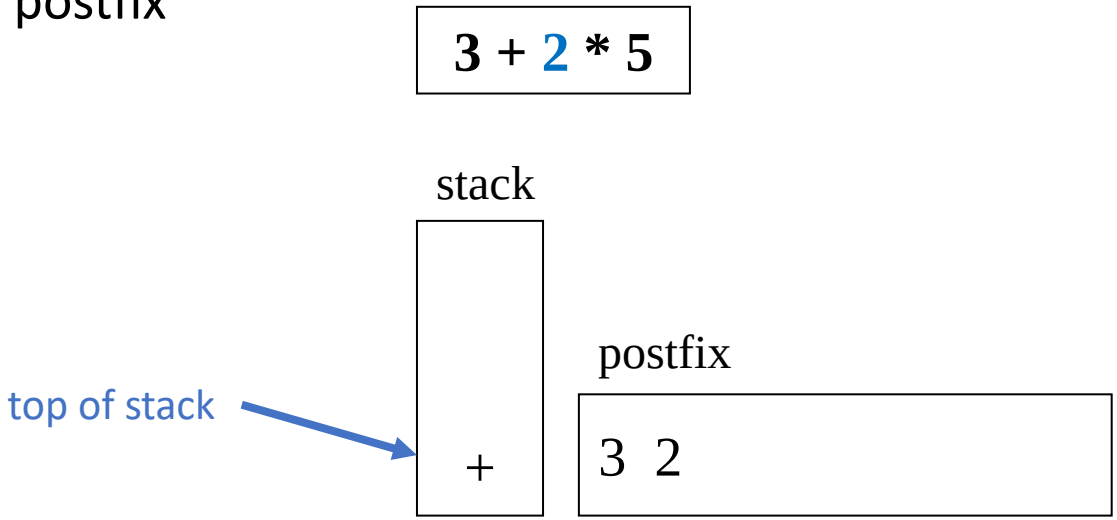
Push to stack because the stack is still empty



Case Study 1

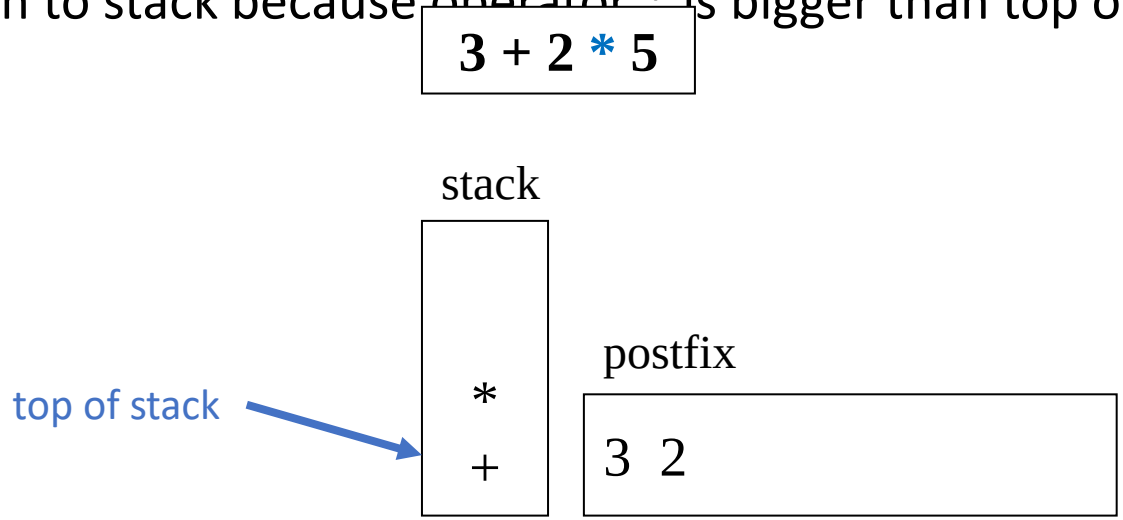
- Step 3: Operand 2

Enter postfix



Case Study 1

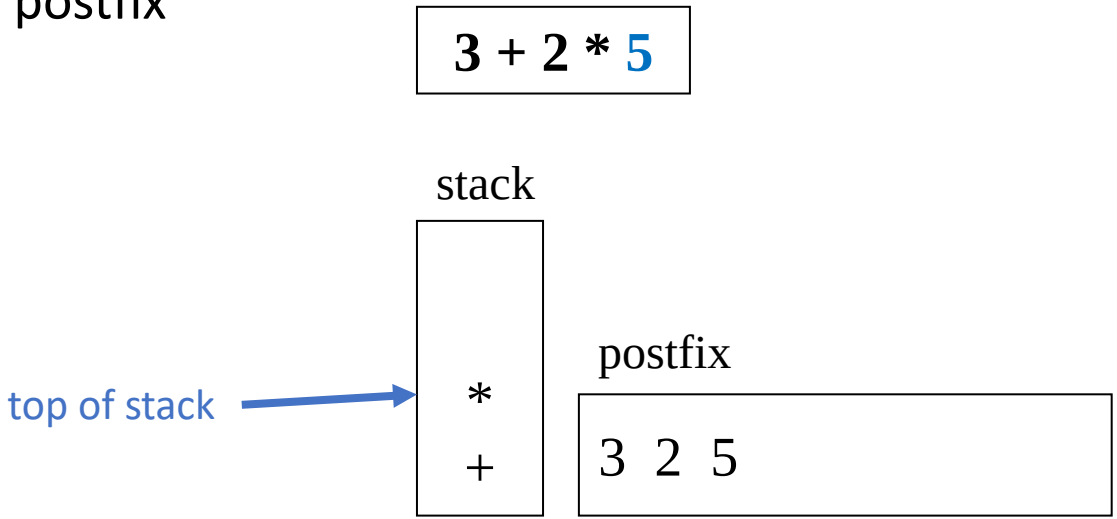
- Step 4: Operator *
- Push to stack because operator * is bigger than top of stack ie. +



Case Study 1

- Step 5: Operand 5

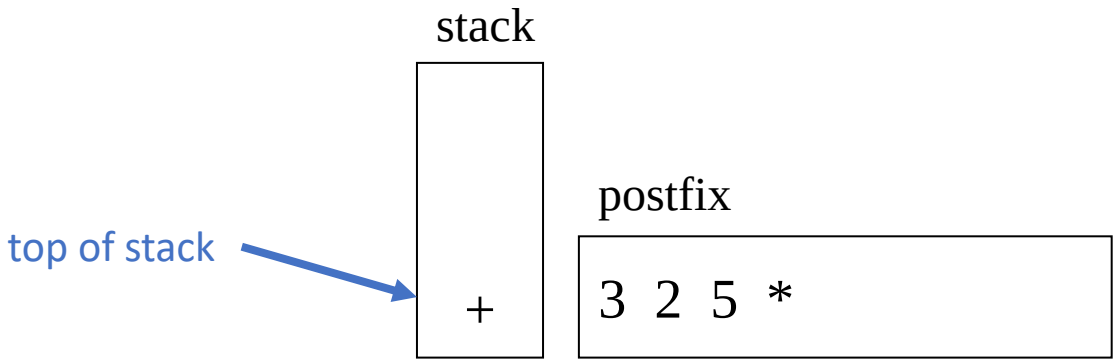
Enter postfix



Case Study 1

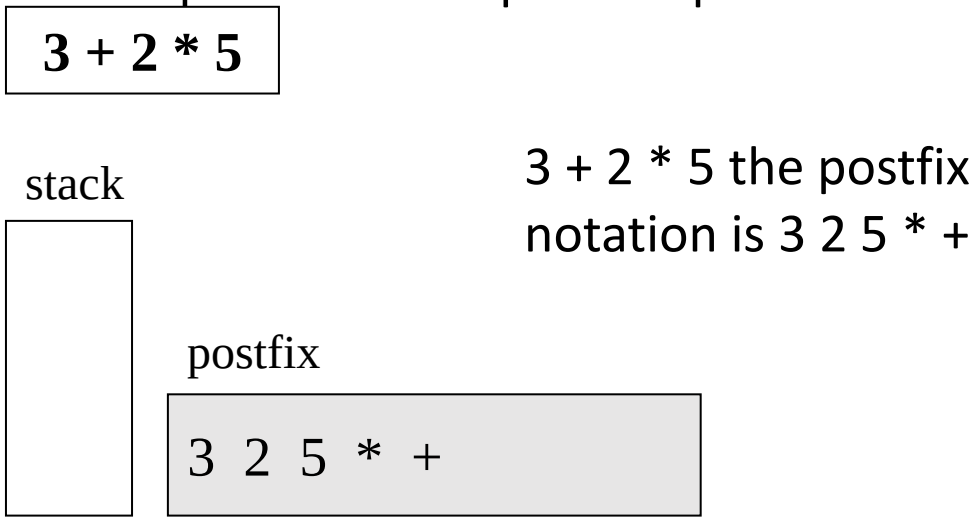
- Step 6

All the equations are read, pop all the contents of the stack and put them in postfix in seque **3 + 2 * 5** the * operator first



Case Study 1

- Step 7 After the pop is done on the operator * and put into postfix, then the pop is done on the operator + and put into postfix



3 + 2 * 5 the postfix notation is 3 2 5 * +

Case Study 2

- Misalkan terdapat persamaan:

  $$15 - (7 + 4) / 3$$

- Operasi di atas disebut notasi **infix**, notasi infix tersebut harus diubah menjadi notasi **postfix**

Case Study 2

- Read the equation from left to right

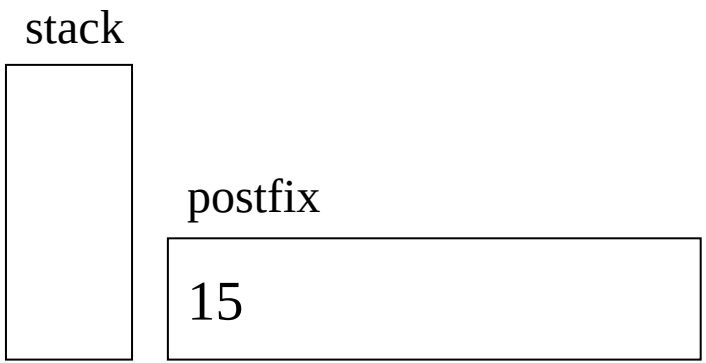
15 – (7 + 4) / 3

Case Study 2

- Step 1: Operand 15

Enter postfix

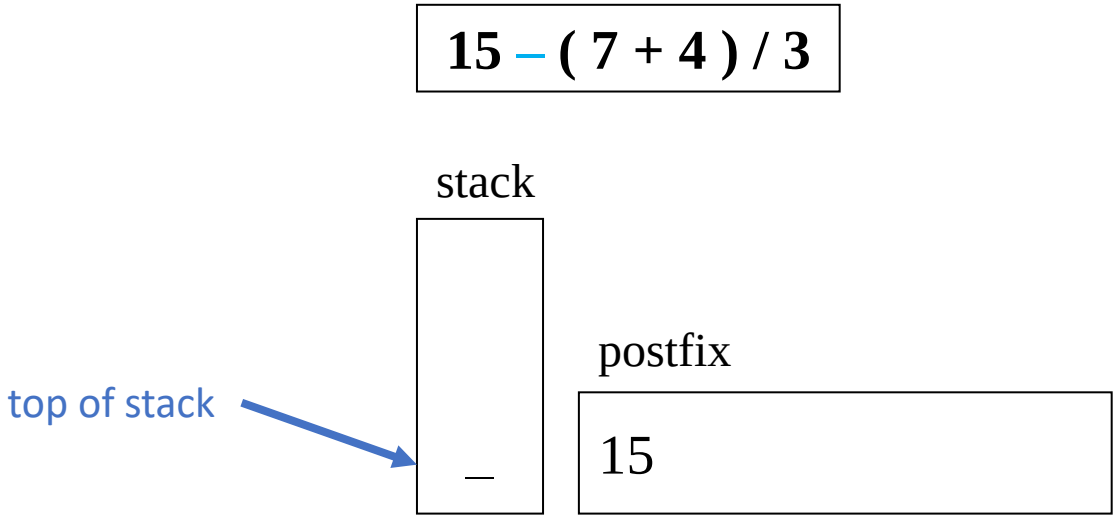
15 – (7 + 4) / 3



Case Study 2

- Step 2: Operator –

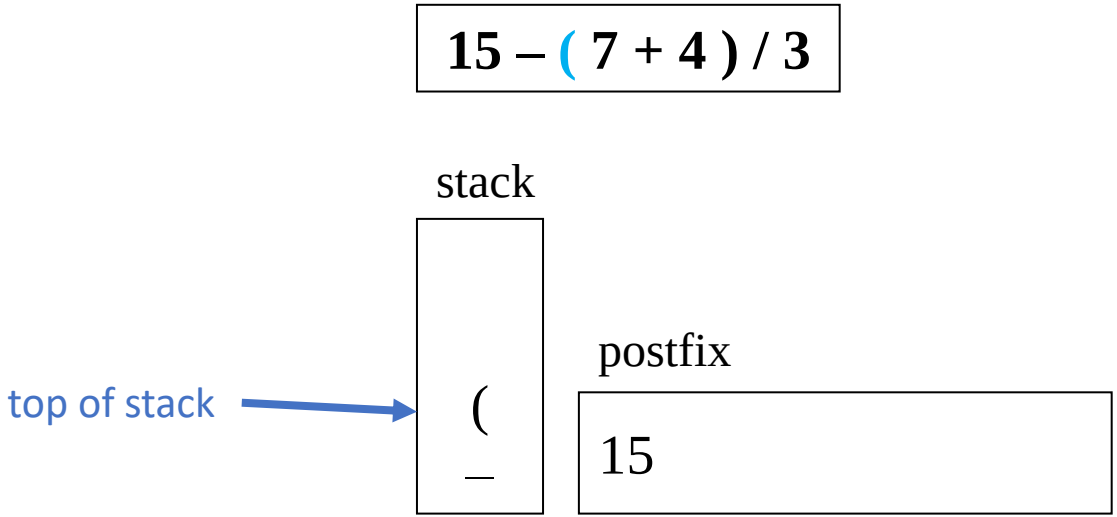
Push to stack because the stack is still empty



Case Study 2

- Step 3: Sign (

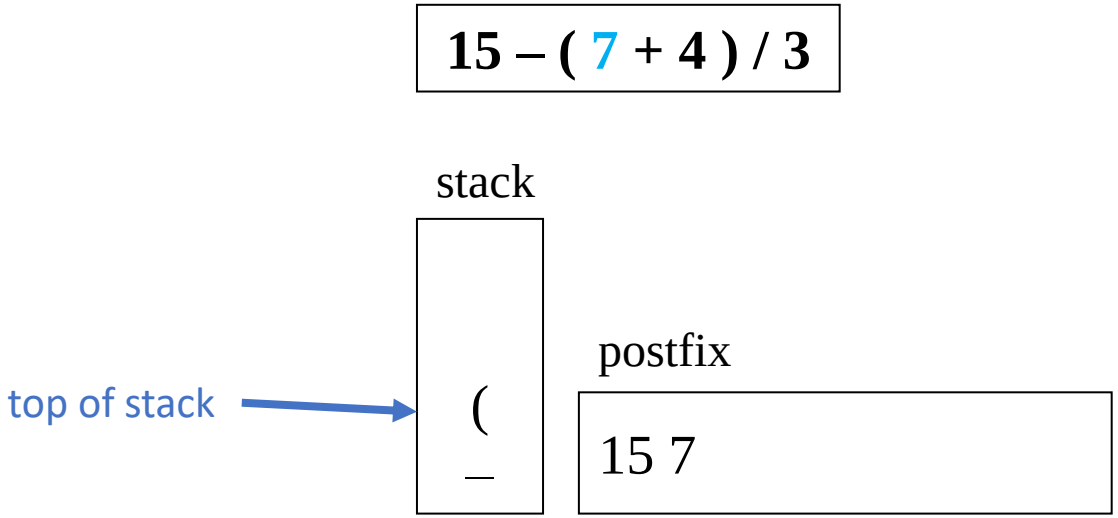
Push to stack



Case Study 2

- Step 4: Operand 7

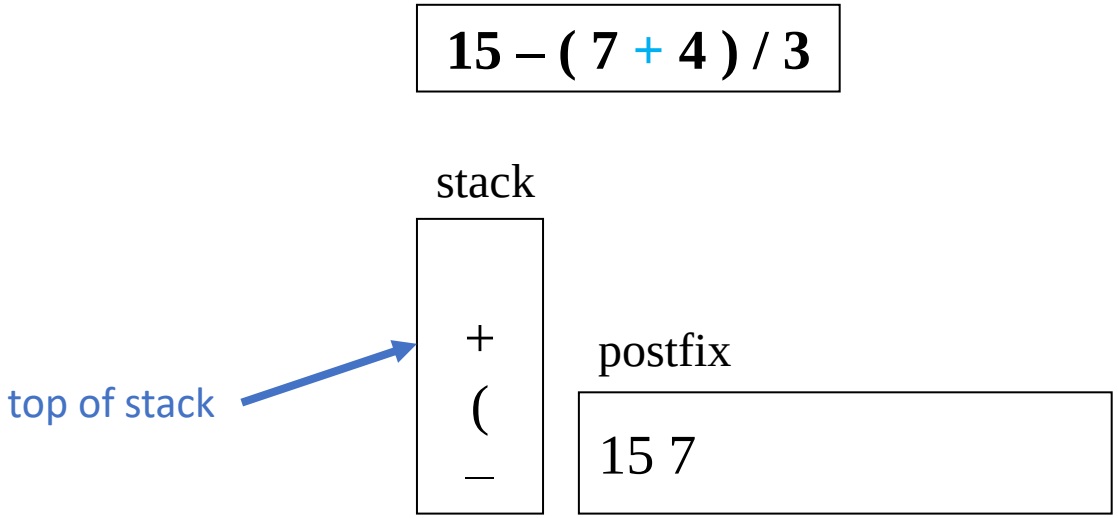
Enter postfix



Case Study 2

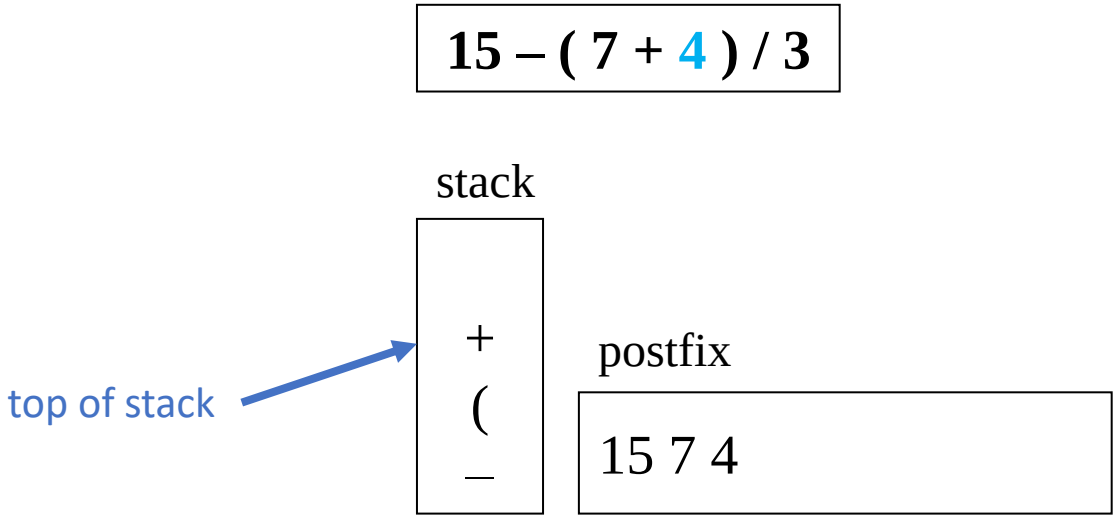
- Step 5: Operator +

Push to stack because + is bigger than top of stack ie. (



Case Study 2

- Step 6: Operand 4
- Enter postfix

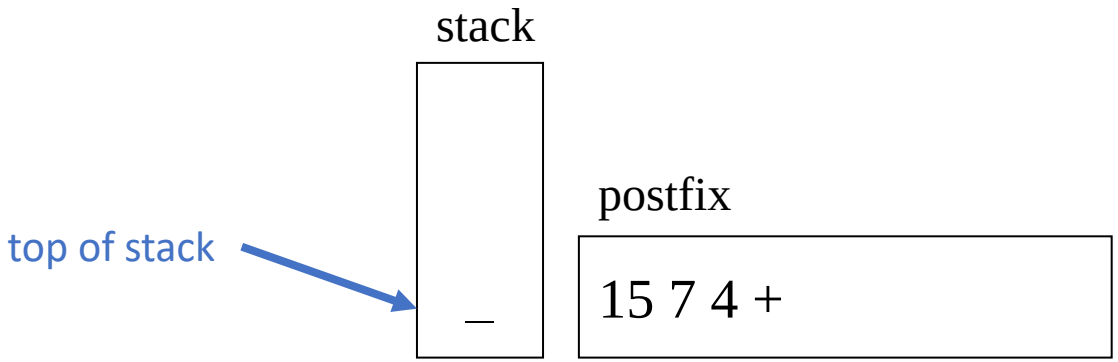


Case Study 2

- Step 7: Sign)

Pop the contents of the stack ie operator +, then enter into postfix. Alerts (only pop need not be added to postfix

15 – (7 + 4) / 3

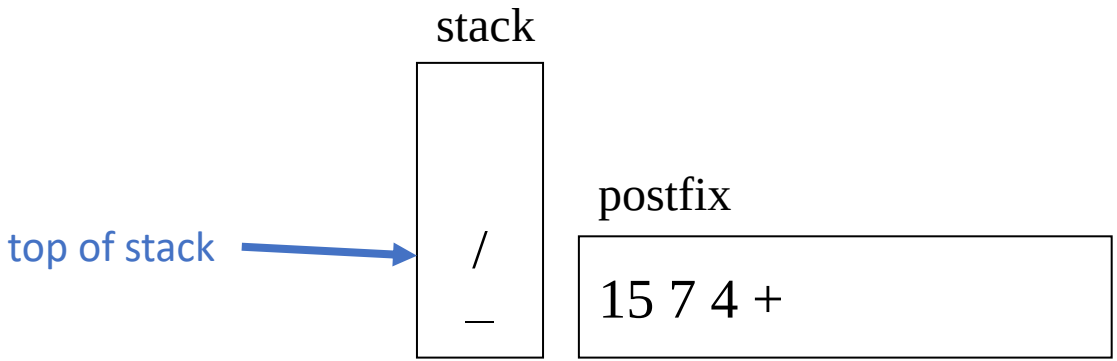


Case Study 2

- Step 8: Operator /

Push to stack because operator / is greater than top of stack ie. -

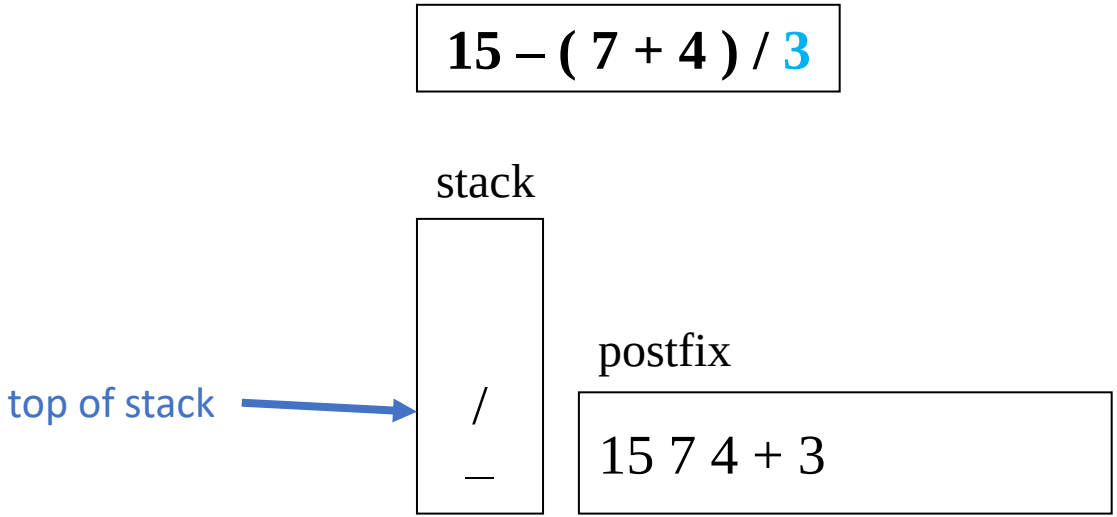
15 – (7 + 4) / 3



Case Study 2

- Step 9: Operand 3

Enter postfix

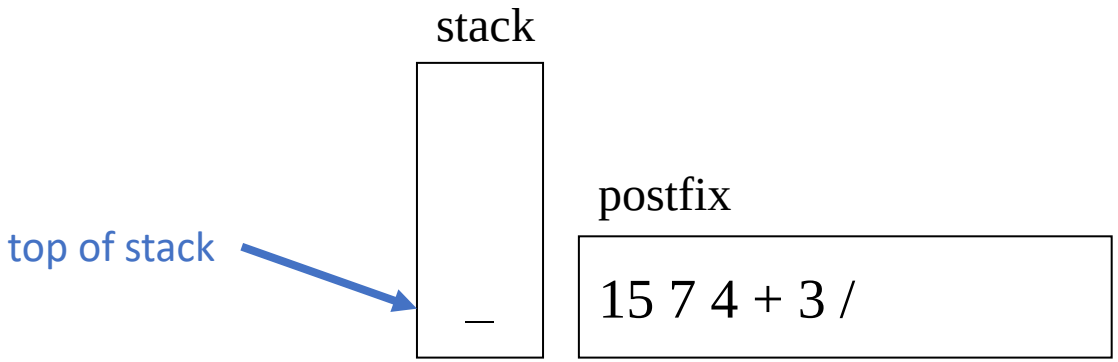


Case Study 2

- Step 10

All expressions are read, pop all the contents of the stack and put them in postfix in sequence, that is, the operator / first

15 – (7 + 4) / 3



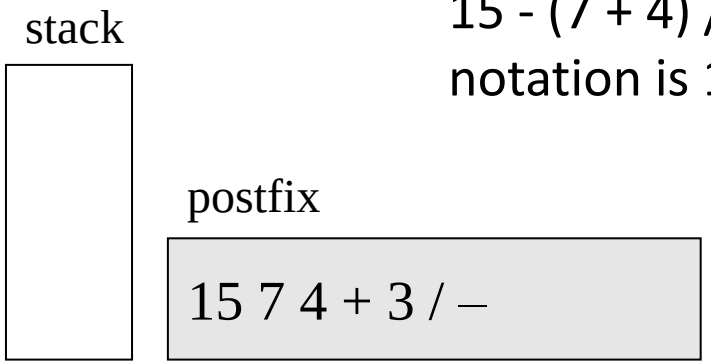
Case Study 2

- Step 11

After you pop the operator / and put it into postfix, then you pop it into the operator - and put it in postfix

15 – (7 + 4) / 3

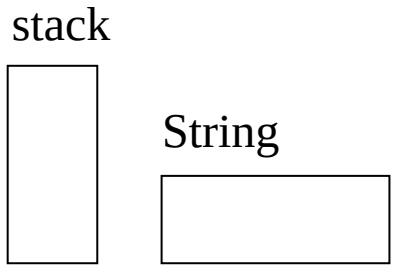
15 - (7 + 4) / 3 the postfix notation is 15 7 4 + 3 / -



Decimal to Binary Conversion

Algorithm of Decimal to Binary Conversion

Create and initialize **stack** to save modulo
WHILE decimal != 0 **DO**
    Calculate modulo = decimal % 2
    Push modulo into stack
    Devide decimal by 2 (int value)
END WHILE
Create an empty string to save binary
WHILE stack is not empty **DO**
    Pop modulo
    Put modulo into string.
END WHILE
Return string

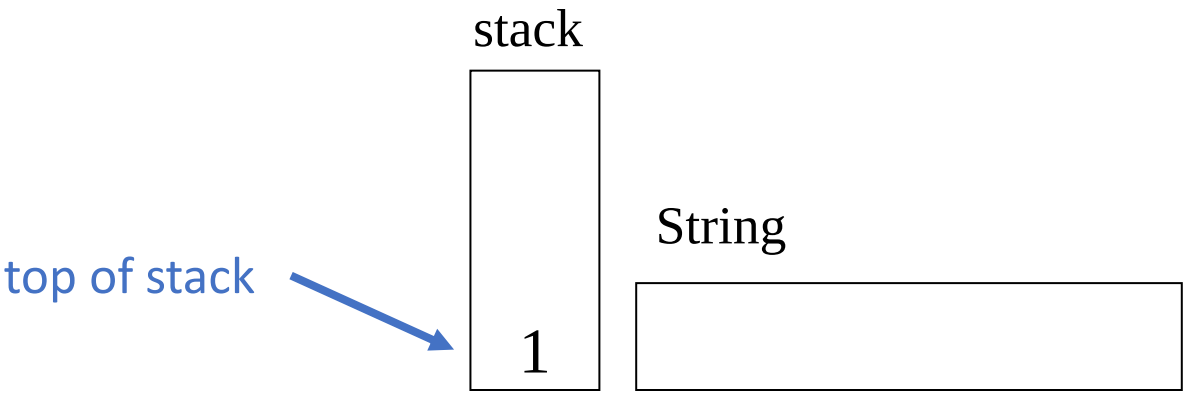


Example

- Convert **11** from decimal to binary

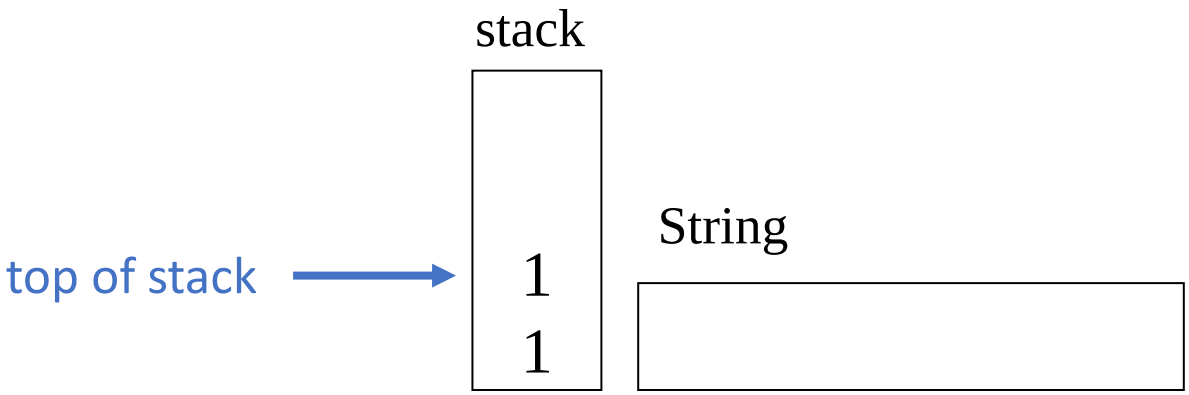
Example

- Decimal = **11**
- Modulo = 11 % 2 = 1, push into stack
- Update decimal = 11 / 2 = **5**



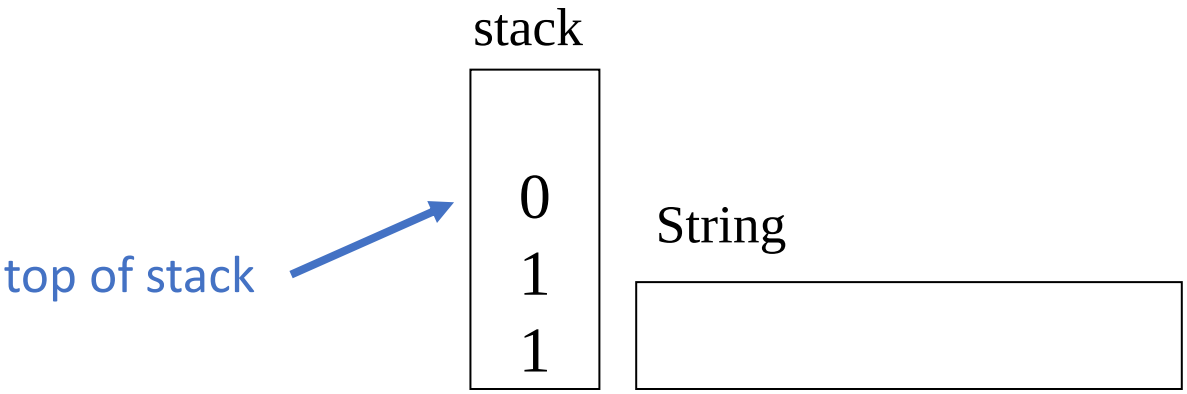
Example

- Decimal = **5**
- Modulo = 5 % 2 = **1**, push into stack
- Update decimal = 5 / 2 = **2**



Example

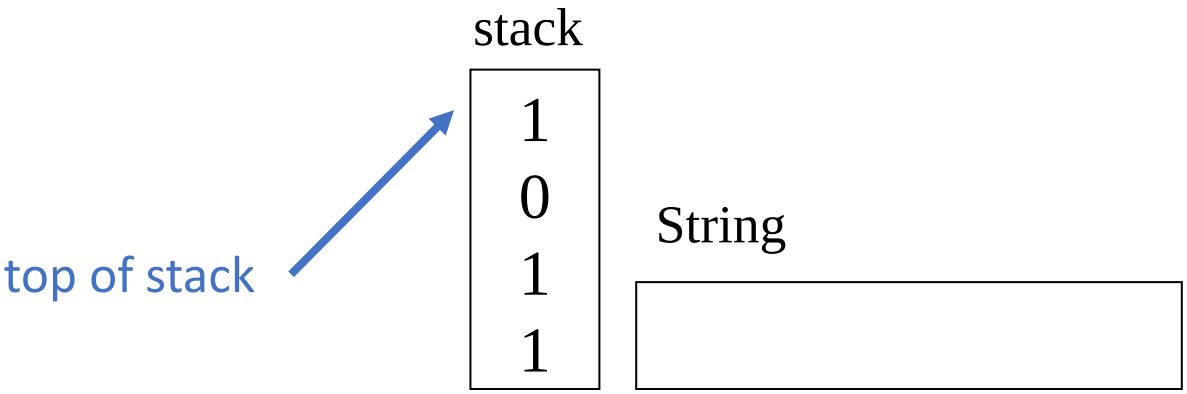
- Decimal = 2
- Modulo = 2 % 2 = 0, push into stack
- Update decimal = 2 / 2 = 1



Example

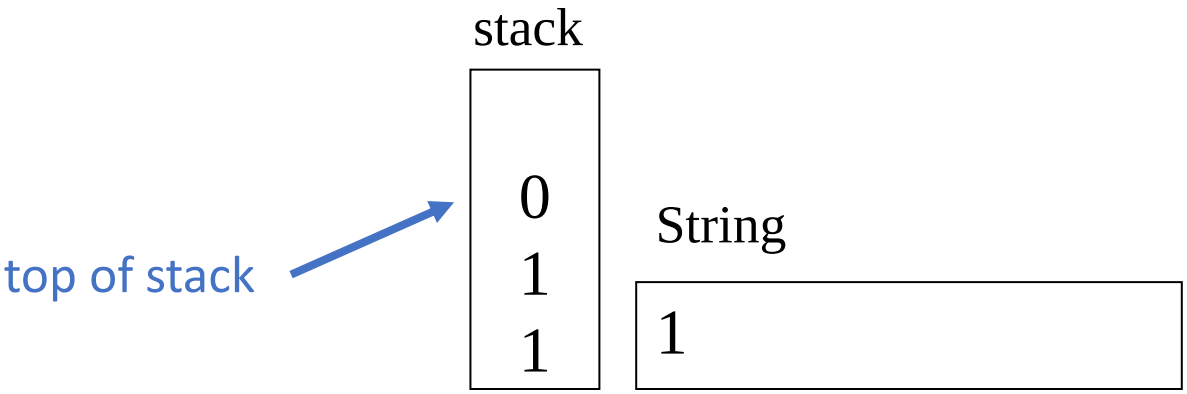
- Decimal = 1
- Modulo = 1 % 2 = 1, push into stack
- Update decimal = 1 / 2 = 0

Since the lates decimal is 0, then this is the final step



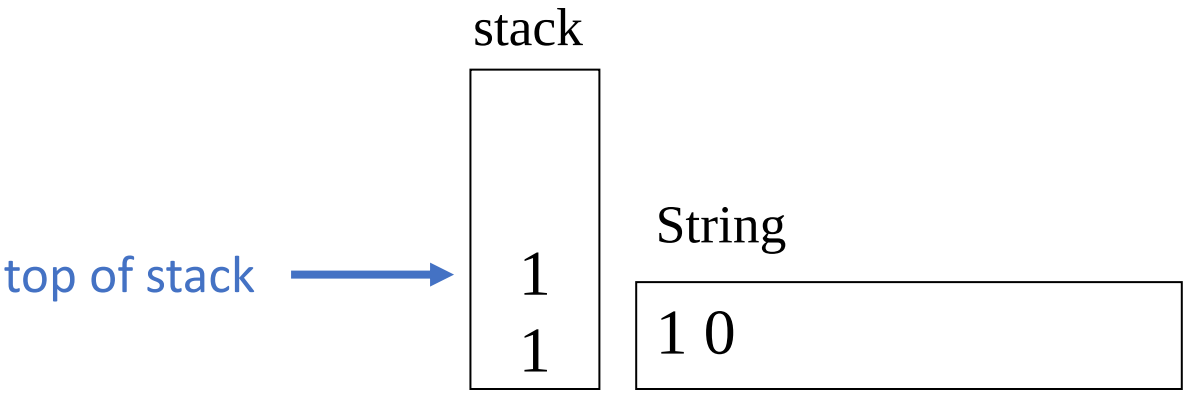
Example

- Pop stack and put into String



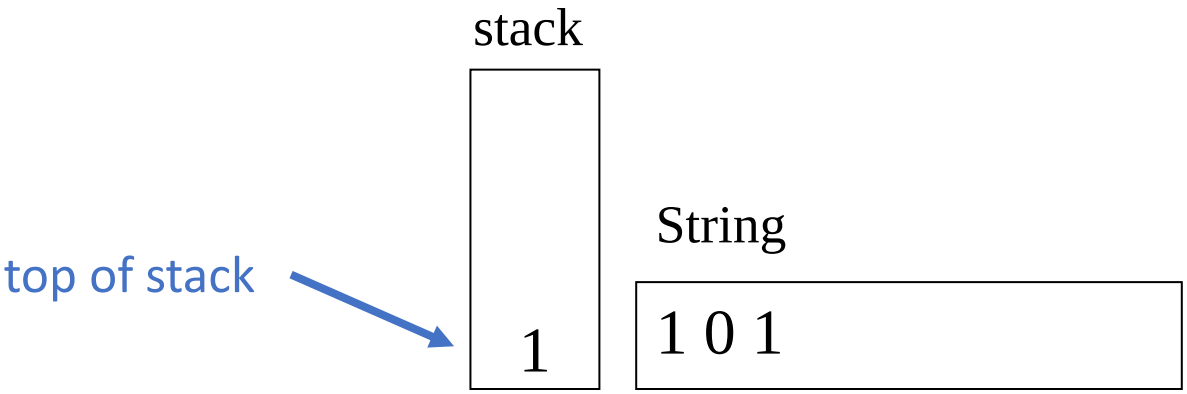
Example

- Pop stack and put into String



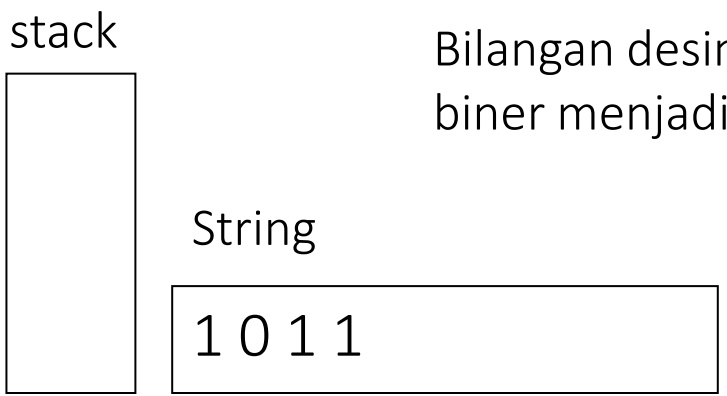
Example

- Pop stack and put into String



Example

- Pop stack and put into String



Bilangan desimal 11 dikonversi ke biner menjadi 1011

Calculating Postfix Expression

Calculating Mathematical Expressions

- Mathematical expressions arranged in the form of postfix notation can be calculated the end result
- Example:

  2 5 *

  Result: 10

Algorithm for Calculating Mathematical Expressions

An example of P is a mathematical expression written in postfix notation and the Q is the result variable

- Read equation P from left to right, determine the next symbol
- If it's an operand, then push to the stack
- If the operator (called opt), then
  - Pop the top 1 element of the stack, save it in variable X
  - Pop the top 1 element of the stack, save it in the Y variable
  - Calculate variable (Y opt X), save the result in variable H
  - Push the H variable onto the stack
- Repeat the above steps until all characters in P scanned
- Pop the contents of the stack and save it in the Q variable as the final result

Case study

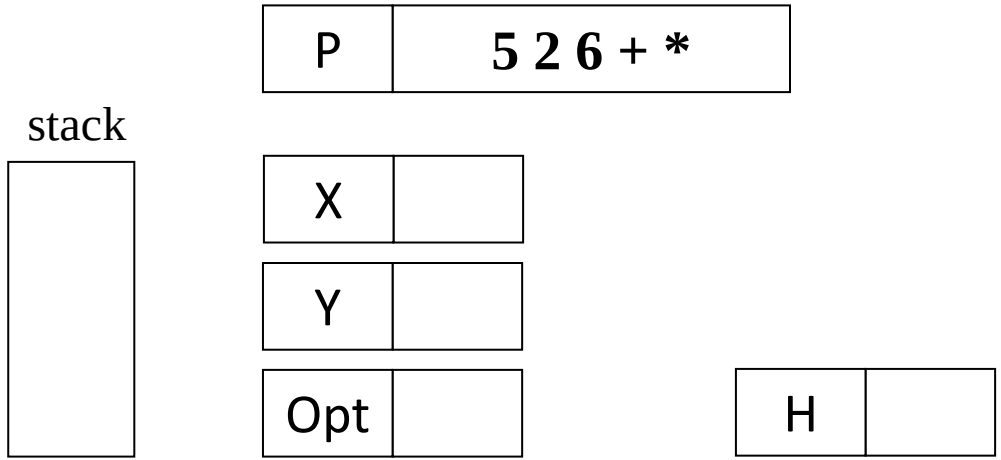
- For example, there is a mathematical equation in the form of postfix notation

  P = 5 2 6 + *

- The results of the mathematical equation can be calculated without the need to change to infix notation

Case study

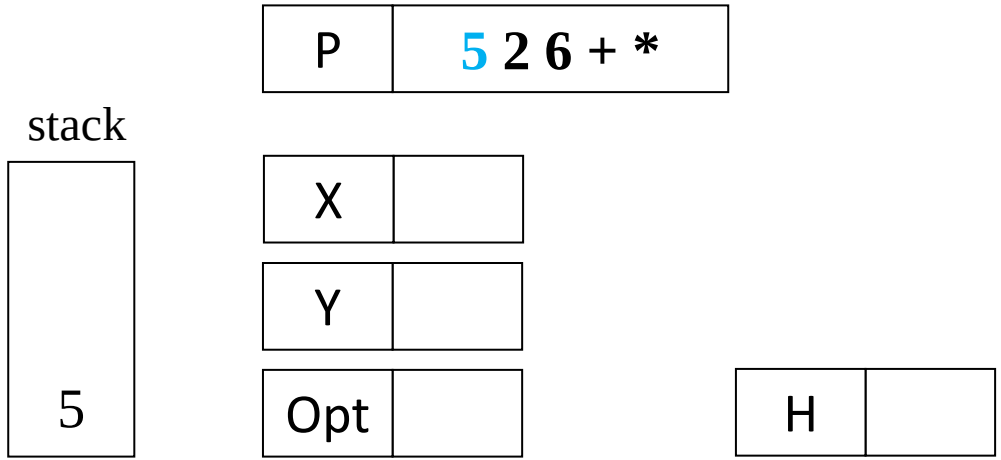
- Read equation P from left to right



Case study

- Step 1: Operand 5

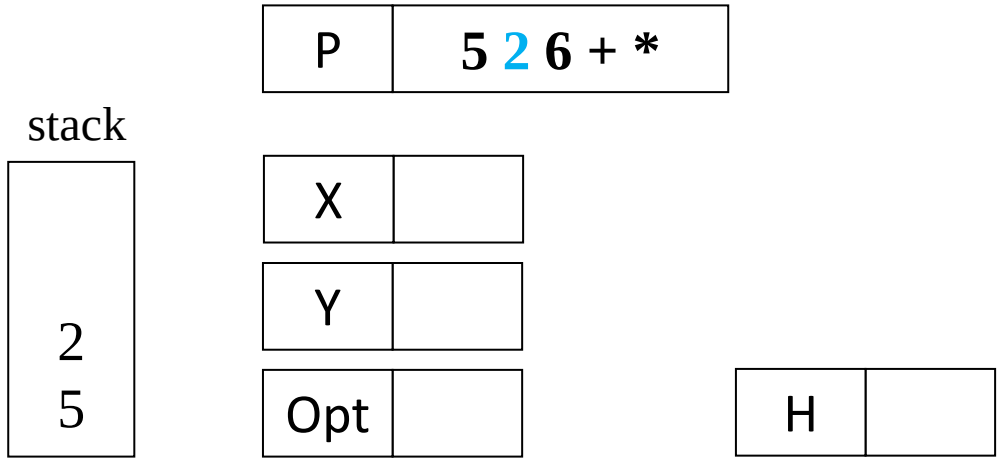
Push to stack



Case study

- Step 2: Operand 2

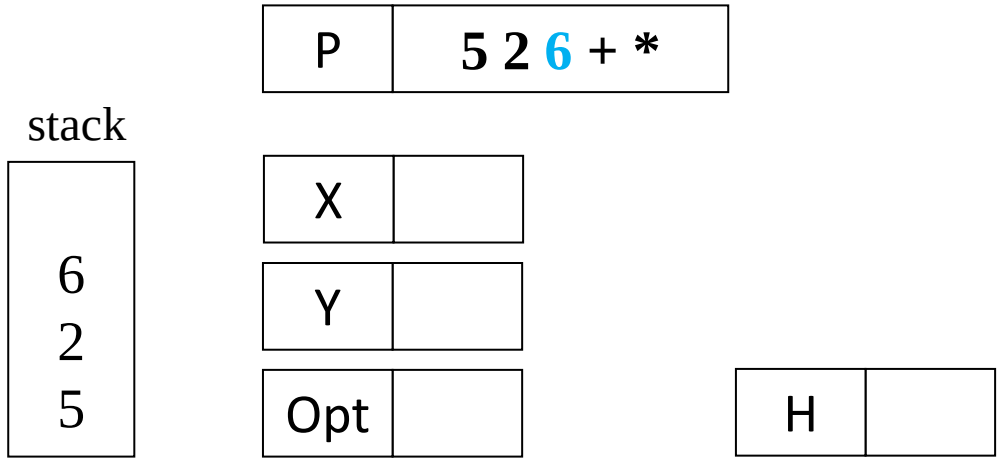
Push to stack



Case study

- Step 3: Operand 6

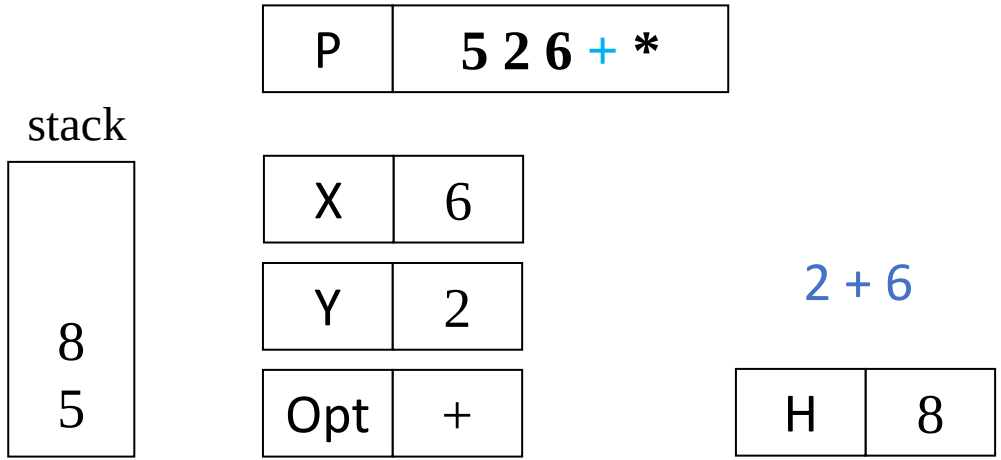
Push to stack



Case study

- Step 4: Operator +

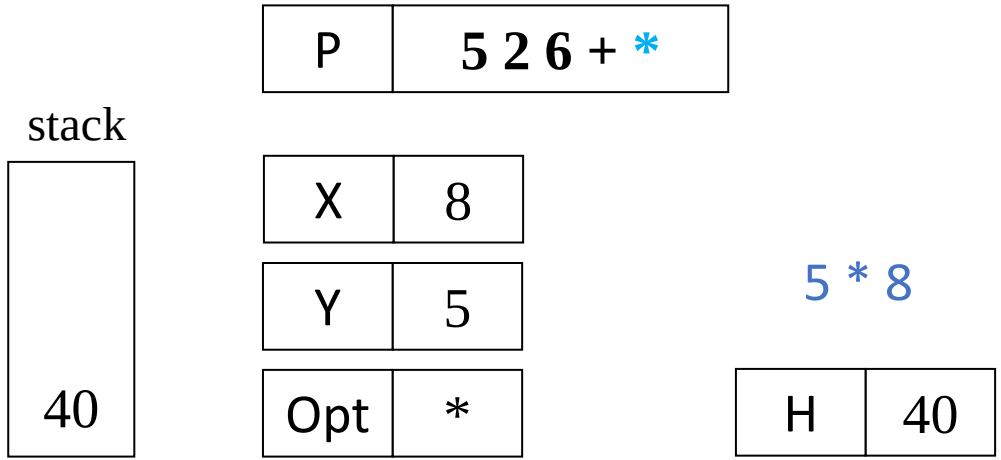
Pop the top element 6 then save it in X and pop the next top element 2 then save it to Y. Calculate Y Opt X, save the result in the variable H then push H to the stack



Case study

- Step 5: Operator *

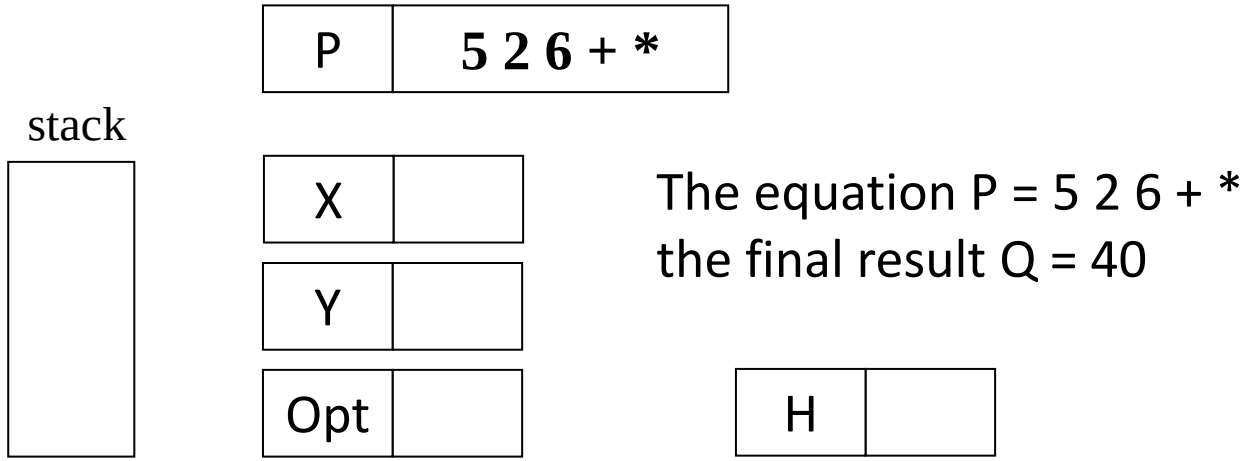
Pop the top element that is 8 then save it in X and pop the next top element that is 5 then save it to Y. Calculate Y Opt X, save the result in the variable H then push H to the stack



Case study

- Step 6:

Pop the contents of the stack and save it in the Q variable as the final result



Questions

- Create the ilustrations of the Infix-to-Postfix conversion for the following operations:
  - x + y ^ z – w
  - 2 + 4 * (9 - 5) / 3
  - 12 - 3 ^ (4 % 2)
- Convert the following decimal numbers into binary using the Stack!
  - 14
  - 23