INFORMATIKA UMN

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

IF232
ALGORITHMS
&
DATA STRUCTURES

06
STACKS

DENNIS GUNAWAN

REVIEW

Linked Lists:

Double Linked Lists

Circular Linked Lists

OUTLINE

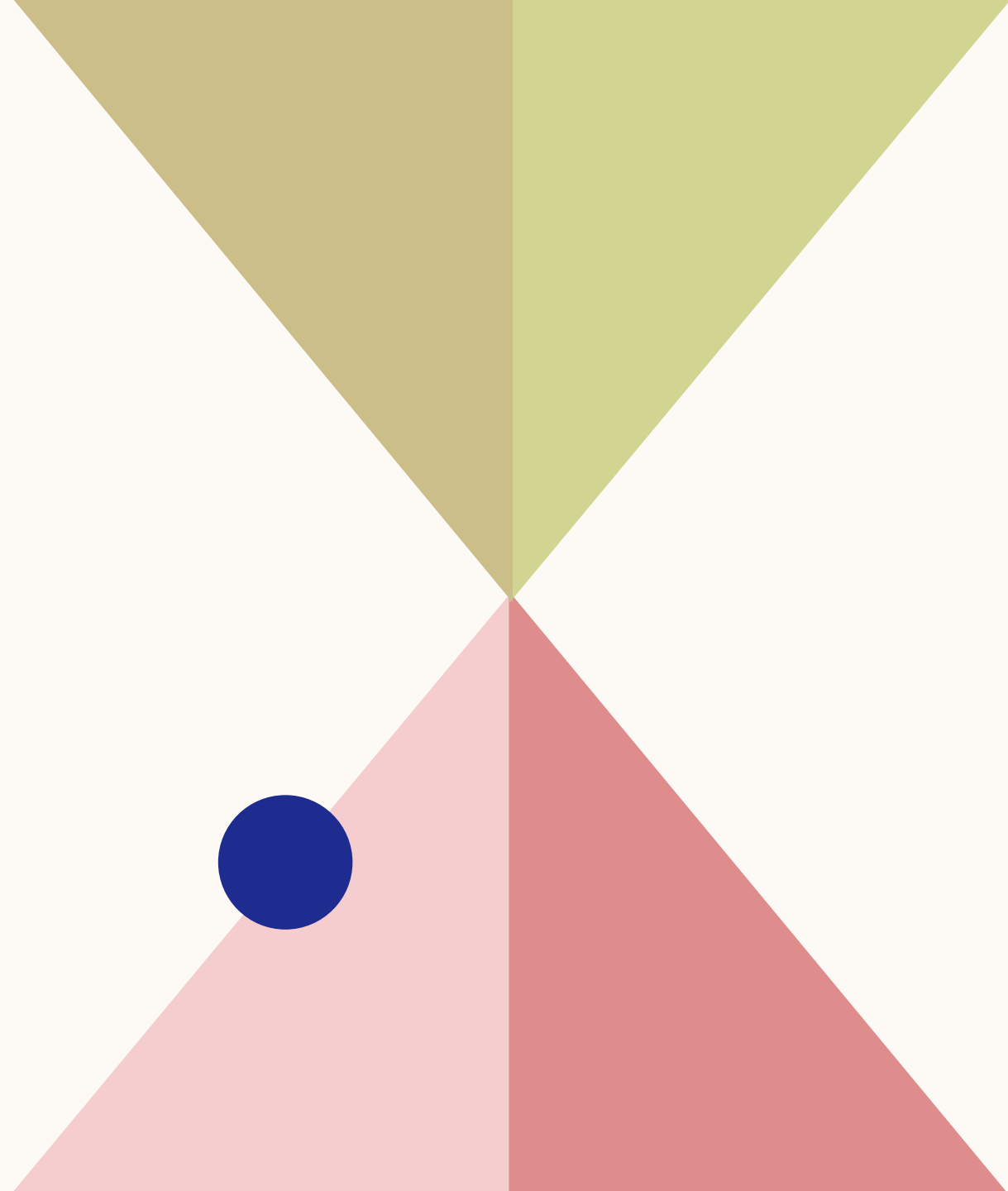
Array Representation of Stacks

Operations on a Stack

Linked Representation of Stacks

Operations on a Linked Stack

Applications of Stacks

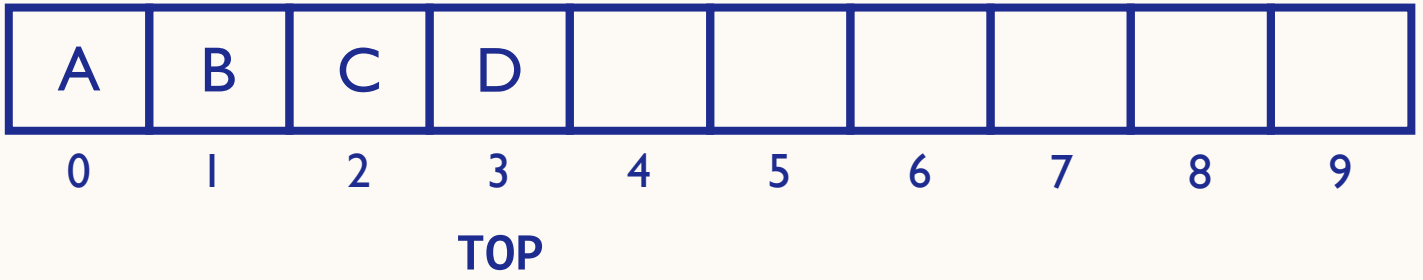


STACKS

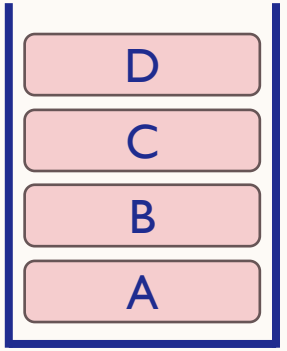
- LIFO: Last In First Out
- **TOP** is used to store the address of the topmost element of the stack
  - It is this position where the element will be added to or deleted from
- **SIZE** or **MAX** is used to store the maximum number of elements that the stack can hold

ARRAY REPRESENTATION OF STACKS

MAX = 10

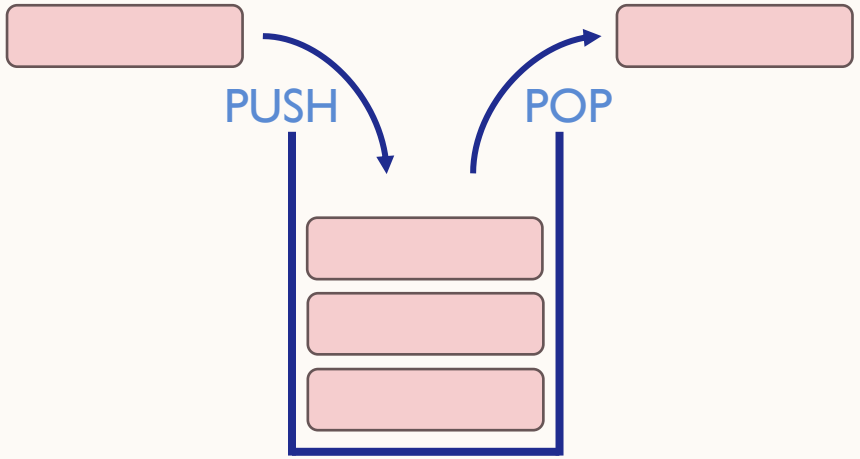


- The stack is empty: **TOP = -1**
- The stack is full: **TOP = MAX-1**



OPERATIONS ON A STACK

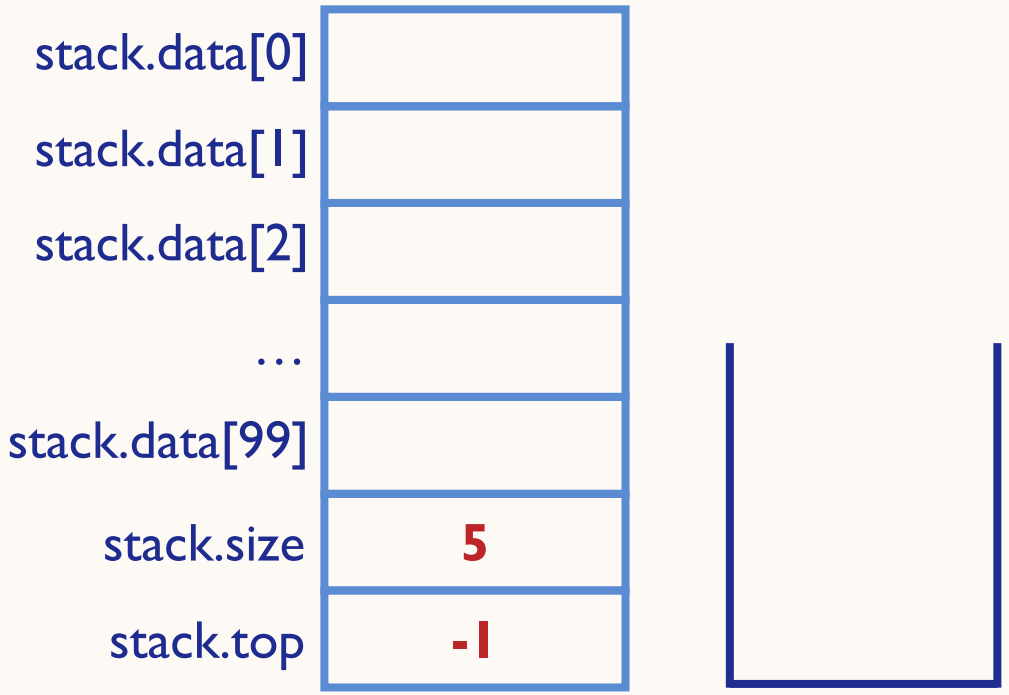
- **Push**: adds an element to the top of the stack
- **Pop**: removes the element from the top of the stack
- **Peek**: returns the value of the topmost element of the stack



DECLARATION

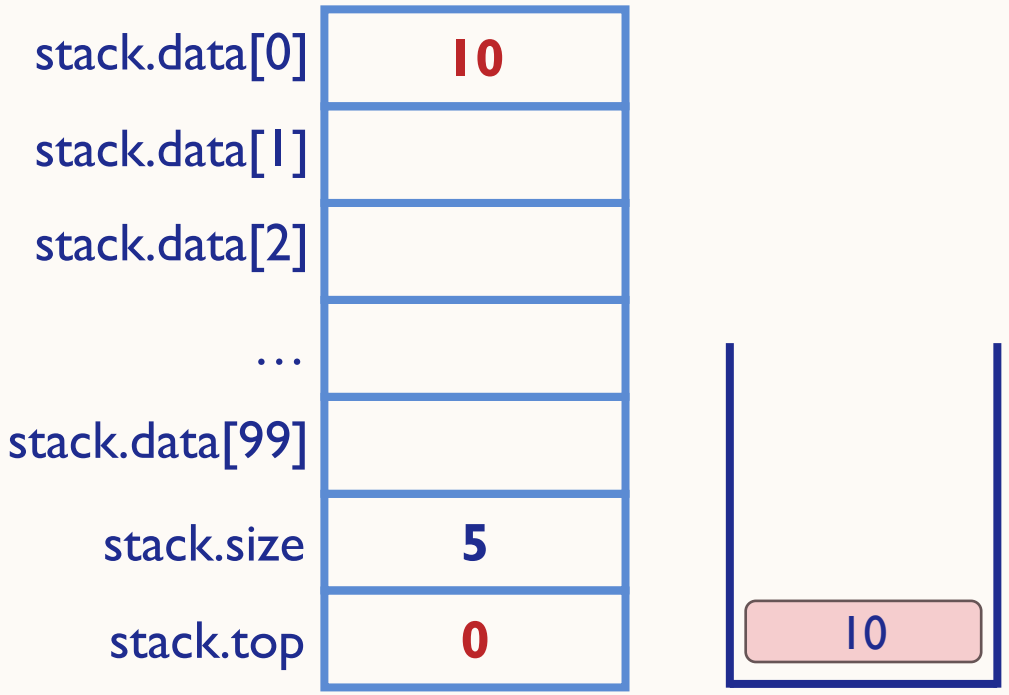
```
struct tstack{
    int data[100];
    int size;
    int top;
};
```

```
int main()
{
    struct tstack stack;
    int number;
    ...
    stack.size = 5;
    stack.top = -1;
    ...
}
```



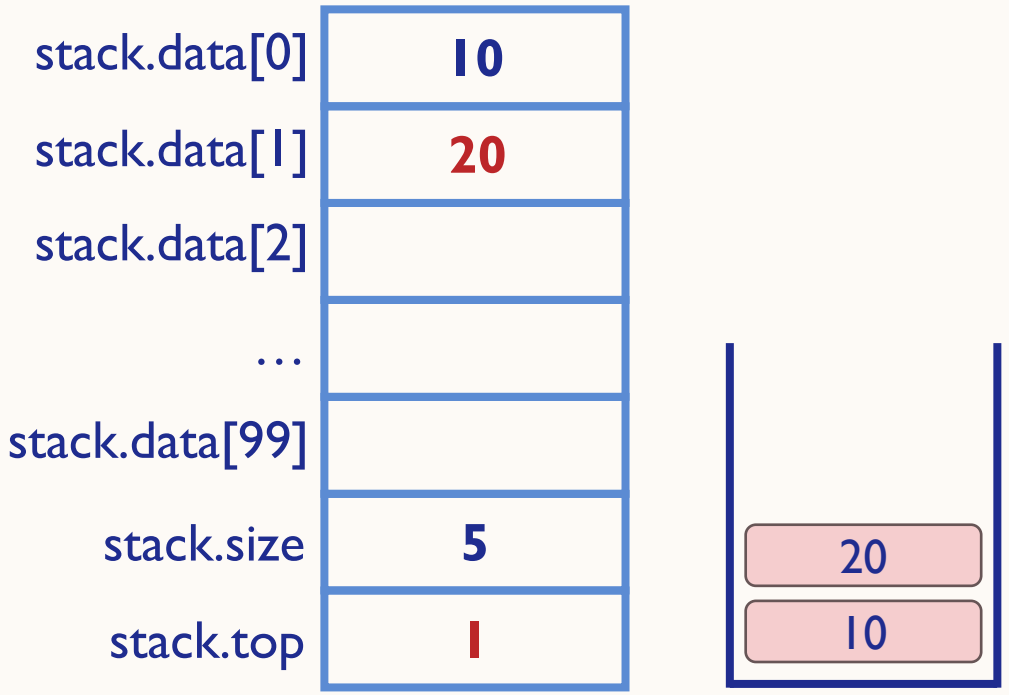
PUSH OPERATION

```
scanf("%d", &number); //10

if(stack.top == stack.size - 1)
    printf("OVERFLOW");
else{
    stack.top++;
    stack.data[stack.top] = number;
}
```



PUSH OPERATION

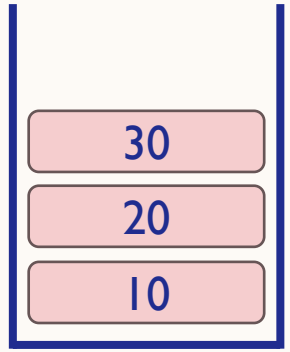
```
scanf("%d", &number); //20

if(stack.top == stack.size - 1)
    printf("OVERFLOW");
else{
    stack.top++;
    stack.data[stack.top] = number;
}
```



PUSH OPERATION

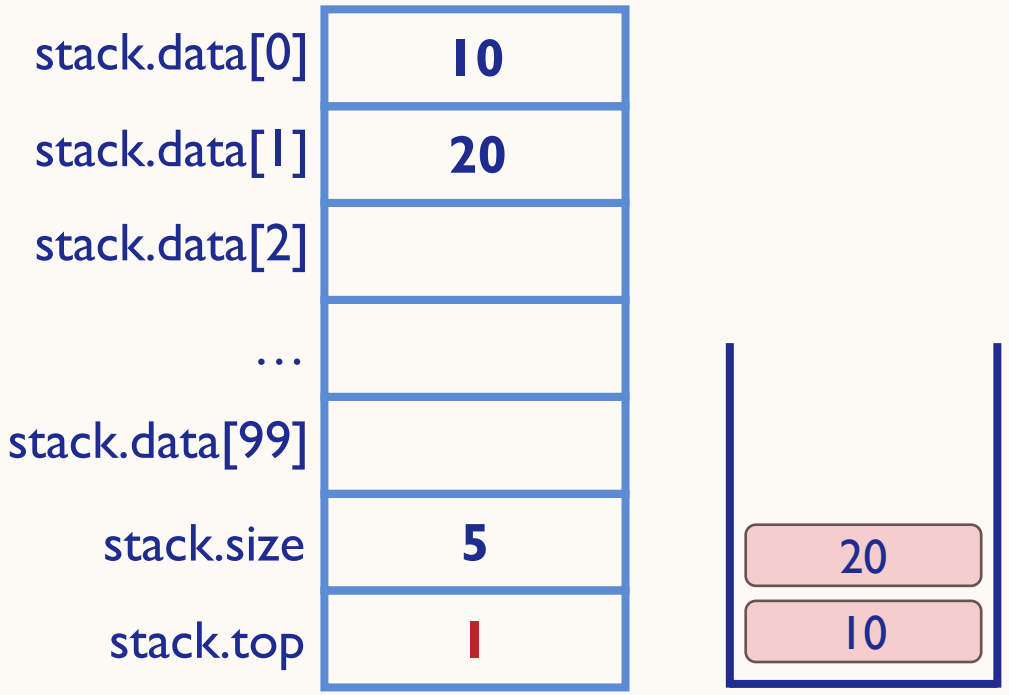
```
scanf("%d", &number); //30

if(stack.top == stack.size - 1)
    printf("OVERFLOW");
else{
    stack.top++;
    stack.data[stack.top] = number;
}
```

stack.data[0]	10
stack.data[1]	20
stack.data[2]	30
…	
stack.data[99]	
stack.size	5
stack.top	2



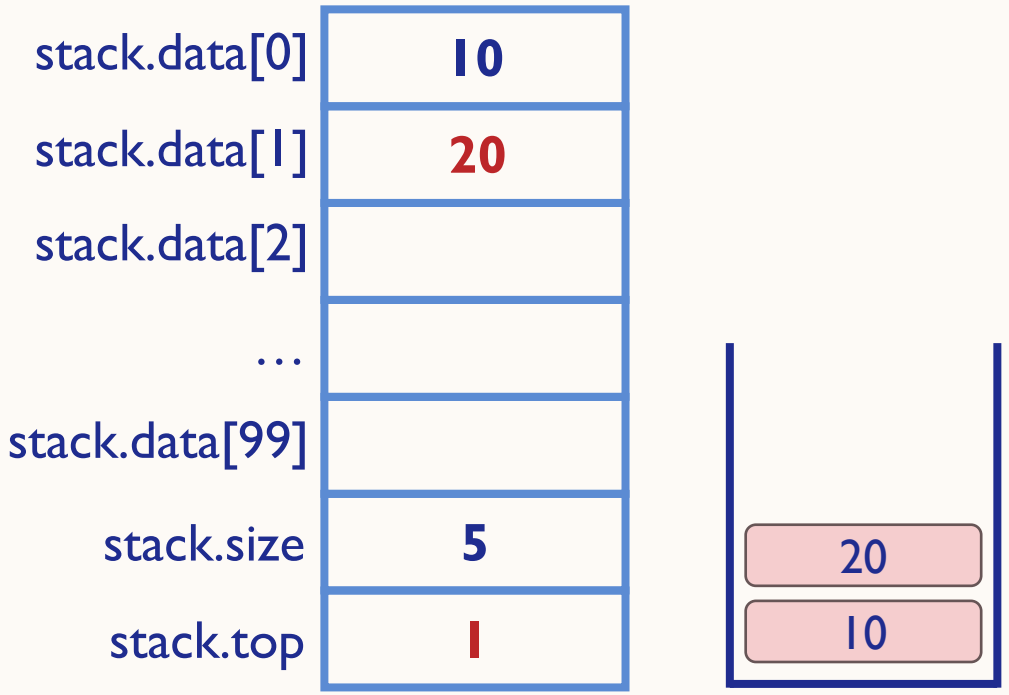
POP OPERATION

```
if(stack.top == -1)
    printf("UNDERFLOW");
else{
    number = stack.data[stack.top]; //30
    stack.top--;
    //process number
}
```

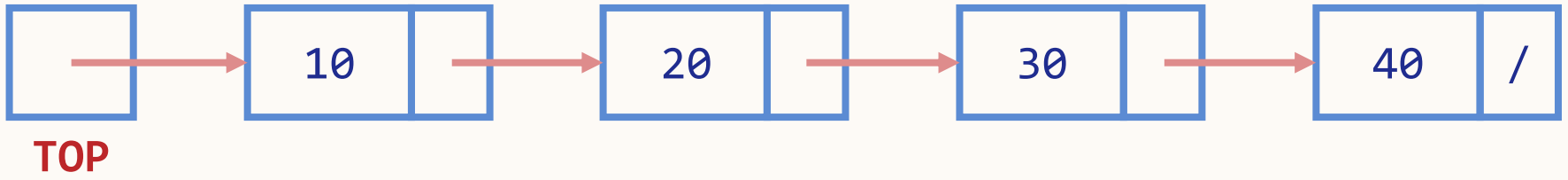


PEEK OPERATION

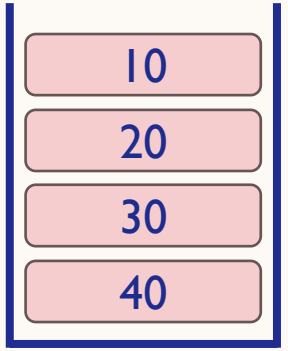
```
if(stack.top == -1)
    printf("EMPTY");
else{
    //process stack.data[stack.top]; 20
}
```



LINKED REPRESENTATION OF STACKS



- All insertions and deletions are done at the node pointed by **TOP**
- The stack is empty: **TOP = NULL**



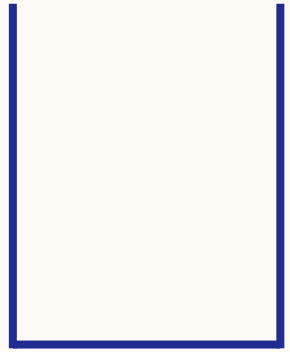
DECLARATION

```
struct tstack{
    int data;
    struct tstack *next;
};
```

```
int main()
{
    struct tstack *top, *node;
    int number;
    ...
    top = NULL;
    ...
}
```

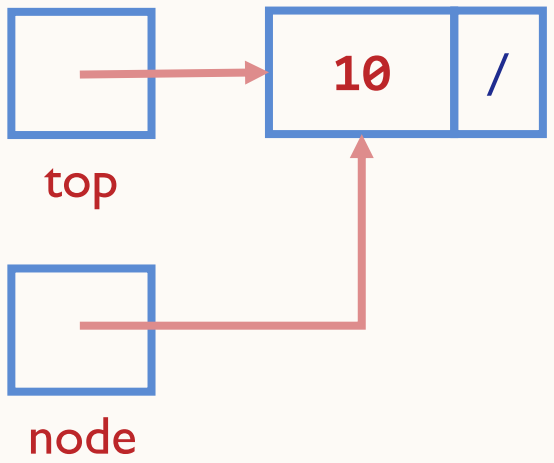
top

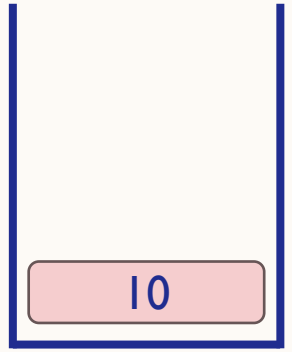
node



PUSH OPERATION

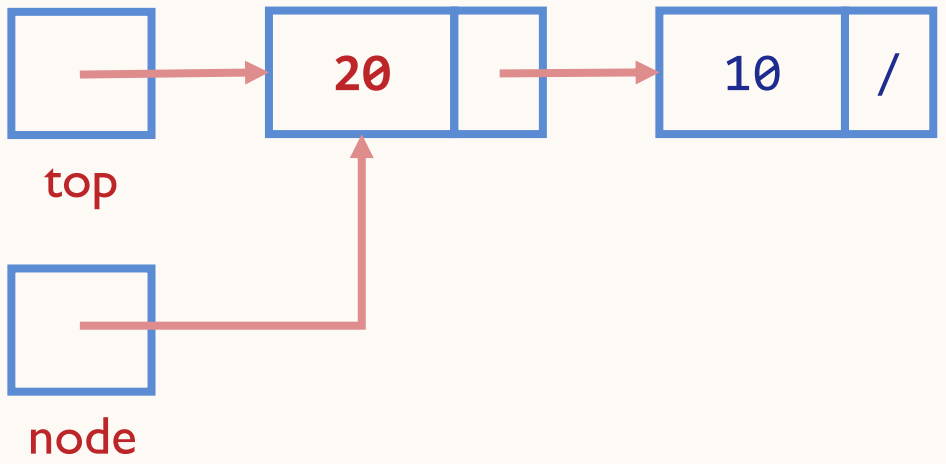
```
scanf("%d", &number); //10

node = (struct tstack *) malloc
       (sizeof(struct tstack));
node->data = number;

if(top == NULL)
    node->next = NULL;
else
    node->next = top;
top = node;
```

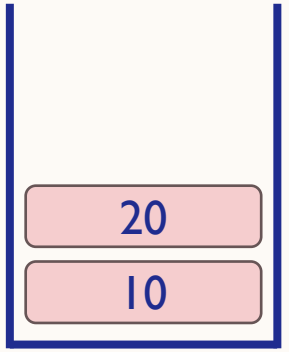




PUSH OPERATION

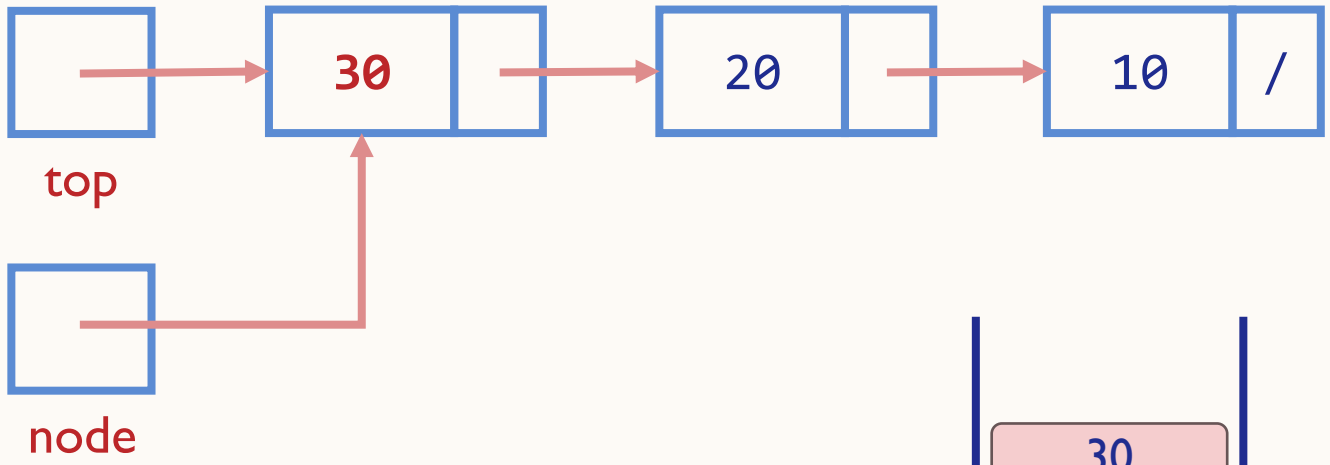
```
scanf("%d", &number); //20

node = (struct tstack *) malloc
       (sizeof(struct tstack));
node->data = number;

if(top == NULL)
    node->next = NULL;
else
    node->next = top;
top = node;
```

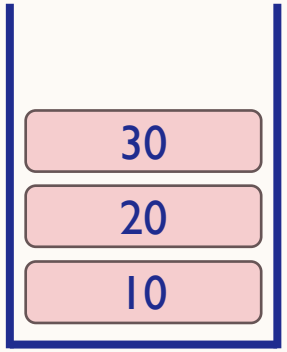




PUSH OPERATION

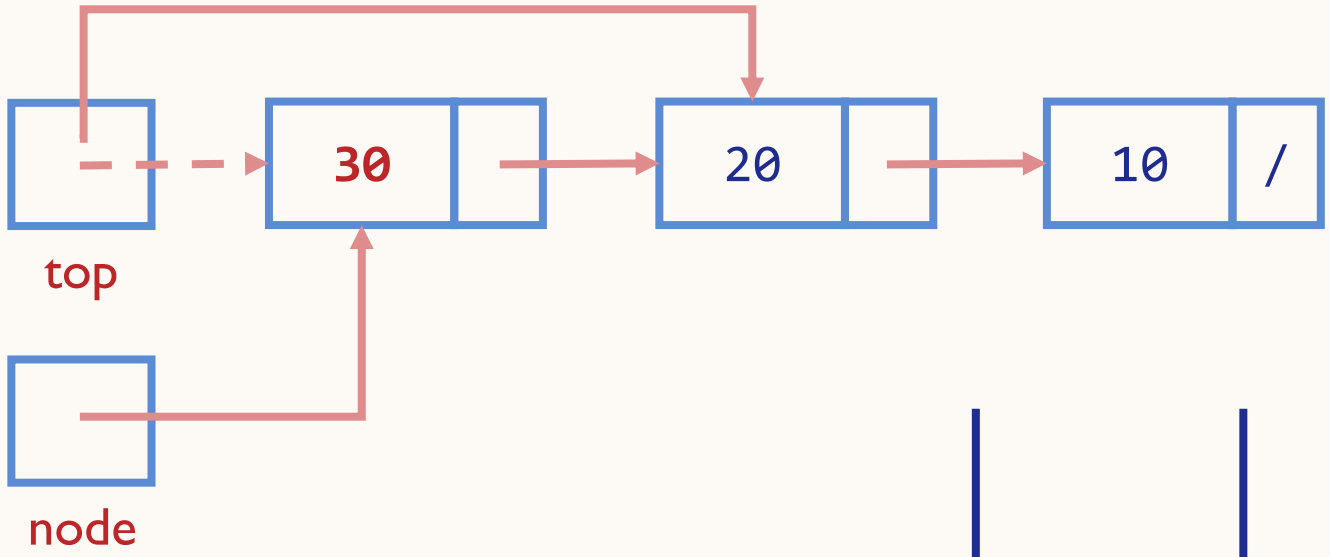
```
scanf("%d", &number); //30

node = (struct tstack *) malloc
       (sizeof(struct tstack));
node->data = number;

if(top == NULL)
    node->next = NULL;
else
    node->next = top;
top = node;
```

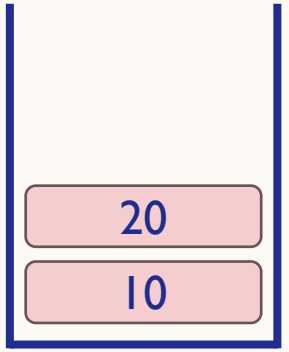




POP OPERATION

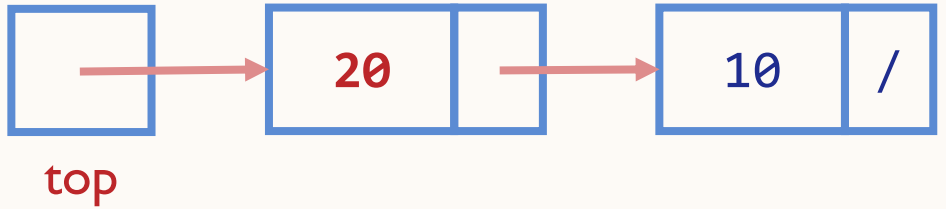
```
if(top == NULL)
    printf("UNDERFLOW");
else{
    node = top;
    top = top->next;
    //process node->data 30
    free(node);
}
```

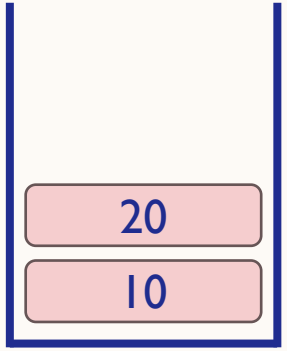




PEEK OPERATION

```
if(top == NULL)
    printf("EMPTY");
else{
    //process top->data 20
}
```





APPLICATIONS OF STACKS

- Reversing a list
- Parentheses checker
- Palindrome checker
- Conversion of an infix expression into a postfix expression
- Evaluation of a postfix expression
- Conversion of an infix expression into a prefix expression
- Evaluation of a prefix expression

CONVERSION OF AN INFIX EXPRESSION INTO A POSTFIX EXPRESSION

- Convert the following infix expressions into postfix expressions.

(A - B) * (C + D)

[AB-] * [CD+]

AB-CD+*

(A + B) / (C + D) – (D * E)

[AB+] / [CD+] – [DE*]

[AB+CD+/] – [DE*]

AB+CD+/DE*–

CONVERSION OF AN INFIX EXPRESSION INTO A POSTFIX EXPRESSION

- Algorithm to convert an infix notation to postfix notation

```
Step 1: Add “)” to the end of the infix expression
Step 2: Push “(” on the stack
Step 3: Repeat until each character in the infix notation is scanned
        IF a “(” is encountered, push it on the stack
        IF an operand (whether a digit or a character) is encountered, add it to the postfix expression
        IF a “)” is encountered, then
           a. Repeatedly pop from stack and add it to the postfix expression until a “(” is encountered
           b. Discard the “(”. That is, remove the “(” from stack and do not add it to the postfix expression
        IF an operator O is encountered, then
           a. Repeatedly pop from stack and add each operator (popped from the stack) to the postfix expression which
              has the same precedence or a higher precedence than O
           b. Push the operator O to the stack
        [END OF IF]
Step 4: Repeatedly pop from the stack and add it to the postfix expression until the stack is empty
Step 5: EXIT
```

CONVERSION OF AN INFIX EXPRESSION INTO A POSTFIX EXPRESSION

- Convert the following infix expression into postfix expression using the algorithm.

`(A - B) * (C + D)`

Infix Character Scanned	Stack	Postfix Expression
	(	
(	((	
A	((	A
-	((-	A
B	((-	A B
)	(	A B -
*	(*	A B -
(	(* (	A B -
C	(* (	A B - C
+	(* (+	A B - C
D	(* (+	A B – C D
)	(*	A B – C D +
)		**A B – C D + ***

CONVERSION OF AN INFIX EXPRESSION INTO A POSTFIX EXPRESSION

- Convert the following infix expression into postfix expression using the algorithm.

A - B * C + D

Infix Character Scanned	Stack	Postfix Expression
	(	
A	(	A
-	(-	A
B	(-	A B
*	(- *	A B
C	(- *	A B C
+	(+	A B C * -
D	(+	A B C * - D
)		**A B C * - D +**

EVALUATION OF A POSTFIX EXPRESSION

- Algorithm to evaluate a postfix expression

```
Step 1: Scan every character of the postfix expression and repeat Step 2 until the end of
        the expression
Step 2: IF an operand is encountered, push it on the stack
        IF an operator O is encountered, then
           a. Pop the top two elements from the stack as A and B
           b. Evaluate B O A, where A is the topmost element and B is the element below A
           c. Push the result of evaluation on the stack
        [END OF IF]
Step 3: SET RESULT equal to the topmost element of the stack
```

EVALUATION OF A POSTFIX EXPRESSION

- Evaluate the following postfix expression using the algorithm.

`9 3 4 * 8 + 4 / -`

Character Scanned	Stack
9	9
3	9 3
4	9 3 4
*	9 12
8	9 12 8
+	9 20
4	9 20 4
/	9 5
-	**4**

CONVERSION OF AN INFIX EXPRESSION INTO A PREFIX EXPRESSION

- Convert the following infix expressions into prefix expressions.

(A + B) * C

[+AB] * C

*+ABC

(A - B) * (C + D)

[-AB] * [+CD]

*-AB+CD

(A + B) / (C + D) – (D * E)

[+AB] / [+CD] – [*DE]

[/+AB+CD] – [*DE]

–/+AB+CD*DE

CONVERSION OF AN INFIX EXPRESSION INTO A PREFIX EXPRESSION

- Algorithm to convert an infix expression to prefix expression

```
Step 1: Reverse the infix string. Note that while reversing the string you must
        interchange left and right parentheses.
Step 2: Obtain the postfix expression of the infix expression obtained in Step 1.
Step 3: Reverse the postfix expression to get the prefix expression.
```

CONVERSION OF AN INFIX EXPRESSION INTO A PREFIX EXPRESSION

- Convert the following infix expression into prefix expression using the algorithm.

(A – B / C) * (A / K – L)

```
Step 1: (L – K / A) * (C / B – A)

Step 2: (L – [KA/]) * ([CB/] – A)

        [LKA/–] * [CB/A–]

        L K A / – C B / A – *

Step 3: * – A / B C – / A K L
```

EVALUATION OF A PREFIX EXPRESSION

- Algorithm for evaluation of a prefix expression

```
Step 1: Repeat until all the characters in the prefix expression have been scanned
          a. Scan the prefix expression from right, one character at a time
          b. IF the scanned character is an operand, push it on the operand stack
          c. IF the scanned character is an operator, then
               (i) Pop two values from the operand stack
              (ii) Apply the operator on the popped operands
             (iii) Push the result on the operand stack
```

EVALUATION OF A PREFIX EXPRESSION

- Apply the algorithm to evaluate the following prefix expression.

+ – 7 2 * 8 / 12 4

Character Scanned	Operand Stack
4	4
12	4 12
/	3
8	3 8
*	24
2	24 2
7	24 2 7
–	24 5
+	**29**

PRACTICE

EXERCISES

1. Draw the stack structure in each case when the following operations are performed on an empty stack.

    a. Add A, B, C, D, E, F
    b. Delete two letters
    c. Add G
    d. Add H
    e. Delete four letters
    f. Add I

EXERCISES

2. Convert the following infix expressions to their postfix equivalents.

   a. `A - B + C`
   b. `A * B + C / D`
   c. `(A - B) + C * D / E - C`
   d. `(A * B) + (C / D) - (D + E)`
   e. `((A - B) + D / ((E + F) * G))`
   f. `(A - 2 * (B + C) / D * E) + F`
   g. `A - (B / C + (D % E * F) / G) * H`

EXERCISES

3. Find the infix equivalents of the following postfix equivalents.

   a. A B + C * D –
   b. A B C * + D –

4. Give the infix expression of the following prefix expressions.

   a. * – + A B C D
   b. + – A * B C D

EXERCISES

5. Convert the expression given below into its corresponding postfix expression and then evaluate it.

   a. `10 + ((7 – 5) + 10) / 2`
   b. `14 / 7 * 3 – 4 + 10 / 2`

REFERENCES

- Deitel, P. and Harvey Deitel (2022), C How to Program (9th Edition), Pearson Education.
- Thareja, R. (2014), Data Structures Using C (2nd Edition), India: Oxford University Press.

NEXT

Queues:

Array Representation of Queues

Operations on Queues

Linked Representation of Queues

Operations on Linked Queues

Applications of Queues

VISION

To become an **outstanding** undergraduate Computer Science program that produces **international-minded** graduates who are **competent** in software engineering and have **entrepreneurial spirit** and **noble character**.

MISSION

1. To conduct studies with the best technology and curriculum, supported by professional lecturer
2. To conduct research in Informatics to promote science and technology
3. To deliver science-and-technology-based society services to implement science and technology

Without hard work,
nothing grows but weeds.

UMN
UNIVERSITAS MULTIMEDIA NUSANTARA

INFORMATIKA UMN

Have patience.
All things are difficult before they become easy.