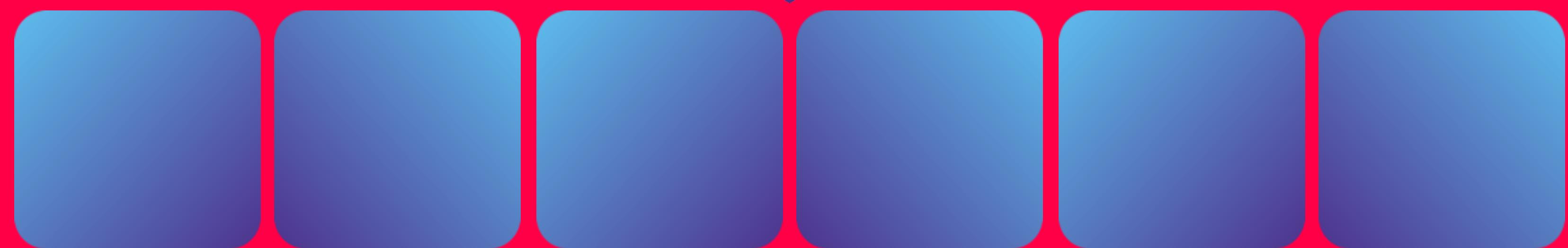


ساختمان داده:

”پشته“ و ”صف“

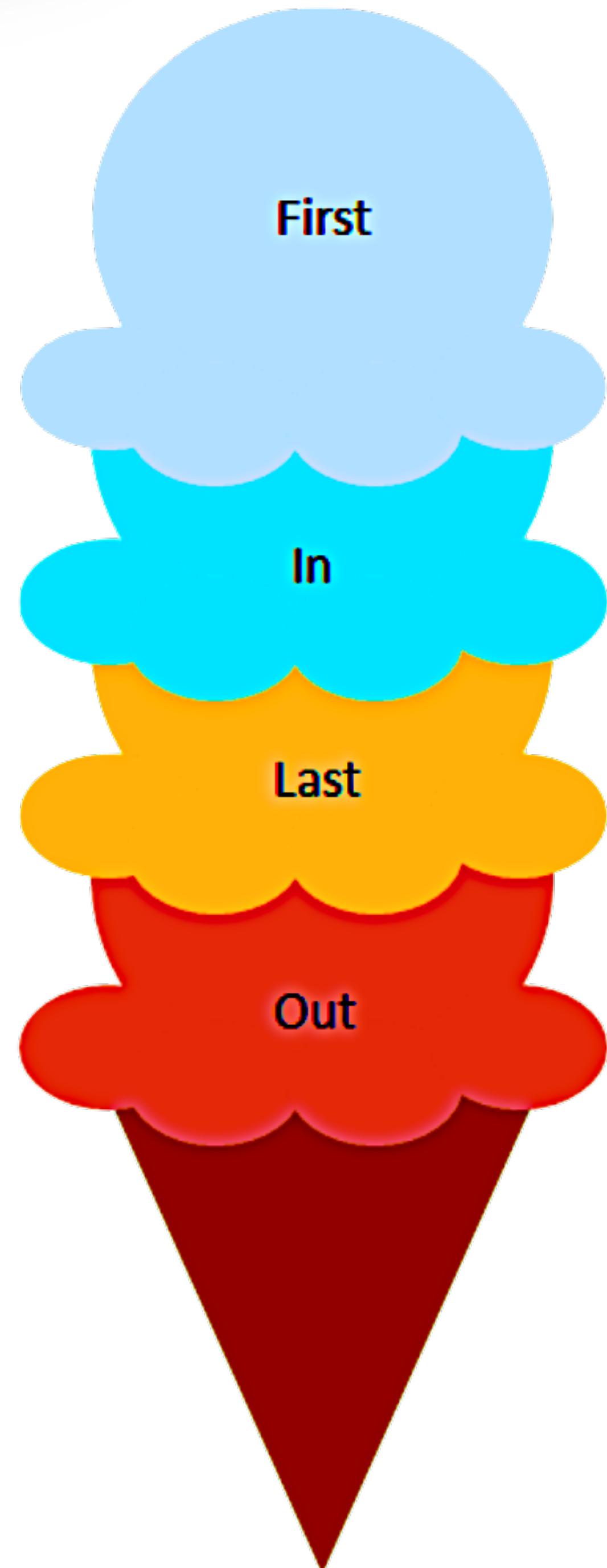


oooooooooooo>>



پشته (Stack) ساختمان داده ای است که از لیست برای سازماندهی داده ها استفاده میکند و در عین حال از انتزاع نیز پشتیبانی میکند و یک نوع داده انتزاعی را فراهم میسازد.

در پشته عمل اضافه کردن و حذف عنصر، فقط در یک طرف آن، بنام بالای پشته انعام میشود. یعنی عنصری که از همه دیرتر وارد پشته شد، از همه زودتر از پشته حذف میگردد. بهمین دلیل گفته میشود که پشته از سیاست خروج به ترتیب عکس ورود (LIFO) پیروی میکند.



FILO = FIRST INPUT LAST OUTPUT
اولین ورودی در آخرین مرتبه خروجی ظاهر میشود
LIFO = LAST INPUT FIRST OUTPUT
آخرین ورودی در اولین مرتبه خروجی ظاهر میشود

و FILO و LIFO درستک هردو به یک معنی هستند.





عملیات های قابل اجرا روی پشته:

: که عنصری را به بالای پشته اضافه میکند. **Push()**

: که عنصر بالای پشته را حذف میکند. **Pop()**

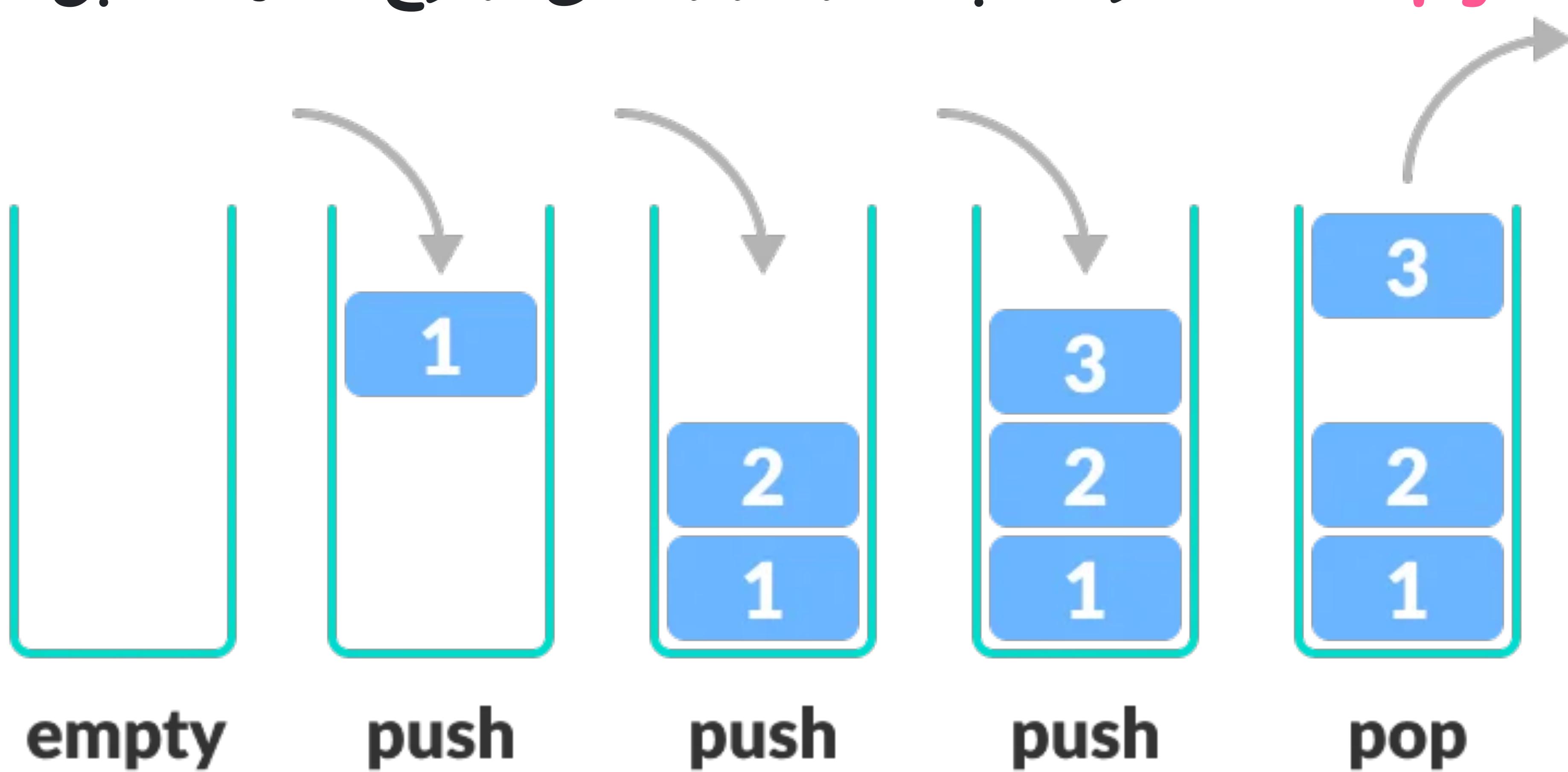
: که عنصر بالای پشته را بازیابی میکند ولی حذف نمیکند. **Peek()**

: که خالی بودن پشته را تست میکند. **StackEmpty()**

: تمام عناصر پشته را حذف میکند. **Clear()**

: مشخص میکند که عنصری در پشته وجود دارد یا خیر. **Contains()**

: محتویات پشته را در آرایه ای از نوع Object کپی میکند. **CopyTo()**



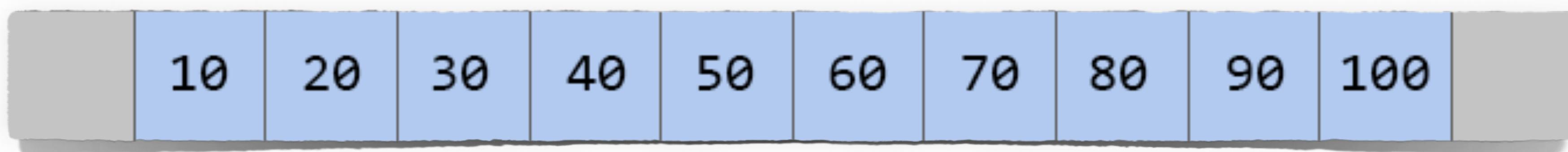


پیاده سازی:

دو راه برای پیاده سازی یک پشته وجود دارد:

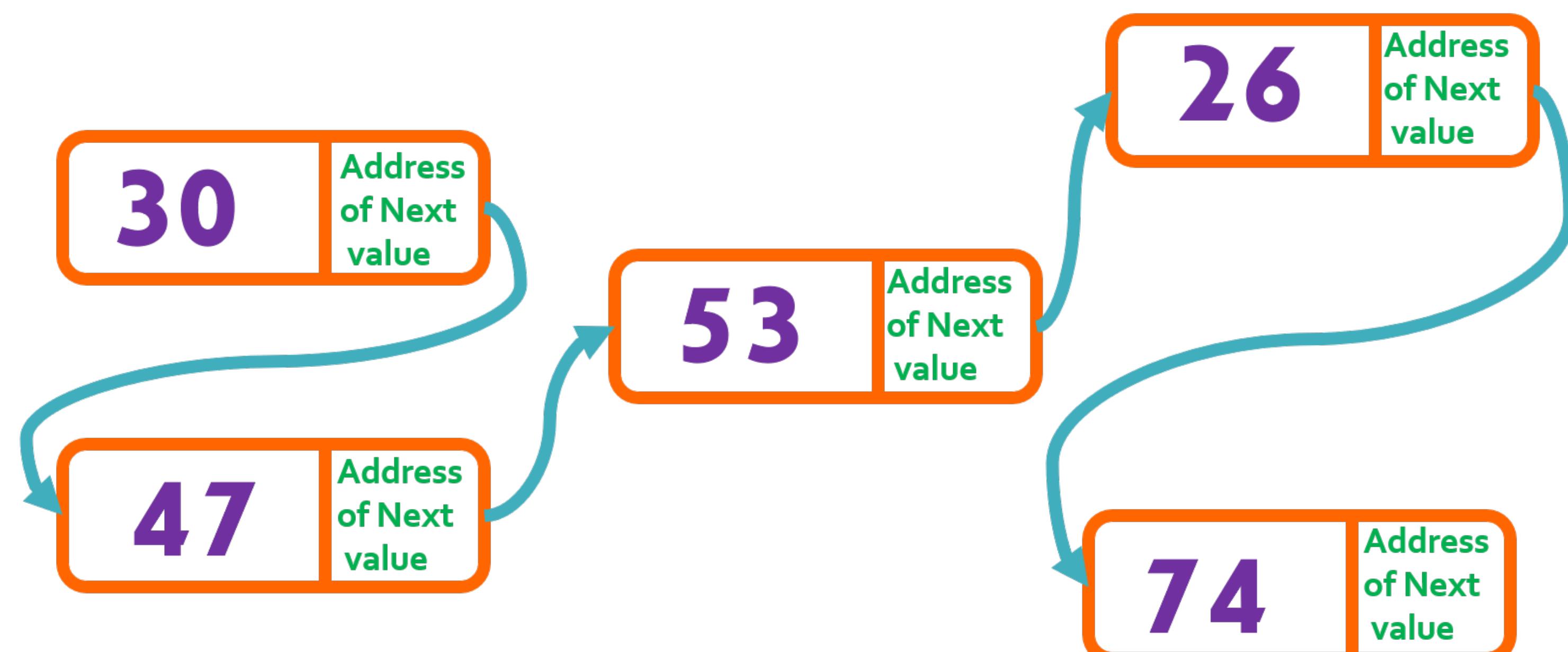
استفاده از آرایه

(به پست مربوط به آرایه مراجعه شود)



استفاده از لیست پیوندی

(به پست مربوط به لیست پیوندی مراجعه شود)





C++

```
1 // Stack_Built_With_Array
2 //
3 // Created by Shayan Aryania
4 //
5
6 #include <iostream>
7 using namespace std;
8
9 #define MAX 1000
10
11 class Stack {
12     int top;
13
14 public:
15     int a[MAX]; // Maximum size of
16     Stack
17     Stack() { top = -1; }
18     bool push(int x);
19     int pop();
20     int peek();
21     bool isEmpty();
22 };
23
24 bool Stack::push(int x)
25 {
26     if (top >= (MAX - 1)) {
27         cout << "Stack Overflow";
28         return false;
29     }
30     else {
31         a[++top] = x;
32         cout << x << " pushed into
33             stack\n";
34         return true;
35     }
36 }
37 int Stack::pop()
38 {
39     if (top < 0) {
40         cout << "Stack Underflow";
41         return 0;
42     }
43     else {
44         int x = a[top--];
45         return x;
46     }
47 }
48 int Stack::peek()
49 {
50     if (top < 0) {
51         cout << "Stack is Empty";
52         return 0;
53     }
54     else {
55         int x = a[top];
56         return x;
57     }
58 }
59
60 bool Stack::isEmpty()
61 {
62     return (top < 0);
63 }
64
65 // Driver program to test above
66 // functions
67 int main()
68 {
69     class Stack s;
70     s.push(10);
71     s.push(20);
72     s.push(30);
73     cout << s.pop() << " Popped
74         from stack\n";
75     //print all elements in stack :
76     cout<<"Elements present in
77         stack : ";
78     while(!s.isEmpty())
79     {
79         // print top element in
80         // stack
81         cout<<s.peek()<<" ";
82         // remove top element in
83         // stack
84         s.pop();
85     }
86     return 0;
87 }
```

Line: 30 Col: 11

```
10 pushed into stack
20 pushed into stack
30 pushed into stack
30 Popped from stack
Elements present in stack : 20 10 Program ended with exit code: 0
```

برای رانلور فایل کام کردها به لینک
گستاخ است.

Python

The screenshot shows a split-screen view of a Python script in VS Code. The left pane contains the following code:

```
#Shayan Aryania
# python program for "Stack_Built_With_Array"
from sys import maxsize

# Function to create a stack. It initializes size of
def createStack():
    stack = []
    return stack

# Stack is empty when stack size is 0
def isEmpty(stack):
    return len(stack) == 0

# Function to add an item to stack. It increases size by 1
def push(stack, item):
    stack.append(item)
    print(item + " pushed to stack ")

10 pushed to stack
20 pushed to stack
30 pushed to stack
```

The right pane contains the following code:

```
# Function to remove an item from stack. It decreases size by 1
def pop(stack):
    if (isEmpty(stack)):
        return str(-maxsize -1) # return minus infinity
    return stack.pop()

# Function to return the top from stack without removing it
def peek(stack):
    if (isEmpty(stack)):
        return str(-maxsize -1) # return minus infinity
    return stack[len(stack) - 1]

# Driver program to test above functions
stack = createStack()
push(stack, str(10))
push(stack, str(20))
push(stack, str(30))
print(pop(stack) + " popped from stack")
```

The terminal at the bottom shows the output of the driver program:

```
zsh
Python De...
code/extensions/ms-python.python-2022.0.1814523869/pythonFiles/lib/python/debugpy/laun
cher 49424 -- /Users/shayan/Desktop/Stack_Built_With_Array.py
10 pushed to stack
20 pushed to stack
30 pushed to stack
30 popped from stack
shayan@Shayans-MBP Desktop %
```



C++



```
Stack_Built_With_LinkedList
main.cpp
Stack_Built_With_LinkedList
Stack_Built_With_LinkedList
Stack_Built_With_LinkedList

1 // Stack_Built_With_LinkedList
2 // Created by Shayan Aryania
3
4 #include <iostream>
5 using namespace std;
6
7 // A structure to represent a stack
8 class StackNode {
9 public:
10     int data;
11     StackNode* next;
12 };
13
14 StackNode* newNode(int data)
15 {
16     StackNode* stackNode = new
17         StackNode();
18     stackNode->data = data;
19     stackNode->next = NULL;
20     return stackNode;
21 }
22
23 int isEmpty(StackNode* root)
24 {
25     return !root;
26 }
27
28 void push(StackNode** root, int data)
29 {
30     StackNode* stackNode =
31         newNode(data);
32     stackNode->next = *root;
33     *root = stackNode;
34     cout << data << " pushed to
35     stack\n";
36 }
37
38 int pop(StackNode** root)
39 {
40     if (isEmpty(*root))
41         return INT_MIN;
42     StackNode* temp = *root;
43     *root = (*root)->next;
44     int popped = temp->data;
45     free(temp);
46     return popped;
47 }
48
49 int peek(StackNode* root)
50 {
51     if (isEmpty(root))
52         return INT_MIN;
53     return root->data;
54 }
55
56 // Driver code
57 int main()
58 {
59     StackNode* root = NULL;
60     push(&root, 10);
61     push(&root, 20);
62     push(&root, 30);
63
64     cout << pop(&root) << " popped
65     from stack\n";
66
67     cout << "Top element is " <<
68     peek(root) << endl;
69
70     cout << "Elements present in stack
71     : ";
72
73     // print all elements in stack :
74     while(!isEmpty(root))
75     {
76         // print top element in stack
77         cout << peek(root) << " ";
78         // remove top element in
79         stack
80         pop(&root);
81     }
82
83     return 0;
84 }
```

10 pushed to stack
20 pushed to stack
30 pushed to stack
30 popped from stack
Top element is 20
Elements present in stack : 20 10 Program ended with exit code: 0

Python



```
Stack_Built_With_LinkedList.py
Stack_Built_With_LinkedList.py
Stack_Built_With_LinkedList.py

1 #Shayan Aryania
2 # python program for "Stack_Built_With_LinkedList"
3
4 # Class to represent a node
5 class StackNode:
6
7     # Constructor to initialize a node
8     def __init__(self, data):
9         self.data = data
10        self.next = None
11
12
13
14 class Stack:
15
16     # Constructor to initialize the root
17     def __init__(self):
18         self.root = None
19
20     def isEmpty(self):
21         return True if self.root is None else False
22
23
24     def push(self, data):
25         newNode = StackNode(data)
26         newNode.next = self.root
27         self.root = newNode
28         print ("%d pushed to stack" % data)
29
30
31     def pop(self):
32         if (self.isEmpty()):
33             return float("-inf")
34         temp = self.root
35         self.root = self.root.next
36         popped = temp.data
37         return popped
38
39     def peek(self):
40         if (self.isEmpty()):
41             return float("-inf")
42         return self.root.data
43
44 # Driver code
45 stack = Stack()
46 stack.push(10)
47 stack.push(20)
48 stack.push(30)
49
50 print ("%d popped from stack" % (stack.pop()))
51 print ("Top element is %d" % (stack.peek()))
```

With_LinkedList.py
10 pushed to stack
20 pushed to stack
30 pushed to stack
30 popped from stack
Top element is 20
shayan@Shayan-MBP Desktop %

برای دانلود فایل کامل کد های لینک
در هایلایت درج شده مراجعه کنید.
لینک دانلود کلیک کنید.

000+000000>>

استفاده از لیست پیوندی

برای سازی



@ShayanAryania



صف

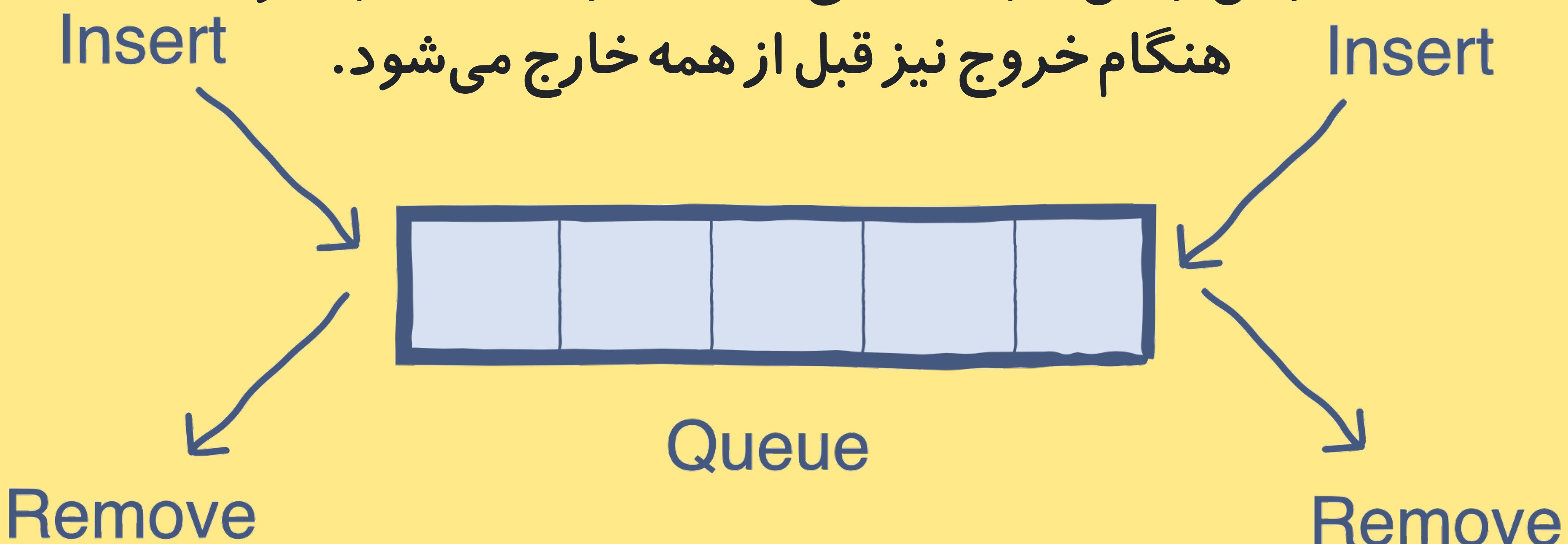
(QUEUE)

صف یک ساختار داده است که تا حدودی شبیه پشته محسوب می شود؛
اما برخلاف پشته، صف از هر دو سمت باز است.

از یک سمت همواره برای **درج دادهها** و از سمت دیگر برای **حذف دادهها** استفاده می شود.
صف از روش FIFO (ورودی اول-خروجی اول) استفاده می کند.

در این روش هر داده های که اول در صف ذخیره شود،

هنگام خروج نیز قبل از همه خارج می شود.



FIFO = FIRST INPUT FIRST OUTPUT

اولین ورودی در اولین مرتبه خروجی ظاهر می شود

LIFO = LAST INPUT FIRST OUTPUT

آخرین ورودی در اولین مرتبه خروجی ظاهر می شود

FIFO و LIFO در صف هردو به یک معنی هستند.

صف از هر دو سمت باز



عملیات های قابل اجرا روی پشته:

صفها دو اشاره‌گر داده رانگه داری می‌کنند که Front و Rear نام دارند.
از این رو پیاده‌سازی عملیات‌های آن‌ها نسبت به پشته‌ها کاملاً پیچیده‌تر است.

عملیات :Dequeue

گام ۱: بررسی کن که صف خالی است یا نه.

(`isempty()`, `isfull()`)

گام ۲: اگر صف خالی بود خطای Overflow را صادر کرده و خارج شو.

گام ۳: اگر صف خالی نبود، به داده‌ای که اشاره‌گر Front نشان می‌دهد، دسترسی ایجاد کن.

گام ۴: اشاره‌گر Front را یک واحد افزایش بده تا به موقعیت بعدی اشاره کند.

گام ۵: پیام موفقیت را بازگردان.

عملیات :Enqueue

گام ۱: بررسی کن که آیا صف پر است یا نه.

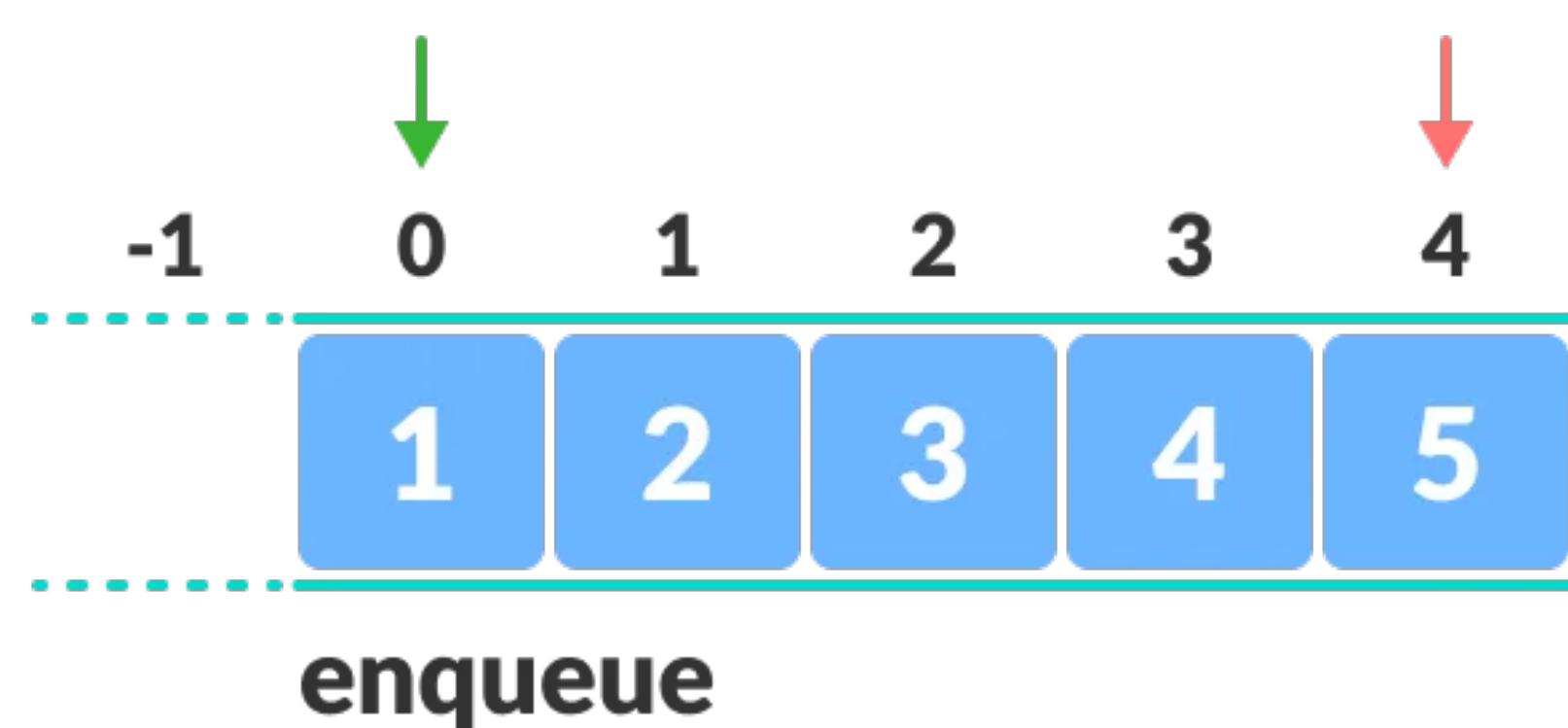
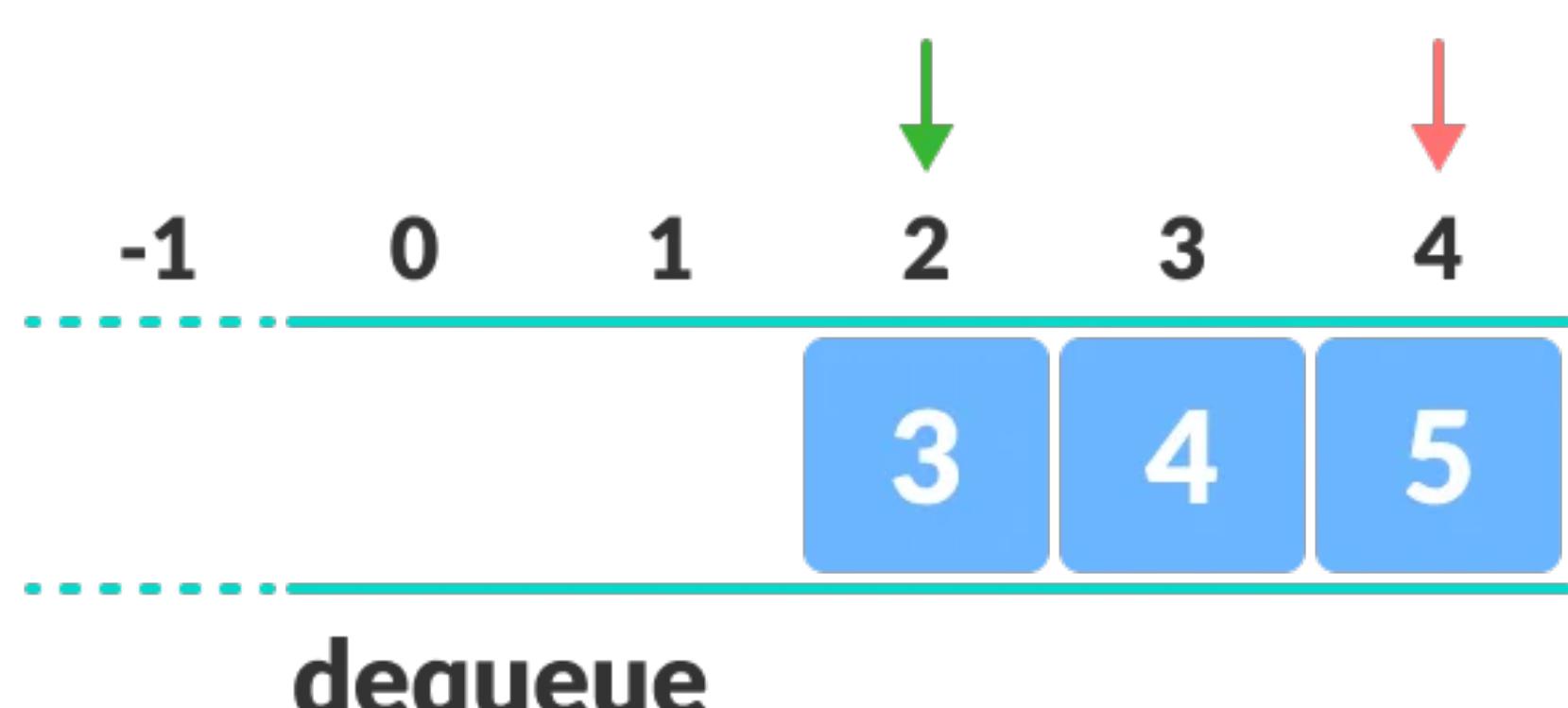
(`isempty()`, `isfull()`)

گام ۲: اگر صف پر بود، خطای Overflow را صادر کرده و خارج شو.

گام ۳: اگر صف پر نبود، مقدار اشاره‌گر را یک واحد افزایش بده تا به فضای خالی بعدی اشاره کند.

گام ۴: عنصر داده‌ای را به موقعیت صف که اشاره‌گر Rear نشان می‌دهد، اضافه کن.

گام ۵: پیام موفقیت را بازگردان.



000000 + 000 >>

C++



Queue main Queue > C' main > createQueue(capacity)

```

1 // Queue
2 //
3 // Created by Shayan Aryania
4 //
5
6 #include <iostream>
7 using namespace std;
8
9 // A structure to represent a
10 class Queue {
11 public:
12     int front, rear, size;
13     unsigned capacity;
14     int* array;
15 };
16
17 // function to create a queue
18 // of given capacity.
19 // It initializes size of queue
20 // as 0
21 Queue* createQueue(unsigned
22 capacity)
23 {
24     Queue* queue = new Queue();
25     queue->capacity = capacity;
26     queue->front = queue->size =
27         0;
28     // This is important, see
29     // the enqueue
30     queue->rear = capacity - 1;
31     queue->array = new
32         int[queue->capacity];
33     return queue;
34 }

```

Queue main Queue > C' main > No Selection

```

29     return queue;
30 }
31
32 // Queue is full when size
33 // becomes equal to the capacity
34 int isFull(Queue* queue)
35 {
36     return (queue->size ==
37         queue->capacity);
38
39 // Queue is empty when size is 0
40 int isEmpty(Queue* queue)
41 {
42     return (queue->size == 0);
43 }
44
45 // Function to add an item to
46 // the queue.
47 // It changes rear and size
48 void enqueue(Queue* queue, int
49 item)
50 {
51     if (isFull(queue))
52         return;
53     queue->rear = (queue->rear
54         + 1)
55         %
56         queue->capac
57     it
58     queue->array[queue->rear] =
59         item;
60     queue->size = queue->size +
61         1;
62     cout << item << " enqueued
63         to queue\n";
64 }

```

Queue main Queue > C' main > No Selection

```

57
58 // Function to remove an item from queue.
59 // It changes front and size
60 int dequeue(Queue* queue)
61 {
62     if (isEmpty(queue))
63         return INT_MIN;
64     int item =
65         queue->array[queue->front];
66     queue->front = (queue->front + 1)
67         %
68         queue->capacity;
69     queue->size = queue->size - 1;
70     return item;
71 }
72
73 // Function to get front of queue
74 int front(Queue* queue)
75 {
76     if (isEmpty(queue))
77         return INT_MIN;
78     return queue->array[queue->front];
79 }
80
81 // Function to get rear of queue
82 int rear(Queue* queue)
83 {
84     if (isEmpty(queue))
85         return INT_MIN;
86     return queue->array[queue->rear];
87 }
88
89
90
91
92
93
94 // Driver code
95 int main()
96 {
97     Queue* queue =
98         createQueue
99         (1000);
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114

```

Line: 93 Col: 1

All Output

```

10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
40 enqueued to queue
10 dequeued from queue
Front item is 20
Rear item is 40

```

Python



Queue.py - Desktop

```

1 #Shayan Aryania
2 # python program for "Queue"
3
4 class Queue:
5
6     # __init__ function
7     def __init__(self, capacity):
8         self.front = self.size = 0
9         self.rear = capacity - 1
10        self.Q = [None]*capacity
11        self.capacity = capacity
12
13    # Queue is full when size becomes
14    # equal to the capacity
15    def isFull(self):
16        return self.size == self.capacity
17
18    # Queue is empty when size is 0
19    def isEmpty(self):
20        return self.size == 0
21
22    # Function to add an item to the
23    # queue. It changes rear and size
24    def EnQueue(self, item):
25        if self.isFull():
26            print("Full")
27            return
28        self.rear = (self.rear + 1) % (
29            self.capacity)
30        self.Q[self.rear] = item
31        self.size = self.size + 1
32        print("% s enqueued to queue" %
33            item)
34
35    # Function to remove an item from
36    # queue. It changes front and size
37    def DeQueue(self):
38        if self.isEmpty():
39            print("Empty")
40            return
41        print("% s dequeued from queue" %
42            self.Q[self.front])
43        self.front = (self.front + 1) %
44            self.capacity
45        self.size = self.size - 1
46
47    # Function to get front or queue
48    def que_front(self):
49        if self.isEmpty():
50            print("Queue is empty")
51        else:
52            print("Front item is", self.Q[self.front])
53
54    # Function to get rear of queue
55    def que_rear(self):
56        if self.isEmpty():
57            print("Queue is empty")
58        else:
59            print("Rear item is", self.Q[self.rear])
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```

10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
40 enqueued to queue
10 dequeued from queue
Front item is 20
Rear item is 40

```

Ln 2, Col 28 Spaces: 4 UTF-8 LF Python 3.10.1 64-bit

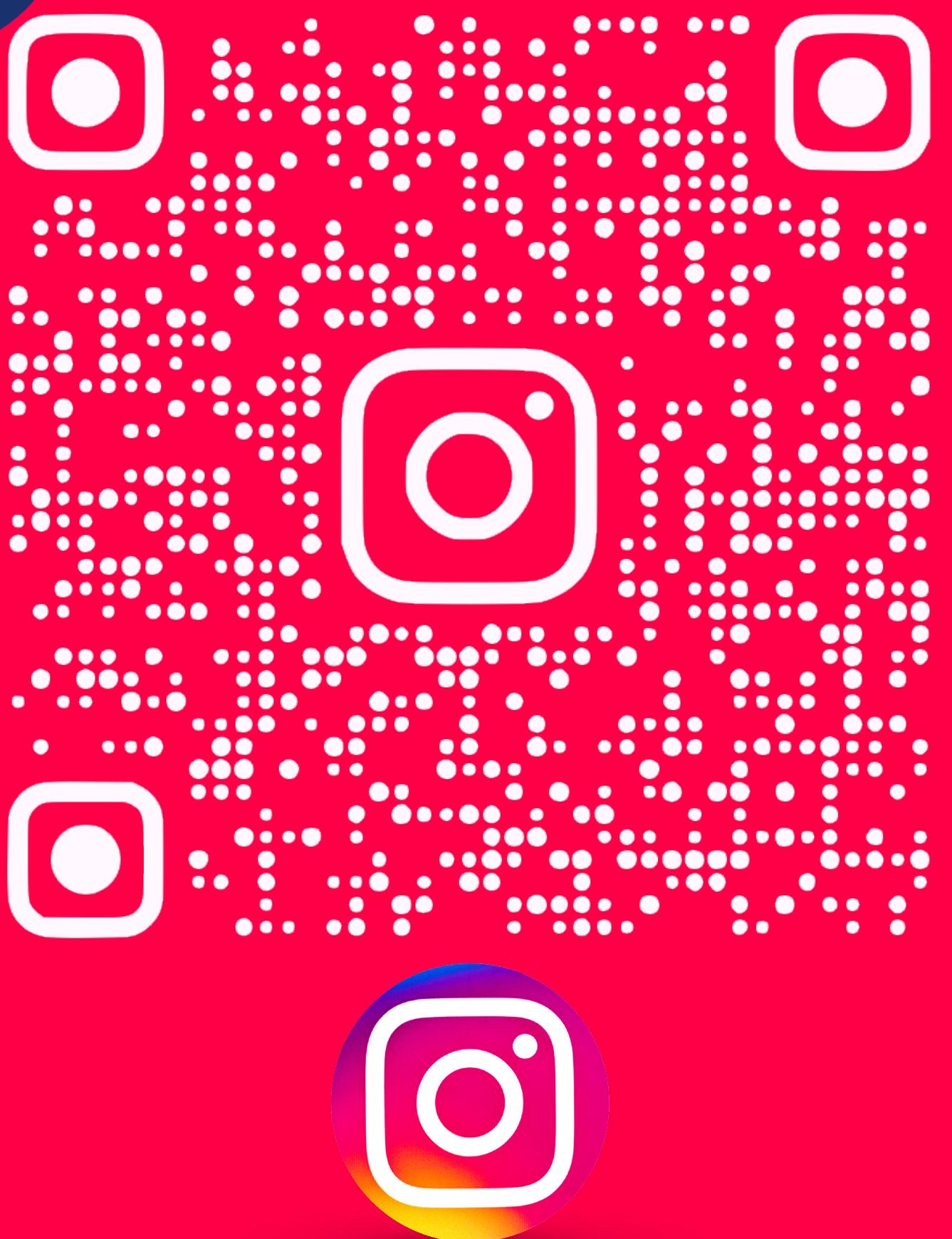
برای دانلود فایل کامل کدها به لینک
کلیک کنید.
و در هایلایت در جستجو مراجعه کنید.

000000000>>

@ShayanAryania



Follow



www.instagram.com/shayanaryania



www.linkedin.com/in/shayanaryania



Like & Share

Save



عملیات‌های ابتدایی:

عملیات‌های صف می‌تواند شامل راه‌اندازی یا تعریف، استفاده و سپس حذف کامل آن از حافظه باشد.

: یک آیتم را به صف اضافه (ذخیره) می‌کند. **.Enqueue**

: یک آیتم را از صف حذف می‌کند (مورد دسترسی قرار می‌دهد). **Dequeue**

چند کارکرد دیگر نیز هستند که برای کارآمدسازی عملیات‌های فوق در صف ضروری هستند:
: یک عنصر را بدون حذف کردن، از ابتدای صف دریافت می‌کند. **peek**

: پر بودن صف را بررسی می‌کند. **isfull**

: خالی بودن صف را بررسی می‌کند. **isempty**

