

**Name : Tamanna**

**UID : 22BCS13649**

**Section : 22BCS\_IOT\_615-B**

**Assignment : 4**

### **STACK AND QUEUE STANDARD QUESTION**

#### **VERY EASY:**

**1 Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.**

**Implement the MinStack class:**

- MinStack() initializes the stack object.
- void push(int val) pushes the element val onto the stack.
- void pop() removes the element on the top of the stack.
- int top() gets the top element of the stack.
- int getMin() retrieves the minimum element in the stack.

You must implement a solution with O(1) time complexity for each function.

**Example 1:**

**Input**

["MinStack","push","push","push","getMin","pop","top","getMin"]

[[],[-2],[0],[-3],[],[],[],[ ]]

**Output**

[null,null,null,null,-3,null,0,-2]

**Explanation**

```
MinStack minStack = new MinStack();
```

```
minStack.push(-2); minStack.push(0);
```

```
minStack.push(-3);
```

```
minStack.getMin(); // return -3
```

```
minStack.pop();
```

```
minStack.top();    // return 0 minStack.getMin();
```

```
// return -2
```

### Example 2:

#### Input:

```
["MinStack", "push", "push", "push", "push", "getMin", "pop", "getMin", "top", "getMin"]
```

```
[[], [5], [3], [7], [3], [], [], [], [], []]
```

#### Output

```
[null, null, null, null, null, 3, null, 3, 7, 3]
```

#### Explanation:

```
MinStack minStack = new MinStack(); minStack.push(5); #
Stack: [5], MinStack: [5] minStack.push(3); # Stack: [5, 3],
MinStack: [5, 3] minStack.push(7); # Stack: [5, 3, 7], MinStack:
[5, 3] minStack.push(3); # Stack: [5, 3, 7, 3], MinStack: [5, 3, 3]
minStack.getMin(); # Returns 3 minStack.pop(); # Removes
3; Stack: [5, 3, 7], MinStack: [5, 3] minStack.getMin(); #
Returns 3 minStack.top(); # Returns 7 minStack.getMin(); #
Returns 3
```

- Minimum values are maintained as:  $[5] \rightarrow [5, 3] \rightarrow [5, 3] \rightarrow [5, 3]$  •  
After pops, the minimum values update accordingly.

### Example 3:

#### Input:

```
["MinStack", "push", "push", "push", "getMin", "pop", "getMin", "pop", "getMin"]
```

```
[[], [2], [1], [4], [], [], [], [], []]
```

#### Output:

```
[null, null, null, null, 1, null, 1, null, 2]
```

#### Explanation:

```
minStack = MinStack()
minStack.push(2)
minStack.push(1)
minStack.push(4)
minStack.push(1)
print(minStack.getMin()) # Output: 1 minStack.pop()
print(minStack.getMin()) # Output: 1 minStack.pop()
print(minStack.getMin()) # Output: 1 minStack.pop()
```

`print(minStack.getMin())` # Output: 2

- Minimum values are maintained as:  $[2] \rightarrow [2, 1] \rightarrow [2, 1] \rightarrow [2, 1]$  • After pops, the minimum values update accordingly.

### Constraints:

- $-2^{31} \leq \text{val} \leq 2^{31} - 1$
- Methods `pop`, `top` and `getMin` operations will always be called on non-empty stacks.
- At most  $3 * 10^4$  calls will be made to `push`, `pop`, `top`, and `getMin`.

Ø Sources: <https://leetcode.com/problems/min-stack/>

### Code

```
#include <iostream>
```

```
#include <stack>
```

```
#include <climits>
```

```
class MinStack { private:
```

```
    std::stack<int> stack;
```

```
    std::stack<int> minStack; public:
```

```
    MinStack() {}
```

```
    void push(int val) {        stack.push(val);        if  
(minStack.empty() || val <= minStack.top()) {  
        minStack.push(val);  
    }  
}
```

```
void pop() {      if (stack.top() ==  
minStack.top()) {      minStack.pop();  
    }  
stack.pop();  
}
```

```
int top() {  
return stack.top();  
}
```

```
int getMin() {  
return minStack.top();  
}  
};
```

```
int main() {
```

```
    MinStack minStack;
```

```
    minStack.push(-2);    minStack.push(0);
```

```
minStack.push(-3);    std::cout <<
```

```
minStack.getMin() << std::endl;
```

```
minStack.pop();    std::cout << minStack.top()
```

```
<< std::endl;    std::cout <<
```

```
minStack.getMin() << std::endl;
```

```

    minStack.push(5);    minStack.push(3);

minStack.push(7);    minStack.push(3);

std::cout << minStack.getMin() << std::endl;

minStack.pop();    std::cout <<

minStack.getMin() << std::endl;    std::cout

<< minStack.top() << std::endl;    std::cout <<

minStack.getMin() << std::endl;

return 0; }

```

## Output

```

1  #include <iostream>
2  #include <stack>
3  #include <climits>
4
5  class MinStack {
6  private:
7      std::stack<int> stack;
8      std::stack<int> minStack;
9
10 public:
11     MinStack() {}
12
13     void push(int val) {
14         stack.push(val);
15         if (minStack.empty() || val <= minStack.top()) {
16             minStack.push(val);
17         }
18     }
19
20     void pop() {
21         if (stack.top() == minStack.top()) {
22             minStack.pop();
23         }
24         stack.pop();
25     }
26
27     int getMin() {
28         return minStack.top();
29     }
30 }
31
32 int main() {
33     MinStack minStack;
34     minStack.push(3);
35     minStack.push(0);
36     minStack.push(-2);
37     minStack.push(-2);
38     minStack.push(-2);
39     minStack.push(7);
40     minStack.push(-2);
41     minStack.pop();
42     minStack.pop();
43     minStack.pop();
44     minStack.push(5);
45     minStack.push(3);
46     minStack.push(7);
47     minStack.push(3);
48     std::cout << minStack.getMin() << std::endl;
49     minStack.pop();
50     std::cout <<
51     minStack.getMin() << std::endl;
52     std::cout
53     << minStack.top() << std::endl;
54     std::cout <<
55     minStack.getMin() << std::endl;
56     return 0; }

```

input

```

3
0
-2
-2
-2
7
-2

```

**2 Given a string s, find the first non-repeating character in it and return its index. If it does not exist, return -1.**

**Example 1:**

**Input:** s = "leetcode"

**Output:** 0

**Explanation:**

The character 'l' at index 0 is the first character that does not occur at any other index.

**Example 2:**

**Input:** s = "loveleetcode"

**Output:** 2

**Example 3:**

**Input:** s = "aabb"

**Output:** -1

**Constraints:**

- $1 \leq s.length \leq 105$
- s consists of only lowercase English letters.

**Approach:**

Use a hash map or array of size 26 to store the frequency of each character (since the input consists of lowercase English letters).

Traverse the string to count the frequency of each character.

Traverse the string again to find the first character with a frequency of 1. Return its index.

If no such character is found, return -1.

**Time Complexity:** ( $O(n)$ ), where ( $n$ ) is the length of the string. The string is traversed twice.

**Space Complexity:** ( $O(1)$ ), as the frequency array has a fixed size of 26.

Reference : <https://leetcode.com/problems/first-unique-character-in-a-string/description/>

**Code**

```
#include <iostream>
#include <string>
#include <vector>

int firstUniqChar(const std::string& s) {
    std::vector<int> frequency(26, 0);
```

```

    for (char c : s) {
        frequency[c - 'a']++;
    }
    for (int i = 0; i < s.length(); i++) {
        if (frequency[s[i] - 'a'] == 1) {
            return i;
        }
    }
    return -1;
}

int main() {
    std::string s1 = "leetcode";
    std::string s2 = "loveleetcode";
    std::string s3 = "aabb";

    std::cout << firstUniqChar(s1) << std::endl;
    std::cout << firstUniqChar(s2) << std::endl;    std::cout
    << firstUniqChar(s3) << std::endl;

    return 0;
}

```

## Output

The screenshot shows a C++ program in a code editor with the following code:

```

1 #include <iostream>
2 #include <string>
3 #include <vector>
4
5 int firstUniqChar(const std::string& s) {
6     std::vector<int> frequency(26, 0);
7     for (char c : s) {
8         frequency[c - 'a']++;
9     }
10    for (int i = 0; i < s.length(); i++) {
11        if (frequency[s[i] - 'a'] == 1) {
12            return i;
13        }
14    }
15    return -1;
16 }
17
18 int main() {
19     std::string s1 = "leetcode";
20     std::string s2 = "loveleetcode";
21     std::string s3 = "aabb";
22
23     std::cout << firstUniqChar(s1) << std::endl;
24     std::cout << firstUniqChar(s2) << std::endl;
25     std::cout << firstUniqChar(s3) << std::endl;
26 }

```

The output of the program is displayed in the console:

```

0
2
-1
...Program finished with exit code 0
Press ENTER to exit console.

```

**3 Implement a simple text editor. The editor initially contains an empty string, S. Perform Q operations of the following 4 types:**

➤ append(W) - Append string W to the end of S.

- delete (k)- Delete the last k characters of S.
- print (k)- Print the k<sup>th</sup> character of S.
- undo() - Undo the last (not previously undone) operation of type 1 or 2, reverting S to the state it was in prior to that operation.

### Example 1

S = 'abcde'

Ops=['1 fg', '3 6', '2 5', '4', '3 7', '4', '3 4']

**operation index S ops[index]**

#### explanation

-----0

abcde 1 fg append fg

- |   |             |                                  |
|---|-------------|----------------------------------|
| 1 | abcdefg 3 6 | print the 6th letter - f         |
| 2 | abcdefg 2 5 | delete the last 5 letters        |
| 3 | ab 4        | undo the last operation, index 2 |
| 4 | abcdefg 3 7 | print the 7th character - g      |
| 5 | abcdefg 4   | undo the last operation, index 0 |
| 6 | abcde 3 4   | print the 4th character - d      |

**The results should be printed as:**

f

g d

### Input Format

The first line contains an integer, Q, denoting the number of operations.

Each line i of the Q subsequent lines (where  $0 \leq i < Q$ ) defines an operation to be performed. Each operation starts with a single integer, t (where  $t \in \{1, 2, 3, 4\}$ ), denoting a type of operation as defined in the Problem Statement above. If the operation requires an argument, t is followed by its space-separated argument. For example, if  $t=1$  and  $W="abcd"$ , line i will be 1 abcd.

Example 2 (Custom Test Case)



## Input

9

```
1 hello
1 world
3 10
2 5
3 5
4
3 10
4
3 1
```

## Code Execution

```
operations = [
    "1 hello", "1 world", "3 10", "2 5", "3 5",
    "4", "3 10", "4", "3 1"
]
result = text_editor(operations)
print("\n".join(result)) # Outputs: d, o, d, h Output
```

```
d o
d h
```

## Example 3 (Custom Test Case)

## Input

```
10
1 programming
3 1
2 6
3 4
1 code
3 4
4
3 7
4
3 8
```

## Code Execution

```
operations = [
```

```

    "1 programming", "3 1", "2 6", "3 4", "1 code",
    "3 4", "4", "3 7", "4", "3 8"
]
result = text_editor(operations)
print("\n".join(result)) # Outputs: p, g, o, m

```

## Output

```

p g
o
m

```

## Constraints

- $1 \leq Q \leq 10^6$
- $1 \leq k \leq |S|$
- The sum of the lengths of all in the input  $\leq 10^6$ .
- The sum of over all delete operations  $\leq 2 \cdot 10^6$ .
- All input characters are lowercase English letters.
- It is guaranteed that the sequence of operations given as input is possible to perform.

## Output Format

Each operation of type 3 must print the  $k^{\text{th}}$  character on a new line.

## Sample Input

### STDIN Function

```
-----
```

```
8    Q = 8
```

```
1 abc ops[0] = '1 abc'
```

```
3 3 ops[1] = '3 3'
```

```
2 3 ...
```

```
1 xy
```

```
3 2
```

```
4
```

4

3 1

### Sample Output

c y

a

### Explanation

- 1 Initially, S is empty. The following sequence of 8 operations are described below:
- 2 S="". We append abc to S , so S = "abc".
- 3 Print the 3<sup>rd</sup> character on a new line. Currently, the 3<sup>rd</sup> character is c.
- 4 Delete the last 3 characters in S(abc), so S= "".
- 5 Append xy to S , so S= "xy".
- 6 Print the 2<sup>nd</sup> character on a new line. Currently, the 2<sup>nd</sup> character is y.
- 7 Undo the last update to S, making S empty again (i.e.,S="").
- 8 Undo the next to last update to S (the deletion of the last 3 characters), making S="abc".
- 9 Print the 1<sup>st</sup> character on a new line. Currently, the 1<sup>st</sup> character is a.

➤ **Frequently Asked By:** Facebook, Atlassian, Adobe(**Year:** 2018–2022)

➤ Reference: <https://www.hackerrank.com/challenges/simple-text-editor>

### Code

```
#include <iostream>
```

```
#include <stack>
```

```
#include <string>
```

```
#include <vector>
```

```
int main() { int
```

```
Q; std::cin >>
```

```
Q;
```

```

    std::string S;          std::stack<std::pair<int,
std::string>>> history;    std::vector<char>
output;

```

```

    while (Q--) {

        int t;
std::cin >> t;

        if (t == 1) {
std::string W;
std::cin >> W;
history.push({1,
W});          S +=
W;          } else if (t
== 2) {          int k;
std::cin >> k;
history.push({2,
S.substr(S.size() -
k)});

        S.erase(S.size() - k);      }

else if (t == 3) {          int k;
std::cin >> k;
output.push_back(S[k - 1]);      }

else if (t == 4) {          if
(!history.empty()) {          auto
last = history.top();

```

```
history.pop();          if (last.first
== 1) {

    S.erase(S.size() - last.second.size());

    } else if (last.first == 2) {

        S += last.second;

    }

    }

    }

}

for (char c : output) {

    std::cout << c << std::endl;

}

return 0;

}
```

## Output

```
main.cpp
1 #include <iostream>
2 #include <stack>
3 #include <string>
4 #include <vector>
5
6 int main() {
7     int Q;
8     std::cin >> Q;
9
10    std::string S;
11    std::stack<std::pair<int, std::string>> history;
12    std::vector<char> output;
13
14    while (Q-- > 0) {
15        int t;
16        std::cin >> t;
17
18        if (t == 1) {
19            std::string W;
20            std::cin >> W;
21            history.push({t, W});
22            S += W;
23        } else if (t == 2) {
24            int k;
25            if (history.empty()) {
26                std::cout << "Error: Stack is empty\n";
27                return 1;
28            }
29            std::pair<int, std::string> p = history.top();
30            history.pop();
31            output.push_back(p.second[0]);
32        }
33    }
34    std::cout << S << "\n";
35    return 0;
36}
```

input

```
3
2
4
terminate called after throwing an instance of 'std::out_of_range'
what():  basic_string::substr: __pos (which is 18446744073709551612) > this->size() (which is 0)
```

**4 Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).**

**Implement the MyQueue class:**

**void push(int x)** Pushes element x to the back of the queue.

**int pop()** Removes the element from the front of the queue and returns it.

**int peek()** Returns the element at the front of the queue. **boolean empty()**

Returns true if the queue is empty, false otherwise.

**Notes:**

You must use only standard operations of a stack, which means only push to top, peek/pop from top, size, and is empty operations are valid.

Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.

**Example 1:**

**Input**

```
["MyQueue",
 "push", "push",
 "peek", "pop",
 "empty"]
[[], [1], [2], [], [], []]
```

**Output**

```
[null, null, null, 1, 1, false]
```

**Explanation**

- `MyQueue myQueue = new MyQueue();`
- `myQueue.push(1);` // queue is: [1]
- `myQueue.push(2);` // queue is: [1, 2] (leftmost is front of the queue)
- `myQueue.peek();` // return 1
- `myQueue.pop();` // return 1, queue is [2]
- `myQueue.empty();` // return false

### Example 2:

#### Input:

`["MyQueue", "push", "push", "push", "peek", "pop", "peek", "empty"]`  
`[[], [3], [5], [7], [], [], [], []]`

#### Output:

`[null, null, null, null, 3, 3, 5, false]` **Explanation:**

```
MyQueue myQueue = new MyQueue()
myQueue.push(3) # queue is: [3]
myQueue.push(5) # queue is: [3, 5]
myQueue.push(7) # queue is: [3, 5, 7]
myQueue.peek() # return 3
myQueue.pop()  # return 3, queue is: [5, 7]
myQueue.peek() # return 5
myQueue.empty() # return False
```

### Example 3:

#### Input:

`["MyQueue", "push", "peek", "push", "pop", "pop", "empty"]`  
`[[], [8], [], [10], [], [], []]`

#### Output:

`[null, null, 8, null, 8, 10, true]` **Explanation:**

```
MyQueue myQueue = new MyQueue()
myQueue.push(8) # queue is: [8]
myQueue.peek() # return 8
myQueue.push(10) # queue is: [8, 10]
myQueue.pop()   # return 8, queue is: [10]
myQueue.pop()   # return 10, queue is: []
myQueue.empty() # return True
```

### Constraints:

- $1 \leq x \leq 9$
- At most 100 calls will be made to push, pop, peek, and empty.
- All the calls to pop and peek are valid.

**Asked By :** Amazon,Google

**Reference :** <https://leetcode.com/problems/implement-queue-using-stacks/description/>

## Code

```
#include <iostream>

#include <stack>

class MyQueue {

private:

    std::stack<int> stack1, stack2;


    void transferStack() {        if
(stack2.empty()) {                while
(!stack1.empty()) {
stack2.push(stack1.top());
stack1.pop();
        }
    }
}

public:

    MyQueue() {}


    void push(int x) {
stack1.push(x);
    }
```



```

    int pop() {
transferStack();    int front
=                stack2.top();
stack2.pop();        return
front;
    }

```

```

    int peek() {
transferStack();    return
stack2.top();
    }

```

```

    bool empty() {    return stack1.empty()
&& stack2.empty();
    }
};

```

```

int main() {

    MyQueue myQueue;    myQueue.push(3);
myQueue.push(5);    myQueue.push(7);
std::cout << myQueue.peek() << std::endl;
std::cout << myQueue.pop() << std::endl;
std::cout << myQueue.peek() << std::endl;
std::cout << myQueue.empty() << std::endl;
}

```

```

    MyQueue anotherQueue;

    anotherQueue.push(8);    std::cout <<

    anotherQueue.peek() << std::endl;

    anotherQueue.push(10);   std::cout <<

    anotherQueue.pop() << std::endl;    std::cout <<

    anotherQueue.pop() << std::endl;    std::cout <<

    anotherQueue.empty() << std::endl;

    return 0; }

```

## Output

The screenshot shows a C++ IDE with a file named `main.cpp`. The code implements a `MyQueue` class using two stacks, `stack1` and `stack2`. The `transferStack` method moves elements from `stack1` to `stack2` until `stack1` is empty. The `push` method adds elements to `stack1`, and the `pop` method removes elements from `stack1`. The output window shows the results of the program execution.

```

main.cpp
1  #include <iostream>
2  #include <stack>
3
4  class MyQueue {
5  private:
6      std::stack<int> stack1, stack2;
7
8      void transferStack() {
9          if (stack2.empty()) {
10             while (!stack1.empty()) {
11                 stack2.push(stack1.top());
12                 stack1.pop();
13             }
14         }
15     }
16
17 public:
18     MyQueue() {}
19
20     void push(int x) {
21         stack1.push(x);
22     }
23
24     int pop() {

```

input

```

8
10
1

...Program finished with exit code 0
Press ENTER to exit console.

```

**5 You are given an array of strings tokens that represents an arithmetic expression in a Reverse Polish Notation.**

**Evaluate the expression. Return an integer that represents the value of the expression.**

**Note that:**

- The valid operators are '+', '-', '\*', and '/'.
- Each operand may be an integer or another expression.
- The division between two integers always truncates toward zero.
- There will not be any division by zero.
- The input represents a valid arithmetic expression in a reverse polish notation.
- The answer and all the intermediate calculations can be represented in a 32-bit integer.

**Example 1:**

**Input:** tokens = ["2","1","+","3","\*"]

**Output:** 9

**Explanation:**  $((2 + 1) * 3) = 9$

**Example 2:**

**Input:** tokens = ["4","13","5","/","+"]

**Output:** 6

**Explanation:**  $(4 + (13 / 5)) = 6$

**Example 3:**

**Input:** tokens = ["10","6","9","3","+","-11","\*","/","\*","17","+","5","+"]

**Output:** 22

**Explanation:**  $((10 * (6 / ((9 + 3) * -11))) + 17) + 5$

$= ((10 * (6 / (12 * -11))) + 17) + 5$

$= ((10 * (6 / -132)) + 17) + 5$

$= ((10 * 0) + 17) + 5$

$= (0 + 17) + 5$

$= 17 + 5$

$= 22$

**Constraints:**

- `1 <= tokens.length <= 104`
- `tokens[i]` is either an operator: `"+"`, `"-"`, `"*"`, or `"/"`, or an integer in the range `[-200, 200]`.

## Approach

Using Fundamentals of STACK & LAMBDA...

**Time complexity:**  $O(n)$

**Space complexity:**  $O(n)$

## Code

```
#include <iostream>
```

```
#include <vector>
```

```
#include <stack>
```

```
#include <string>
```

```
int evalRPN(std::vector<std::string>& tokens) {
```

```
    std::stack<int> stack;
```

```
    for (const auto& token : tokens) {        if (token == "+" || token
== "-" || token == "*" || token == "/") {        int b = stack.top();
stack.pop();        int a = stack.top();        stack.pop();
```

```
        if (token == "+") stack.push(a + b);
else if (token == "-") stack.push(a - b);
else if (token == "*") stack.push(a * b);
else if (token == "/") stack.push(a / b);
```

```

        } else {
stack.push(std::stoi(token));

        }

    }

    return stack.top();
}

int main() {    std::vector<std::string> tokens1 = {"2", "1", "+", "3", "*"};    std::vector<std::string>
tokens2 = {"4", "13", "5", "/", "+"};    std::vector<std::string> tokens3 = {"10", "6", "9", "3", "+", "-
11", "*", "/", "*", "17", "+", "5", "+"};

    std::cout << evalRPN(tokens1) << std::endl;
std::cout << evalRPN(tokens2) << std::endl;
std::cout << evalRPN(tokens3) << std::endl;

    return 0;
}

```

## Output

```
main.cpp
1 #include <iostream>
2 #include <vector>
3 #include <stack>
4 #include <string>
5
6 int evalRPN(std::vector<std::string>& tokens) {
7     std::stack<int> stack;
8
9     for (const auto& token : tokens) {
10         if (token == "+" || token == "-" || token == "*" || token == "/") {
11             int b = stack.top();
12             stack.pop();
13             int a = stack.top();
14             stack.pop();
15
16             if (token == "+") stack.push(a + b);
17             else if (token == "-") stack.push(a - b);
18             else if (token == "*") stack.push(a * b);
19             else if (token == "/") stack.push(a / b);
20         } else {
21             stack.push(std::stoi(token));
22         }
23     }
24 }
```

input

9  
6  
22

...Program finished with exit code 0  
Press ENTER to exit console.

## MEDIUM

1. Given a circular integer array `nums` (i.e., the next element of `nums[nums.length - 1]` is `nums[0]`), return the next greater number for every element in `nums`.

The next greater number of a number `x` is the first greater number to its traversing-order next in the array, which means you could search circularly to find its next greater number. If it doesn't exist, return `-1` for this number.

**Example 1:**

**Input:** `nums = [1,2,1]`

**Output:** `[2,-1,2]` **Code:**

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;

bool parseBoolExpr(string expression) {
    stack<bool> s;

    for (int i = expression.size() - 1; i >= 0; i--) {
        char c = expression[i];

        if (c == 't') {
            s.push(true);
        } else if (c == 'f') {
            s.push(false);
        } else if (c == ')') {
            continue;
        } else if (c == '!') {
            bool val = s.top(); s.pop();
            s.push(!val);
        } else if (c == '&') {
            bool result = true;
            while (expression[i] != '(') {
                i--;
            }
            while (expression[i] != ')') {
                if (expression[i] == 't') {
                    result = result && true;
                } else if (expression[i] == 'f') {
                    result = result && false;
                }
                i--;
            }
            s.push(result);
        }
    }
}
```

```

    } else if (c == '|') {
        bool result = false;
        while (expression[i] != '(') {
            i--;
        }
        while (expression[i] != ')') {
            if (expression[i] == 't') {
                result = result || true;
            } else if (expression[i] == 'f') {
                result = result || false;
            }
            i--;
        }
        s.push(result);
    }
}

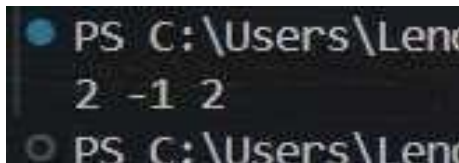
return s.top();
}

int main() {
    string expression = "&(|(f))";
    cout << parseBoolExpr(expression) << endl; // Output: false

    return 0;
}

```

**Output:**



**2. Given a queue, write a recursive function to reverse it.**

**Standard operations allowed :**

**enqueue(x) :** Add an item x to rear of queue. **dequeue() :**

**Remove an item from front of queue. empty() :** Checks if a queue is empty or not.

**Examples 1:**

**Input : Q = [5, 24, 9, 6, 8, 4, 1, 8, 3, 6]**

**Output : Q = [6, 3, 8, 1, 4, 8, 6, 9, 24, 5]**

**Explanation :** Output queue is the reverse of the input queue.

**Code:**

```
#include <iostream>
```



```
#include <queue>

using namespace std;

void reverseQueue(queue<int>& q) {

    if (q.empty()) {

        return;

    }

    int frontElement = q.front();

    q.pop();

    reverseQueue(q);

    q.push(frontElement);

}
```

```
int main() {

    queue<int> q;

    q.push(5);

    q.push(24);

    q.push(9);

    q.push(6);

    q.push(8);

    q.push(4);

    q.push(1);

    q.push(8);

    q.push(3);

}
```

```
q.push(6);

cout << "Original Queue: ";

queue<int> temp = q;

while (!temp.empty()) {

    cout << temp.front() << " ";

    temp.pop();

}

cout << endl;

reverseQueue(q);

cout << "Reversed Queue: ";

while (!q.empty()) {

    cout << q.front() << " ";

    q.pop();

}

cout << endl;

return 0;

}
```

**Output:**

```
● PS C:\Users\Lenovo\Desktop\c++> cd "c:
Original Queue: 5 24 9 6 8 4 1 8 3 6
Reversed Queue: 6 3 8 1 4 8 6 9 24 5
○ PS C:\Users\Lenovo\Desktop\c++>
```

**3. Given a balanced parentheses string  $s$ , return the score of the string.**

**The score of a balanced parentheses string is based on the following rule:**

**"()" has score 1.**

**AB has score  $A + B$ , where A and B are balanced parentheses strings.**

**(A) has score  $2 * A$ , where A is a balanced parentheses string.**

**Example 1:**

**Input:  $s = "()"$**

**Output: 1**

**Code:**

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;

int scoreOfParentheses(string s) {
    stack<int> st;
    for (char c : s) {
        if (c == '(') {
            st.push(0);
        } else {
            int top = st.top();
            st.pop();

            if (top == 0) {
                st.push(1);
            } else {
                int innerScore = top * 2;
                st.pop();
                st.push(st.empty() ? innerScore : innerScore + st.top());
            }
        }
    }
}
```

```

    }
}

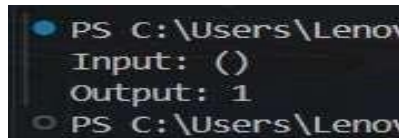
int score = 0;
while (!st.empty()) {
    score += st.top();
    st.pop();
}

return score;
}

int main() {
    string s = "()";
    cout << "Input: " << s << endl;
    cout << "Output: " << scoreOfParentheses(s) << endl;
    return 0;
}

```

**Output:**



```

PS C:\Users\Lenov
Input: ()
Output: 1
PS C:\Users\Lenov

```

## HARD

1. You are given an array of integers **nums**, there is a sliding window of size **k** which is moving from the very left of the array to the very right. You can only see the **k** numbers in the window. Each time the sliding window moves right by one position.  
**Return the max sliding window.**

**Example 1:**

**Input:** **nums** = [1,3,-1,-3,5,3,6,7], **k**  
= 3 **Output:** [3,3,5,5,6,7]

**CODE:**

```

#include <iostream>
#include <vector>
#include <deque>
using namespace std;

vector<int> maxSlidingWindow(vector<int>& nums, int k) {
    deque<int> dq;
    vector<int> result;

    for (int i = 0; i < nums.size(); ++i) {

```

```

    if (!dq.empty() && dq.front() == i - k) {
        dq.pop_front();
    }

    while (!dq.empty() && nums[dq.back()] < nums[i]) {
        dq.pop_back();
    }

    dq.push_back(i);

    if (i >= k - 1) {
        result.push_back(nums[dq.front()]);
    }
}

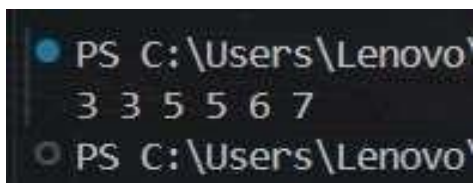
return result;
}

int main() {
    vector<int> nums = {1, 3, -1, -3, 5, 3, 6, 7};
    int k = 3;
    vector<int> result = maxSlidingWindow(nums, k);
    for (int x : result) {
        cout << x << " ";
    }
    cout << endl;

    return 0;
}

```

**Output:**



2. You have an infinite number of stacks arranged in a row and numbered (left to right) from 0, each of the stacks has the same maximum capacity.

**Implement the DinnerPlates class:**

**DinnerPlates(int capacity)** Initializes the object with the maximum capacity of the stacks capacity.

**void push(int val)** Pushes the given integer val into the leftmost stack with a size less than capacity.

**int pop()** Returns the value at the top of the rightmost non-empty stack and removes it from that stack, and returns -1 if all the stacks are empty.

**int popAtStack(int index)** Returns the value at the top of the stack with the given index index and removes it from that stack or returns -1 if the stack with that given index is empty.

**Code:**

```
#include <iostream>
#include <vector>
#include <stack>
#include <set>
using namespace std;

class DinnerPlates {
private:
    int capacity;
    vector<stack<int>> stacks;
    set<int> available;

public:
    DinnerPlates(int cap) : capacity(cap) {}

    void push(int val) {
        if (available.empty()) {
            stacks.push_back(stack<int>());
            available.insert(stacks.size() - 1);
        }

        int idx = *available.begin();
        stacks[idx].push(val);
        if (stacks[idx].size() == capacity) {
            available.erase(idx);
        }
    }

    int pop() {
        while (!stacks.empty() && stacks.back().empty()) {
            stacks.pop_back();
        }
        if (stacks.empty()) return -1;

        int val = stacks.back().top();
        stacks.back().pop();
        available.insert(stacks.size() - 1);
        return val;
    }

    int popAtStack(int index) {
        if (index >= stacks.size() || stacks[index].empty()) return -1;
```

```

        int val = stacks[index].top();
        stacks[index].pop();
        available.insert(index);
        return val;
    }
};

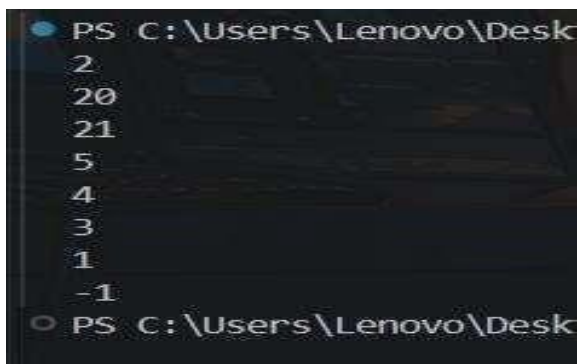
int main() {
    DinnerPlates D(2);
    D.push(1);
    D.push(2);
    D.push(3);
    D.push(4);
    D.push(5);

    cout << D.popAtStack(0) << endl; // 2
    D.push(20);
    D.push(21);

    cout << D.popAtStack(0) << endl; // 1
    cout << D.popAtStack(2) << endl; // -1
    cout << D.pop() << endl;        // 5
    cout << D.pop() << endl;        // 4
    cout << D.pop() << endl;        // -1
    cout << D.pop() << endl;
    return 0;}

```

### Output:



```

PS C:\Users\Lenovo\Desktop>
2
20
21
5
4
3
1
-1
PS C:\Users\Lenovo\Desktop>

```

3. Suppose there is a circle. There are  $N$  petrol pumps on that circle. Petrol pumps are numbered 0 to  $(N-1)$  (both inclusive). You have two pieces of information corresponding to each of the petrol pump: (1) the amount of petrol that particular petrol pump will give, and (2) the distance from that petrol pump to the next petrol pump.

Initially, you have a tank of infinite capacity carrying no petrol. You can start the tour at any of the petrol pumps. Calculate the first point from where the truck will be able to complete the circle. Consider that the truck will stop at each of the petrol pumps. The truck will move one kilometer for each litre of the petrol.

**Code:**

```
#include <iostream>
#include <vector>
using namespace std;

int findStartingPoint(int N, vector<pair<int, int>> &pumps) {
    long long total_petrol = 0, total_distance = 0;
    long long current_petrol = 0;
    int start = 0;

    for (int i = 0; i < N; ++i) {
        total_petrol += pumps[i].first;
        total_distance += pumps[i].second;
        current_petrol += pumps[i].first - pumps[i].second;

        if (current_petrol < 0) {
            start = i + 1;
            current_petrol = 0;
        }
    }

    if (total_petrol >= total_distance) {
        return start;
    }

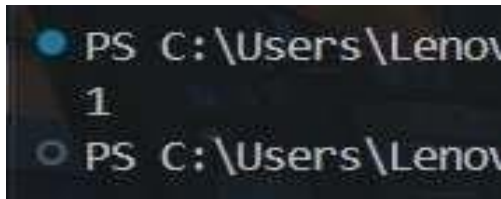
    return -1;
}
```



```
}
```

```
int main() {  
    int N = 3;  
    vector<pair<int, int>> pumps = {{1, 5}, {10, 3}, {3, 4}};  
    cout << findStartingPoint(N, pumps) << endl;  
    return 0;  
}
```

**Output:**



### VERY HARD

1. There are a number of plants in a garden. Each of the plants has been treated with some amount of pesticide. After each day, if any plant has more pesticide than the plant on its left, being weaker than the left one, it dies.

**You are given the initial values of the pesticide in each of the plants. Determine the number of days after which no plant dies, i.e. the time after which there is no plant with more pesticide content than the plant to its left.**

**Code:**

```
#include <iostream>  
#include <vector>  
#include <stack>  
#include <algorithm>  
using namespace std;  
  
int poisonousPlants(vector<int>& p) {  
    int n = p.size();  
    stack<int> st;  
    vector<int> days(n, 0);  
    int maxDays = 0;  
  
    for (int i = 0; i < n; ++i) {  
        while (!st.empty() && p[i] <= p[st.top()]) {  
            st.pop();  
        }  
        if (!st.empty()) {  
            days[i] = days[st.top()] + 1;  
        }  
        st.push(i);  
    }  
    maxDays = *max_element(days.begin(), days.end());  
    return maxDays;  
}
```

```

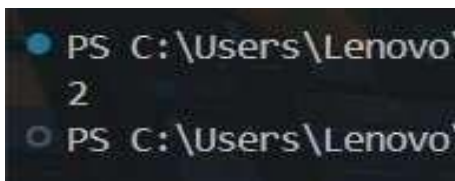
    }
    st.push(i);
    maxDays = max(maxDays, days[i]);
}

return maxDays;
}

int main() {
    int n = 7;
    vector<int> p = {6, 5, 8, 4, 7, 10, 9};
    cout << poisonousPlants(p) << endl;
    return 0;
}

```

**Output:**



**2. You are given an integer array nums of length n and an integer array queries.**

**Let gcdPairs denote an array obtained by calculating the GCD of all possible pairs (nums[i], nums[j]), where  $0 \leq i < j < n$ , and then sorting these values in ascending order.**

**For each query queries[i], you need to find the element at index queries[i] in gcdPairs. Return an integer array answer, where answer[i] is the value at gcdPairs[queries[i]] for each query.**

**The term gcd(a, b) denotes the greatest common divisor of a and b.**

**Code:**

```

#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>

using namespace std;

vector<int> gcdQueries(vector<int> &nums, vector<int> &queries) {
    vector<int> gcdPairs;
    int n = nums.size();

    // Calculate GCD for all pairs

```

```

for (int i = 0; i < n; ++i) {
    for (int j = i + 1; j < n; ++j) {
        gcdPairs.push_back(gcd(nums[i], nums[j]));
    }
}

// Sort GCD values in ascending order
sort(gcdPairs.begin(), gcdPairs.end());

vector<int> answer;
for (int query : queries) {
    answer.push_back(gcdPairs[query]);
}

return answer;
}

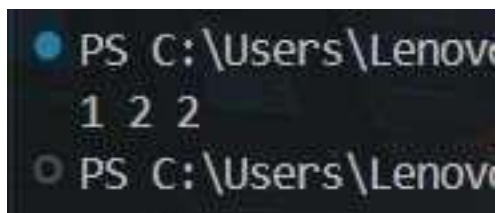
int main() {
    vector<int> nums = {2, 3, 4};
    vector<int> queries = {0, 2, 2};

    vector<int> result = gcdQueries(nums, queries);
    for (int res : result) {
        cout << res << " ";
    }
    cout << endl;

    return 0;
}

```

**Output:**



3. Given a string containing just the characters '(' and ')', return the length of the longest valid (well-formed) parentheses substring.

**Example 1:**

**Input:** s = "()" **Output:** 2

**Code:**

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;

int longestValidParentheses(string s) {
    stack<int> st;
    st.push(-1);
    int maxLength = 0;

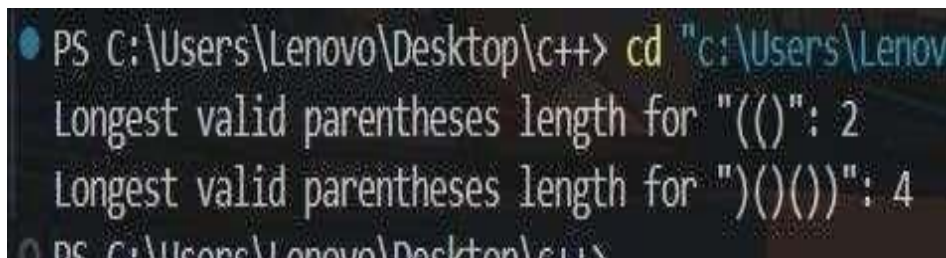
    for (int i = 0; i < s.length(); ++i) {
        if (s[i] == '(') {
            st.push(i);
        } else {
            st.pop();
            if (!st.empty()) {
                maxLength = max(maxLength, i - st.top());
            } else {
                st.push(i);
            }
        }
    }

    return maxLength;
}

int main() {
    string s1 = "(()";
    string s2 = ")()()";

    cout << "Longest valid parentheses length for \"" << s1 << "\": " << longestValidParentheses(s1)
    << endl;
    cout << "Longest valid parentheses length for \"" << s2 << "\": " << longestValidParentheses(s2)
    << endl;

    return 0;
}
```

**Output:**

```
PS C:\Users\Lenovo\Desktop\c++> cd "c:\Users\Lenovo\Desktop\c++"
Longest valid parentheses length for "(()": 2
Longest valid parentheses length for ")()()": 4
PS C:\Users\Lenovo\Desktop\c++>
```

4. You are given an integer array `nums` and an integer `k`.

Find the longest subsequence of `nums` that meets the following requirements:

- a. The subsequence is strictly increasing and
- b. The difference between adjacent elements in the subsequence is at most `k`.

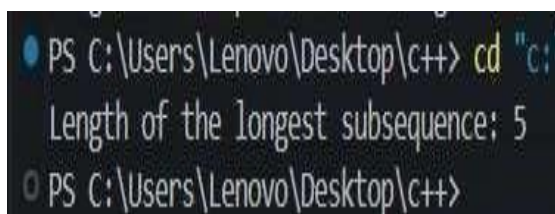
Return the length of the longest subsequence that meets the requirements.

A subsequence is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

**Code:**

```
#include <iostream>
#include <vector> #include
<algorithm> using namespace
std;
int longestSubsequence(vector<int> &nums, int k)
{   int n =
    nums.size();
    vector<int> dp(n, 1);
    for (int i = 1; i < n; ++i)
        {   for (int j = 0; j < i; ++j)
            {
                if (nums[i] > nums[j] && nums[i] - nums[j] <= k)
                {
                    dp[i] = max(dp[i], dp[j] + 1);
                }
            }
        }
    return *max_element(dp.begin(), dp.end());
} int main()
{
    vector<int> nums = {4, 2, 1, 4, 3, 4, 5, 8, 15};
    int k = 3;
    cout << "Length of the longest subsequence: " << longestSubsequence(nums, k) << endl;
    return 0;
}
```

**Output:**



```
PS C:\Users\Lenovo\Desktop\c++> cd "c:\Users\Lenovo\Desktop\c++>
Length of the longest subsequence: 5
PS C:\Users\Lenovo\Desktop\c++>
```

