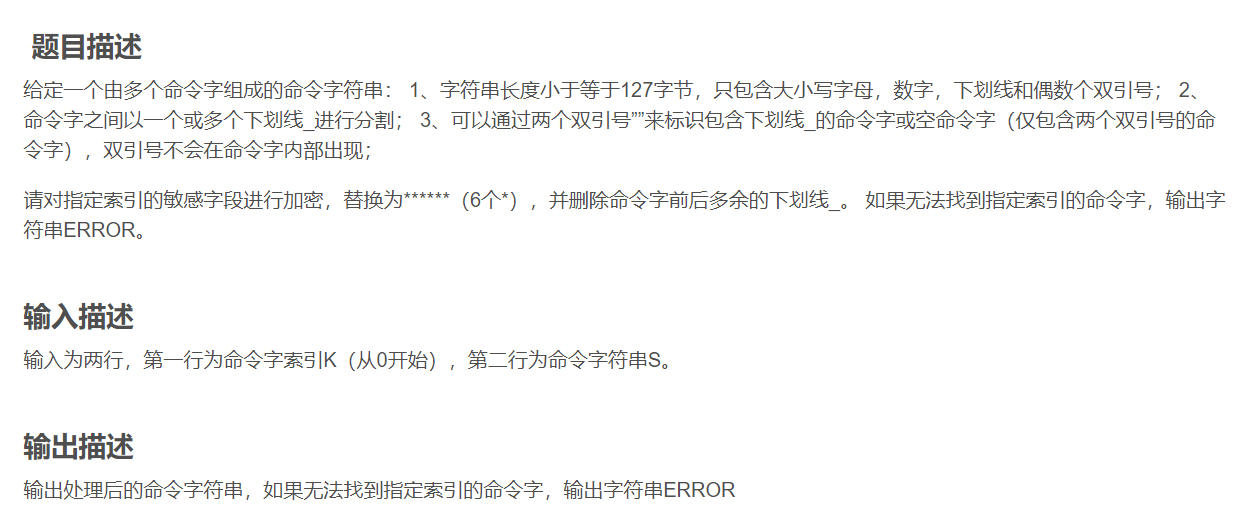
# **E卷-敏感字段加密[100分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**





1

password\_\_a12345678\_timeout\_100



password\_\*\*\*\*\*\*\_timeout\_100



2

aaa\_password\_"a12\_45678"\_timeout\_\_100\_""\_



aaa\_password\_\*\*\*\*\*\*\_timeout\_100\_""



import java.util.\*;

public class Main {

public static void main(String[] args) {

try {

// 创建扫描器读取输入

Scanner sc = new Scanner(System.in);

// 读取第一个输入行，表示敏感字段的索引K

int k = sc.nextInt();

sc.nextLine(); // 读取行结束符

// 读取第二个输入行，表示命令字符串S

String s = sc.nextLine();

// 标志变量，用于标记是否处于双引号内

boolean f = false;

// 用于存储处理后的字符串版本，双引号内的下划线会被替换为'!'

String str = "";

// 遍历输入的命令字符串S

for (int i = 0; i < s.length(); ++i) {

char x = s.charAt(i); // 获取当前字符

if (!f) { // 未处于双引号内

str += x; // 直接添加字符到str

if (x == '"') // 遇到双引号，进入双引号内

f = true;

} else { // 处于双引号内

if (x == '"') // 遇到双引号，退出双引号内

f = false;

if (x == '\_') // 将下划线替换为'!'

str += "!";

else

str += x; // 添加字符到str

}

}

// 根据下划线分割处理过的字符串

String[] arr = str.split("\_");

// 用于存储去除空命令字后的结果

ArrayList<String> res = new ArrayList<>();

for (String i : arr) {

if (!i.isEmpty()) {

res.add(i); // 添加非空的命令字到结果集中

}

}

// 检查索引k是否合法，如果不合法则捕获异常输出ERROR

res.set(k, "\*\*\*\*\*\*");

// 标志变量，用于控制输出时是否添加下划线

f = false;

// 遍历结果集，进行输出

for (String i : res) {

if (f)

System.out.print("\_"); // 如果需要则添加下划线

f = true;

System.out.print(i.replaceAll("!", "\_")); // 将'!'替换回下划线输出

}

} catch (Exception e) {

System.out.println("ERROR"); // 捕获任何异常输出ERROR

}

}

}



def main():

try:

import sys

input = sys.stdin.read().splitlines()

k = int(input[0]) # 读取第一个输入行，表示敏感字段的索引K

s = input[1] # 读取第二个输入行，表示命令字符串S

f = False # 标志变量，用于标记是否处于双引号内

str = "" # 用于存储处理后的字符串版本，双引号内的下划线会被替换为'!'

for x in s:

if not f: # 未处于双引号内

str += x # 直接添加字符到str

if x == '"': # 遇到双引号，进入双引号内

f = True

else: # 处于双引号内

if x == '"': # 遇到双引号，退出双引号内

f = False

if x == "\_": # 将下划线替换为'!'

str += "!"

else:

str += x # 添加字符到str

arr = str.split("\_") # 根据下划线分割处理过的字符串

res = [i for i in arr if i] # 用于存储去除空命令字后的结果

# 检查索引k是否合法，如果不合法则捕获异常输出ERROR

if k < 0 or k >= len(res):

print("ERROR")

return

res[k] = "\*\*\*\*\*\*" # 设置敏感字段，替换为\*\*\*\*\*\*

result = "\_".join(i.replace("!", "\_") for i in res) # 将'!'替换回下划线输出

print(result) # 输出结果字符串

except Exception as e:

print("ERROR") # 捕获任何异常输出ERROR

if \_\_name\_\_ == "\_\_main\_\_":

main()



#include <iostream>

#include <string>

#include <vector>

#include <sstream>

using namespace std;

int main() {

try {

int k;

cin >> k;

cin.ignore(); // 忽略换行符

string s;

getline(cin, s);

bool f = false;

string str = "";

// 遍历输入的命令字符串S

for (char x : s) {

if (!f) { // 未处于双引号内

str += x; // 直接添加字符到str

if (x == '"') // 遇到双引号，进入双引号内

f = true;

} else { // 处于双引号内

if (x == '"') // 遇到双引号，退出双引号内

f = false;

if (x == '\_') // 将下划线替换为'!'

str += "!";

else

str += x; // 添加字符到str

}

}

// 根据下划线分割处理过的字符串

vector<string> arr;

stringstream ss(str);

string temp;

while (getline(ss, temp, '\_')) {

arr.push\_back(temp);

}

// 用于存储去除空命令字后的结果

vector<string> res;

for (string i : arr) {

if (!i.empty()) {

res.push\_back(i); // 添加非空的命令字到结果集中

}

}

// 检查索引k是否合法

if (k < 0 || k >= res.size()) {

cout << "ERROR" << endl;

return 0;

}

res[k] = "\*\*\*\*\*\*"; // 设置敏感字段，替换为\*\*\*\*\*\*

// 标志变量，用于控制输出时是否添加下划线

bool first = true;

for (string i : res) {

if (!first) {

cout << "\_";

}

first = false;

for (char c : i) {

if (c == '!') {

cout << "\_"; // 将'!'替换回下划线输出

} else {

cout << c;

}

}

}

cout << endl;

} catch (exception& e) {

cout << "ERROR" << endl; // 捕获任何异常输出ERROR

}

return 0;

}



#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <ctype.h>

#define MAX\_SIZE 256

int main() {

int k;

char s[MAX\_SIZE];

if (scanf("%d\n", &k) != 1) {

printf("ERROR\n");

return 1;

}

if (fgets(s, MAX\_SIZE, stdin) == NULL) {

printf("ERROR\n");

return 1;

}

int len = strlen(s);

if (len > 0 && s[len - 1] == '\n') {

s[len - 1] = '\0';

}

int f = 0;

char str[MAX\_SIZE \* 2] =

""; // 用于存储处理后的字符串版本，双引号内的下划线会被替换为'!'

size\_t str\_pos = 0;

// 遍历输入的命令字符串S

for (int i = 0; s[i] != '\0'; ++i) {

if (!f) { // 未处于双引号内

str[str\_pos++] = s[i]; // 直接添加字符到str

if (s[i] == '"') // 遇到双引号，进入双引号内

f = 1;

} else { // 处于双引号内

if (s[i] == '"') // 遇到双引号，退出双引号内

f = 0;

if (s[i] == '\_') // 将下划线替换为'!'

str[str\_pos++] = '!';

else

str[str\_pos++] = s[i]; // 添加字符到str

}

}

str[str\_pos] = '\0';

char\* arr[MAX\_SIZE];

int arr\_size = 0;

char\* token = strtok(str, "\_");

// 用于存储去除空命令字后的结果

while (token != NULL) {

if (strlen(token) > 0) {

arr[arr\_size++] = token;

}

token = strtok(NULL, "\_");

}

// 检查索引k是否合法

if (k < 0 || k >= arr\_size) {

printf("ERROR\n");

return 1;

}

arr[k] = "\*\*\*\*\*\*"; // 设置敏感字段，替换为\*\*\*\*\*\*

// 标志变量，用于控制输出时是否添加下划线

int first = 1;

for (int i = 0; i < arr\_size; ++i) {

if (!first) {

printf("\_");

}

first = 0;

for (int j = 0; arr[i][j] != '\0'; ++j) {

if (arr[i][j] == '!') {

printf("\_"); // 将'!'替换回下划线输出

} else {

printf("%c", arr[i][j]);

}

}

}

printf("\n");

return 0;

}



const readline = require("readline");

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

let inputs = [];

rl.on("line", (line) => {

inputs.push(line);

if (inputs.length === 2) {

rl.close();

}

});

rl.on("close", () => {

try {

const k = parseInt(inputs[0], 10); // 读取第一个输入行，表示敏感字段的索引K

const s = inputs[1]; // 读取第二个输入行，表示命令字符串S

let f = false; // 标志变量，用于标记是否处于双引号内

let str = ""; // 用于存储处理后的字符串版本，双引号内的下划线会被替换为'!'

// 遍历输入的命令字符串S

for (let i = 0; i < s.length; ++i) {

const x = s[i];

if (!f) {

// 未处于双引号内

str += x; // 直接添加字符到str

if (x === '"')

// 遇到双引号，进入双引号内

f = true;

} else {

// 处于双引号内

if (x === '"')

// 遇到双引号，退出双引号内

f = false;

if (x === "\_")

// 将下划线替换为'!'

str += "!";

else str += x; // 添加字符到str

}

}

// 根据下划线分割处理过的字符串

const arr = str.split("\_").filter((i) => i); // 用于存储去除空命令字后的结果

// 检查索引k是否合法，如果不合法则捕获异常输出ERROR

if (k < 0 || k >= arr.length) {

console.log("ERROR");

return;

}

arr[k] = "\*\*\*\*\*\*"; // 设置敏感字段，替换为\*\*\*\*\*\*

const result = arr.map((i) => i.replace(/!/g, "\_")).join("\_"); // 将'!'替换回下划线输出

console.log(result); // 输出结果字符串

} catch (e) {

console.log("ERROR"); // 捕获任何异常输出ERROR

}

});



package main

import (

"bufio"

"fmt"

"os"

"strings"

)

func main() {

// 创建缓冲读取器

reader := bufio.NewReader(os.Stdin)

// 读取第一个输入行，表示敏感字段的索引k

var k int

fmt.Scan(&k)

// 读取第二个输入行，表示命令字符串s

s, \_ := reader.ReadString('\n')

s = strings.TrimSpace(s)

// 标志变量，用于标记是否处于双引号内

f := false

// 用于存储处理后的字符串版本，双引号内的下划线会被替换为'!'

var str strings.Builder

// 遍历输入的命令字符串S

for \_, x := range s {

if !f { // 未处于双引号内

str.WriteRune(x) // 直接添加字符到str

if x == '"' { // 遇到双引号，进入双引号内

f = true

}

} else { // 处于双引号内

if x == '"' { // 遇到双引号，退出双引号内

f = false

}

if x == '\_' { // 将下划线替换为'!'

str.WriteRune('!')

} else {

str.WriteRune(x) // 添加字符到str

}

}

}

// 根据下划线分割处理过的字符串

arr := strings.Split(str.String(), "\_")

// 用于存储去除空命令字后的结果

var res []string

for \_, i := range arr {

if i != "" {

res = append(res, i) // 添加非空的命令字到结果集中

}

}

// 检查索引k是否合法

if k < 0 || k >= len(res) {

fmt.Println("ERROR")

return

}

// 设置敏感字段，替换为\*\*\*\*\*\*

res[k] = "\*\*\*\*\*\*"

// 标志变量，用于控制输出时是否添加下划线

f = false

for \_, i := range res {

if f {

fmt.Print("\_")

}

f = true

fmt.Print(strings.ReplaceAll(i, "!", "\_")) // 将'!'替换回下划线输出

}

fmt.Println()

}