Introducción a Programación Competitiva en C++

Benjamín Letelier

benjamin.letelier@progcomp.cl

¿Necesito saber programar para poder solucionar un problema de programación competitiva?

¿Necesito saber programar para poder solucionar un problema de programación competitiva?

¡NO!

¿Necesito saber programar para poder solucionar un problema de programación competitiva?

Es más, ¡es común que en los equipos exista gente que no escribe ninguna línea de código en las 5 horas de competencia!

¿Cómo esto es posible?

Porque **NUNCA** deben programar un problema sin pensarlo antes.

Ejemplo

Benja y Alex quieren comerse un jurel que pesa `k` kilos. Ambos son fan de los números pares y detestan dejar comida, por lo que quieren comer un pedazo que tenga un peso par luego de ser porcionada por completo.

Dado el peso del jurel `k`, ¿es posible determinar si existe al menos una forma de cortar el jurel para que ambos coman una cantidad par de kilos de jurel?

Ejemplo 1

```
5
```

```
No
```

Ejemplo 2

```
8
```

```
Si
```

¿Como solucionamos el problema sin código?

Analicemos el problema (`k = 8`)

Para `k = 8`, tenemos tres porciones posibles que cumplen con la condición:

```
k = 8
Benja -> 1 | Alex -> 7 ❌
Benja -> 2 | Alex -> 6 ✅
Benja -> 3 | Alex -> 5 ❌
Benja -> 4 | Alex -> 4 ✅
Benja -> 5 | Alex -> 3 ❌
Benja -> 6 | Alex -> 2 ✅
Benja -> 7 | Alex -> 1 ❌
```

Como **EXISTE** al menos una solución, la respuesta es `Si`.

Analicemos el problema (`k = 5`)

En el caso de `k = 5`, No existe ningúna forma de porcionar posible que cumpla con la condición:

```
k = 5
Benja -> 1 | Alex -> 4 ❌
Benja -> 2 | Alex -> 3 ❌
Benja -> 3 | Alex -> 2 ❌
Benja -> 4 | Alex -> 1 ❌
```

Como **NO EXISTE** al menos una solución, la respuesta es `No`.

Analicemos el problema (General)

A partir de este pequeño análisis, ¿podemos sacar alguna conclusión?

Analicemos el problema (caso impar)

¡Con los números impares no va a funcionar nunca!

Analicemos el problema (caso par)

¿Eso implica que con todos los números pares va a funcionar siempre?

Analicemos el problema (caso `k=4`)

¡Probemos los casos bordes!

```
k = 4
Benja -> 1 | Alex -> 3 ❌
Benja -> 2 | Alex -> 2 ✅
Benja -> 3 | Alex -> 1 ❌
```

¡Funciona!

Analicemos el problema (caso `k=2`)

¡Probemos los casos bordes!

```
k = 2
Benja -> 1 | Alex -> 1 ❌
```

¡Aquí ya no funciona! Cada vez que bajamos de un número par a otro, se eliminan dos opciones.

Solucionemos el problema

Ya pudimos detectar que con cualquier `k` impar y `k=2` nunca va a funcionar. Eso ya nos permite dar solución al problema.

```
LEER(k)
SI((k ES IGUAL A 2) O (k ES IMPAR)):
    ESCRIBE("No")
SINO:
    ESCRIBE("Si")
```

Esto ya es válido para que cualquier miembro del equipo que sepa escribir código pueda transcribir la solución final.

Resultado del problema

Python

```python
k = int(input())

if k == 2 or k % 2 == 1:
    print("No")
else:
    print("Si")
```

C++

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int k;
    cin >> k;

    if (k == 2 or k % 2 == 1)
        cout << "No" << endl;
    else
        cout << "Si" << endl;
}
```

¿Qué fue más importante escribir el código o pensar la solución?

Estructura básica de un programa en C++

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {

}
```

`#include <bits/stdc++.h>`

Incluye la mayoría de las bibliotecas estándar de C++.

Esto nos permite utilizar estructuras y funciones como:

- `cin`
- `cout`
- `string`

sin tener que incluir cada biblioteca por separado..

`using namespace std;`

Permite utilizar los elementos de la biblioteca estándar sin escribir `std::`.

Por ejemplo:

Sin `using namespace std;`

```
std::cout << "Hola" << std::endl;
std::string s;
```

Con `using namespace std;`

```
cout << "Hola" << endl;
string s;
```

`int main()`

Es la función principal del programa **(siempre debe ser escrita así)**.

La ejecución siempre comienza aquí, por lo que toda la solución del problema se escribe dentro de esta función.

```
int main() {
    // Aquí escribiremos nuestra solución
}
```

Tipos de datos básicos en C++

Tipo	¿Qué almacena?	Ejemplo
`int`	Números enteros	`5`, `-20`, `1000`
`long long`	Enteros muy grandes (`> 10^9`)	`1000000000000`
`float`	Números reales (precisión simple)	`3.14`
`double`	Números reales (mayor precisión)	`2.718281828`
`char`	Un único carácter	`'A'`, `'7'`, `'?'`
`string`	Texto	`"Hola Mundo"`

Declarando e imprimiendo variables

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int entero = 10;
    long long entero_grande = 10000000000000LL;
    float real_simple = 3.14f;
    double real_doble = 3.141592653589793;
    char caracter = 'A';
    string texto = "Hola";
    cout << caracter << ' ' << entero_grande << endl;
    cout << texto << endl << "10LL" << endl;
}
```

```
A 10000000000000
Hola
10LL
```

Entrada de datos: `cin`

En programación competitiva, los datos de entrada son entregados por el juez.

Para leer una variable utilizamos `cin`.

```
int k;
cin >> k;
```

Si la entrada es:

```
8
```

Entonces la variable `k` almacenará el valor `8`.

Salida de datos: `cout`

Una vez resuelto el problema, debemos imprimir la respuesta.

Para ello utilizamos `cout`.

```cpp
int respuesta = 42;
cout << respuesta << endl;
```

```
42
```

Podemos imprimir varias cosas seguidas utilizando `<<`, y utilizar `endl` para saltar línea.

```cpp
cout << "La respuesta es " << respuesta << endl;
```

Ejemplo completo (leer número y mostrarlo)

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int k;
    cin >> k;
    cout << k << endl;
}
```

Entrada

```
15
```

Salida

```
15
```

Entrada

```
3
10 20000000000 -30
```

C++

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    long long a, b, c, d;
    cin >> a >> b;
    cin >> c >> d;
    cout << a << ' ' << b;
    cout << ' ' << c << ' ' << d << endl;
    // 3 10 20000000000 -30
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    long long a;
    cin >> a;
    long long b, c, d;
    cin >> b >> c >> d;
    cout << a << ' ' << b << ' ' << c << ' ' << d << endl;
    // 3 10 20000000000 -30
}
```

Operadores matemáticos

Los operadores matemáticos permiten realizar operaciones aritméticas.

Operador	Descripción	Ejemplo
`+`	Suma	`a + b`
`-`	Resta	`a - b`
`*`	Multiplicación	`a * b`
`/`	División	`a / b`
`%`	Módulo (resto)	`a % b`

Operadores lógicos y de comparación

Permiten comparar valores y combinar condiciones.

Comparación

Operador	Significado
`==`	Igual
`!=`	Distinto
`<`	Menor que
`<=`	Menor o igual
`>`	Mayor que
`>=`	Mayor o igual

Lógicos

Operador	Significado
`and`	Y (AND)
`or`	O (OR)
`!`	NO (NOT)

Estructuras condicionales

Las estructuras condicionales permiten ejecutar diferentes bloques de código según se cumpla o no una condición.

```
if (condición) {
    // Se ejecuta si la condición es verdadera.
}
else if (otra_condición) {
    // Se ejecuta si la primera es falsa
    // y esta condición es verdadera.
}
else {
    // Se ejecuta si ninguna condición anterior
    // es verdadera.
}
```

> Se pueden utilizar tantos `else if` como sean necesarios, pero solo puede existir un `else`.

Estructuras condicionales

```cpp
#include <bits/stdc++.h>
using namespace std;

int main (){
    int k;
    cin >> k;
    if (k == 2) {
        cout << "No" << endl;
    }
    else if (k % 2 == 0) {
        cout << "Si" << endl;
    }
    else {
        cout << "No" << endl;
    }
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;

int main (){
    int k;
    cin >> k;
    if (k % 2 == 0) {
        cout << "Si" << endl;
    }
    else if (k == 2) {
        cout << "No" << endl;
    }
    else {
        cout << "No" << endl;
    }
}
```

> ¡Hay que tener mucho cuidado con el orden de las condiciones!

Ejemplo

Benja tiene un poder mágico para poder sumar 1 a cualquier número en una lista. Saliendo de su casa, se encontró con una lista de `n` números con valores `-1`, `0` o `1`, y quiere asegurarse que el producto entre todos los números de la lista sea estrictamente positivo.

¿Cuál es el mínimo número de operaciones que necesita Benja para lograr que el producto sea estrictamente positivo?

Ejemplo 1

```
3
-1 0 1
```

```
3
```

Ejemplo 2

```
4
-1 -1 0 1
```

```
1
```

Ejemplo 3

```
5
-1 -1 -1 0 0
```

```
4
```

Analicemos el problema

En el primer caso existen muchas posibles soluciones. Por ejemplo, Benja puede realizar `+1000` cada posición obteniendo el arreglo `[999, 1000, 1001]` donde claramente el producto es positivo. Esta solución costaría `3000` operaciones.

¿Existirá una forma de lograrlo con menos operaciones?, ¿por qué el resultado es 3?

Analicemos el problema

Vale la pena observar que para que el resultado sea estrictamente positivo, se deben dar dos cosas. La primera es que no exista ningún `0` (ya que la presencia de al menos un `0` convertiría todo el producto en `0`), y la segunda es que los `-1` no influyan en el signo final.

También, hay que darse cuenta que no sirve de nada aumentar cualquier `1`, ya que estaríamos gastando operaciones para nada.

Todo esto implica que el producto final a alcanzar en la lista **SIEMPRE SERÁ** `1`, luego de aplicar todas las operaciones.

Solucionemos el problema

De esta manera, podríamos decir que la solución es la cantidad de `0` (ya que hay que convertirlos a `1`) y si la cantidad de `-1` es impar, hay que añadir `2` operaciones extra (para lograr hacer `-1 -> 0 -> 1`). Esto convertiría la cantidad de `-1` en par, y todos sabemos que **negativo por negativo da positivo**.

Solucionemos el problema

```
LEER(n)
contador_ceros = 0
contador_negativos = 0
REPETIR n VECES:
    LEER(numero)
    SI(numero ES IGUAL A 0):
        contador_ceros = contador_ceros + 1
    SINO SI(numero ES IGUAL A -1):
        contador_negativos = contador_negativos + 1
respuesta = contador_ceros
SI(contador_negativos ES IMPAR):
    respuesta = respuesta + 2
IMPRIME(respuesta)
```

Resultado del problema

Python

```python
n = int(input())
lista = list(map(int, input().split()))
contador_ceros = 0
contador_negativos = 0
for numero in lista:
    if numero == 0:
        contador_ceros = contador_ceros + 1
    elif numero == -1:
        contador_negativos = contador_negativos + 1
respuesta = contador_ceros
if contador_negativos % 2 == 1:
    respuesta = respuesta + 2
print(respuesta)
```

C++

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> numeros(n);
    for(int i = 0; i < n; ++i) {
        int numero;
        cin >> numero;
        numeros[i] = numero;
    }
    int contador_ceros = 0;
    int contador_negativos = 0;
    for(int i = 0; i < n; ++i) {
        int numero = numeros[i];
        if(numero == 0)
            contador_ceros = contador_ceros + 1;
        else if(numero == -1)
            contador_negativos = contador_negativos + 1;
    }
    int respuesta = contador_ceros;
    if (contador_negativos % 2 == 1)
        respuesta = respuesta + 2;
    cout << respuesta << endl;
}
```

El ciclo `for` permite repetir un bloque de código un número determinado de veces.

```
for (inicialización; condición; actualización) {
    // Código a repetir
}
```

> Lo utilizaremos comúnmente para leer input y solucionar problemas como en el ejemplo anterior.

Ejemplo:

```
for (int i = 0; i < 5; i++) {
    cout << i << endl;
}
```

Salida:

```
0
1
2
3
4
```

El ciclo `while` repite un bloque de código **mientras una condición sea verdadera**.

```
while (condición) {
    // Código a repetir
}
```

Ejemplo:

```
int i = 0;

while (i < 5) {
    cout << i << endl;
    i++;
}
```

Salida:

```
0
1
2
3
4
```

> A diferencia del `for`, en un `while` debemos actualizar manualmente la variable de control para evitar un ciclo infinito.

`while` versus `for`

Deberíamos utilizar un `for` cuando sepamos cuántas veces se repetirá algo, mientras que un `while` cuando no sepamos la cantidad exacta.

```cpp
vector<int> v; // vacío
v.push_back(42); // añade al final
v.push_back(7); // añade al final
v.push_back(13); // añade al final
cout << v.size() << endl; // muestra 3
cout << v[1] << endl; // muestra 7
v.pop_back(); // elimina el último elemento
cout << v.back() << endl; // muestra 7
```

Links de interés

OnlineGDB - compilador online de C++.

CPlusPlus Reference - documentación de C++.

Codeforces - donde competirán.