Documentación Técnica – ICPC 2025

Universidad Antonio Nariño

10 de octubre de 2025

Índice

1. Plantillas de Código

1.1. I/O rápido en C++

Se encargan de desincronizar el cin y el cout con el fin de eliminar el overhead y alcanzar una velocidad similar al printf y scanf de C.

```
ios::sync_with_stdio(false);
cin.tie(nullptr);
```

1.2. Macros utiles para el lenguaje

a. Atajos de sintaxis

```
#define pb push_back //metodo de insercion comun en vectores
#define mp make_pair //construccion de pares en ejecucion para maps
#define all(x) (x).begin(), (x).end() //util para sort y otros metodos que requieren iteradorres inicio y fin
#define rall(x) (x).rbegin(), (x).rend() //igual que all pero en orden inverso
#define sz(x) (int)((x).size()) //longitud de las principales DS std
```

b. Tipos de datos frecuentes

```
#define ll  long long //entero 64 bits (maximo espacio c++)
#define ull unsigned long long //igual a ll, para casos sin negativos
//vectores de los tipos principales
#define vi  vector<int>
#define vll vector<long long>
#define vd  vector<double>
#define vs  vector<string>
```

c. Bucles rapidos

```
#define FOR(i,a,b) for(long long i=(a); i<(b); i++) //Bucle for 1 a 1 desde a hasta b
#define REP(i,n)   for(long long i=0; i<(n); i++) //Bucle for 1 a 1 de 0 a n (Ideal para recorrer todo un vector)
#define ROF(i,a,b) for(long long i=(a); i>=(b); i--) //Bucle descendente 1 a 1 de a hasta b
```

d. Constantes

```
#define INF  1000000000 //Posible infinito para int
#define LINF 1000000000000000000LL //Posible infinito para long long
#define MOD  1000000007 //Constante 10^9 + 7 para problemas gigantes
#define MOD2 998244353 //Otra constante modular menos usada
#define EPS  1e-9 //Margen de error para operaciones punto flotante
```

e. Debugging rapido (No sabemos debuggear)

```
#define debug(x) cerr << #x << " = " << (x) << endl //Se usa cerr ya que no es evaluado por el juez
#define debugv(v) { cerr << #v << " = "; for(auto _ : v) cerr << _ << " "; cerr << endl; }
```

f. Conversiones rapidas (Tipos y bases)

```
#define toStr(x) to_string(x) //Conversion numero a string
//String a numeros de diferentes tamanos
#define toInt(x) stoi(x) //A int
#define toLL(x) stoll(x) //A long long
```

```
#define toD(x) stod(x) //A double
//Conversiones de notaciones
#define toBin(x, n) bitset<n>(x).to_string() //Entero a string binario de n bits (n debe ser constante)
#define toBin32(x) bitset<32>(x).to_string() //Entero a string binario de 32 bits (util para int)
#define binToDec(x) stoi(x, nullptr, 2) //String binario a entero(int)
#define hexToDec(x) stoi(x, nullptr, 16) //String hexadecimal a entero
#define toHex(x) ([](int v){ stringstream ss; ss<<hex<<v; return ss.str(); })(x) //Decimal(int) a notacion hexadecimal como string
```

g. Operaciones de bits (Con long long - complejidad O(1))

```
#define checkBit(x,i) ((x) & (1LL<<(i))) //Verificar bit i encendido
#define setBit(x,i)   ((x) | (1LL<<(i))) //Encender bit i
#define clearBit(x,i) ((x) & ~(1LL<<(i))) //Apagar bit i
#define flipBit(x,i)  ((x) ^ (1LL<<(i))) //Cambiar estado del bit inmediatamente (0 a 1 o 1 a 0)
#define getBit(x,i)   (((x)>>(i)) & 1LL) //Obtener valor del bit
#define popcount(x)   __builtin_popcountll(x) //Contar cantidad de bits en 1
#define lowbit(x)     ((x) & (-(x))) //Obtener indice del bit menos significativo encendido (x > 0)
#define msbIndex(x)   (63 - __builtin_clzll(x)) //Longitud de numero en bits (x > 0)
```

2. Lectura de casos

Para la lectura de la entrada del programa se presentan principalmente 3 tipos de entrada:

a. Numero de casos al inicio: Su principal caracteristica se encuentra en que la primera linea nos dice la cantidad de casos que vamos a recibir a continuación, independientemente de si estos casos van a ser numeros, letras o bloques estructurados de información

- Ejemplo de entrada:

```
3 -> Numero de casos
10 -> Caso 1
20 -> Caso 2
30 -> Caso 3
```

- Recepción ideal:
  Lo mejor para estos casos es recibir en una variable la cantidad de casos y luego con un bucle iterar recibiendo en cada caso la información e imprimiendo en el StdOut la respuesta (Nunca acumular la respuesta de multiples casos para arrojarla toda al final)

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

int main() {
  int numCases;
  cin>>numCases;
  for(int i = 0; i < numCases; i++){
    //Se recibe cada caso, se procesa y se da la respuesta inmediatamente
    int case_i;
    cin>>case_i;
    cout<<case_i<<endl;
  }
}
```

b. Casos hasta que se cumpla una condición: Se entregan casos en cada linea (O en bloques de lineas), hasta que se encuentre un caracter especial de finalización (0, -1, END, etc)

- Ejemplo de entrada:

```
2       -> Caso 1: Se recibirá un vector con 2 elementos
1 2     -> Vector del caso 1
2       -> Caso 2: Se recibirá otro vector con 2 elementos
3 4     -> Vector del caso 2
4       -> Caso 3: Se recibirá otro vector, esta vez de 4 elementos
1 2 3 4 -> Vector del caso 3
0       -> Fin de entrada dado por el 0
```

- Recepción ideal:
  Para estos casos lo mas común es utilizar un bucle do while o un while que incluya la recepción dentro de su condición, de esta forma podemos procesar uno a uno los casos, ir imprimiendo su salida y garantizar que frene sin imprimir valores extraños al recibir el caracter de finalización

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

int main() {
  int tamVec;
  while(
    cin>>tamVec //Recepcion dentro del condicional
    && tamVec != 0 //Validacion de las entradas recibidas
        inmediatamente
  ){
    //Creacion del vector de tamano tamVec con valor inicial 0
    vector<int> values(tamVec, 0);
    //Recepcion de los elementos del vector
    for(int i = 0; i < tamVec; i++){
      cin>>values[i];
    }
    //Impresion inmediata de la respuesta del caso
    cout<<"[ ";
    for(int value: values){
      cout<<value<<" ";
    }
    cout<<"]"<<endl;
  }
}
```

c. Casos hasta que se termine la entrada: No existe una indicación de final dentro de la entrada, el programa debe finalizar exactamente cuando ya no hayan mas casos de entrada

- Ejemplo de entrada:

```
10  -> Caso 1
20  -> Caso 2
30  -> Caso 3
40  -> Caso 4
```

- Recepción ideal:
  Para estos casos lo mas optimo (Y casi necesario) consiste en meter la lecutra de entrada dentro de un bucle while, si no sabemos la cantidad de valores exacta que vamos a recibir en cada caso, habrá que utilizar obligatoriamente los flags de fin de linea de cin, o getline si estrictamente cada linea es un caso

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

int main() {
  int num;
  while(cin>>num){ //Bucle se ejecuta hasta que ya no pueda recibir mas datos
    cout<<"Respuesta: "<<num<<endl; //Respuesta inmediata del caso
  }
}
```

d. Casos con texto o cantidad variable de datos por línea: Cada línea de la entrada representa un caso independiente, y puede contener desde texto libre hasta una cantidad arbitraria de números. En este escenario lo más sencillo es leer la línea completa como cadena y luego procesarla según la lógica del problema.

- Ejemplo de entrada:

```
Hola mundo
1 2 3
Esto es un caso
10 20 30 40
```

- Recepción ideal:
  Se recomienda usar `getline` para capturar la línea completa. Si se necesitan números, se puede pasar la línea a un `stringstream` y extraerlos; si es texto, se procesa directamente.

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

int main() {
  string line;
  while (getline(cin, line)) { // Leer cada linea hasta EOF
    if (line.empty()) continue; // Saltar lineas vacias

    // Procesar la linea (ejemplo: imprimirla)
    cout << "Caso: " << line << endl;
  }
}
```

3. Tablas de Complejidad

- `vector`: acceso $O(1)$, inserción al final $O(1)$ amortizado
- `set/map`: inserción/búsqueda $O(\log n)$
- `unordered_map`: inserción/búsqueda $O(1)$ promedio

4. Fórmulas Matemáticas

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\text{Área de triángulo (Herón)} = \sqrt{s(s-a)(s-b)(s-c)}$$

5. Estrategia de Equipo

Roles, protocolos de clarifications, rotación de teclado.