Apunte para tu primer problema

A. A la mitad - TAP 2026 - GNU C++17

Plantilla de C++

Este bloque se incluye una sola vez al principio del apunte generado. Reúne la estructura común para escribir un programa de competencia en GNU C++17. El índice del apunte se calcula a partir de los módulos y partes seleccionados.

El programa empieza a ejecutarse en `main`. Dentro de sus llaves se leen los datos, se calcula la respuesta y se imprime lo que pide el enunciado. Una función auxiliar declarada con su nombre se escribe fuera de `main`. Cada ejemplo de la biblioteca indica cuando necesita esa ubicación.

Los fragmentos breves de las demás fichas se colocan dentro de `main`. Cada bloque parte de los datos que declara, salvo cuando se indique que usa una función definida en el bloque anterior. Los comentarios muestran resultados esperados y condiciones de uso.

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    // Leer los datos.

    // Calcular la respuesta.

    // Imprimir la respuesta.

    return 0;
}
```

Parte	Para qué sirve
`#include <bits/stdc++.h>`	Reúne las cabeceras de la biblioteca estándar en GNU C++
`using namespace std;`	Permite escribir `cin`, `string` o `vector` sin anteponer `std::`
`int main()`	Define el punto de entrada del programa
`ios::sync_with_stdio(false)`	Permite entrada y salida rápidas con los flujos de C++
`cin.tie(nullptr)`	Evita que cada lectura vacíe automáticamente la salida
`return 0;`	Termina el programa normalmente

Con esta configuración, usá `cin` y `cout` de forma consistente. La cabecera `bits/stdc++.h` pertenece a GNU C++, el compilador habitual de estas prácticas. En otros entornos se incluyen las cabeceras estándar concretas, como `<iostream>`, `<vector>` y `<string>`.

Índice

Variables, tipos e inicialización 3
Entrada y salida 5
Condiciones y decisiones 7
Cuentas enteras y restos 9

Variables, tipos e inicialización

Descripción

Una variable guarda un valor con un nombre y un tipo. El tipo determina qué valores puede representar y qué operaciones se hacen con él. Elegilo usando las cotas del enunciado, incluidas las sumas y los productos intermedios.

Usá `int` para índices y cantidades acotadas, `long long` para sumas o productos grandes, `double` para resultados aproximados, `char` para un carácter y `bool` para una condición. Una variable local de tipo numérico necesita recibir un valor antes de leerse. Ese valor puede venir de la entrada o de una inicialización explícita.

`auto` deduce el tipo a partir de la expresión de la derecha. Es cómodo para iteradores y resultados de búsqueda. `const` permite declarar un valor que se consulta durante el cálculo y se mantiene fijo.

Funciones y operaciones

Uso	Resultado o efecto
`int cantidad = 0;`	Entero, habitualmente de 32 bits y hasta aproximadamente 2 mil millones
`long long suma = 0;`	Entero de al menos 64 bits, hasta aproximadamente 9 trillones
`double promedio = 0.0;`	Valor real aproximado
`char letra = 'A';`	Un carácter
`bool posible = true;`	Valor lógico `true` o `false`
`const int LIMITE = 1000;`	Entero cuyo valor queda fijo
`auto x = expresion;`	Crea `x` con el tipo deducido de la expresión
`1LL * a * b`	Hace el producto desde el principio en `long long`
`static_cast<int>(valor)`	Convierte al tipo indicado, que debe poder representar el valor que necesitás

Para los límites exactos de un tipo podés usar `numeric_limits<T>::max()` y `numeric_limits<T>::lowest()`.

Ejemplos

Dar un valor a cada variable

```
int a = 0, b = 0, c = 0;
long long suma = 0;
bool encontrado = false;
```

En `int a, b, c = 0;`, solo `c` queda inicializada. También podés declarar sin inicializar una variable que vas a cargar inmediatamente con `cin`, antes de usarla.

Ampliar antes del producto

```
int ancho = 100000;
int alto = 100000;
long long area = 1LL * ancho * alto; // 10000000000
```

El tipo de la variable a la izquierda no cambia el tipo del cálculo de la derecha. La promoción se hace antes de multiplicar. El resultado final también debe caber en `long long`.

Copiar y convertir

```
int cantidad = 7;
auto copia = cantidad; // copia es int y vale 7
double promedio = 3.75;
int parteEntera = static_cast<int>(promedio); // 3
```

La conversión de un real representable a entero descarta su parte decimal hacia cero. Para redondear se usan las operaciones de la ficha de reales.

Entrada y salida

Descripción

`cin` lee los datos de la entrada estándar y `cout` escribe en la salida estándar. El enunciado define el orden y el tipo de cada dato. Los espacios y saltos de línea separan los datos que leés con `>>`.

Usá `cin >> texto` para una palabra y `getline` para una línea que puede tener espacios. Al pasar de una lectura con `>>` a una línea siguiente, hay que consumir el final de línea pendiente. Si la entrada termina al llegar al fin del archivo, la propia lectura puede controlar el ciclo.

La salida debe respetar el formato pedido, incluidos espacios, saltos de línea y mayúsculas. `\n` escribe un salto de línea. En la consola local puede ser necesario cerrar la entrada para indicar que llegó el fin del archivo.

Funciones y operaciones

Uso	Parámetros	Resultado o efecto
`cin >> a >> b`	Variables donde guardar	Lee los dos datos en ese orden
`getline(cin, linea)`	Flujo y string de destino	Lee hasta el salto de línea, lo consume y guarda el texto anterior
`cin.ignore(numeric_limits<streamsize>::max(), '\n')`	Máximo a descartar y separador	Descarta hasta el próximo salto de línea, incluido
`cout << valor`	Dato a imprimir	Escribe su representación
`cout << '\n'`	Carácter de salto de línea	Termina la línea actual
`while (cin >> x)`	Lectura que controla el ciclo	Repite mientras se puede leer otro valor

Ejemplos

Leer números y una palabra

```
int n;
long long costo;
string palabra;
cin >> n >> costo >> palabra;
```

Leer una línea después de un número

```
int n;
cin >> n;
cin.ignore(numeric_limits<streamsize>::max(), '\n');
string linea;
getline(cin, linea);
```

Este formato supone que la línea que buscás empieza después de la de `n`. Para leer una línea desde el inicio de la entrada, usá directamente `getline(cin, linea)`.

Procesar una cantidad de casos

```
int casos;
cin >> casos;
for (int caso = 0; caso < casos; ++caso) {
    int a, b;
    cin >> a >> b;
    cout << a + b << '\n';
}
```

Leer hasta fin de archivo

```
long long x;
long long suma = 0;
while (cin >> x) {
    suma += x;
}
cout << suma << '\n';
```

Separar una secuencia con espacios

```
vector<int> respuesta = {2, 5, 7};
for (int i = 0; i < static_cast<int>(respuesta.size()); ++i) {
    if (i > 0) cout << ' ';
    cout << respuesta[i];
}
cout << '\n';
// Salida: 2 5 7
```

Condiciones y decisiones

Descripción

Una condición es una expresión que se interpreta como verdadera o falsa. Permite elegir qué instrucciones ejecutar según los datos. Podés guardarla en una variable `bool` cuando darle un nombre aclara qué estás preguntando.

Una cadena de `if`, `else if` y `else` elige un único camino. Varios `if` independientes permiten hacer más de una actualización. Elegí según si las acciones pueden cumplirse a la vez.

`&&` y `||` evalúan de izquierda a derecha y pueden terminar antes de evaluar lo que sigue. Ese comportamiento sirve para consultar un índice o un iterador recién después de comprobar que es válido.

Funciones y operaciones

Uso	Significado
`a == b` / `a != b`	Igualdad / diferencia
`a < b`, `a <= b`, `a > b`, `a >= b`	Comparaciones de orden
`a && b`	Ambas condiciones verdaderas
`` `a ``	
`!a`	Invierte el valor lógico
`if (condicion) { ... }`	Ejecuta el bloque si se cumple
`else if (...)` / `else`	Otros caminos de la misma decisión
`condicion ? valorA : valorB`	Elige una de dos expresiones

Ejemplos

Elegir un camino

```
int a = 7, b = 4;
string resultado;
if (a < b) {
    resultado = "MENOR";
} else if (a == b) {
    resultado = "IGUAL";
} else {
    resultado = "MAYOR";
}
// resultado queda "MAYOR"
```

Consultar límites antes de acceder

```
string s = "TAP";
int i = 1;
int n = static_cast<int>(s.size());
bool iguales = i >= 0 && i + 1 < n && s[i] == s[i + 1];
// iguales vale false
```

Si los índices quedan fuera, el último término no se evalúa. Para expresar que `x` está entre `l` y `r`, escribí `l <= x && x <= r`.

Dos actualizaciones independientes

```
int menor = 5, mayor = 5;
int x = 2;
if (x < menor) menor = x;
if (x > mayor) mayor = x;
// menor vale 2 y mayor vale 5
```

Elegir un valor con ternario

```
int n = 8;
char respuesta = (n % 2 == 0) ? 'S' : 'N'; // 'S'
```

Cuentas enteras y restos

Descripción

Las operaciones entre enteros producen resultados enteros. En una división, la parte fraccionaria se descarta hacia cero. `%` calcula el resto. Estas operaciones sirven para repartir en grupos completos, consultar divisibilidad y separar una cantidad en unidades más grandes.

Para redondear una división positiva hacia arriba, podés usar el cociente y sumar uno si queda resto. Para conservar un resultado entre 0 y m - 1, calculá el resto y sumá m cuando sea negativo.

El tamaño de cada operación intermedia importa. Si un producto puede superar `int`, ampliá uno de sus operandos antes de multiplicar. Aplicar `%` después del producto no corrige un desbordamiento que ya ocurrió.

Funciones y operaciones

Uso	Condición	Resultado
`a / b`	b distinto de cero y cociente representable	Cociente entero hacia cero
`a % b`	División válida	Resto, con el signo de a cuando no es cero
`a / b + (a % b != 0)`	a >= 0 y b > 0	Techo de la división
`abs(x)`	El valor positivo cabe en el tipo	Valor absoluto
`1LL * a * b`	Producto representable en `long long`	Producto en un tipo amplio
`(a + b) % m`	m > 0 y suma representable	Resto de la suma

Ejemplos

Cociente, resto y techo

```
long long a = 11, b = 4;
long long cociente = a / b; // 2
long long resto = a % b; // 3
long long techo = a / b + (a % b != 0); // 3
```

Esta fórmula evita sumar `a + b - 1`, que también podría exceder el tipo.

Normalizar un resto negativo

```
long long x = -3, m = 5;
long long r = x % m;
if (r < 0) r += m;
// r vale 2
```

Calcular un producto modular

```
long long mod = 1000000007LL;
long long a = mod - 1, b = mod - 2;
long long producto = (a * b) % mod; // 2
```

Aquí los operandos reducidos tienen un producto que cabe en 64 bits. Con un módulo mayor hay que volver a comprobar esa cota.

Contar pares de elementos distintos

Para elegir dos posiciones diferentes entre n, sin importar su orden, la cantidad es `n * (n - 1) / 2`. La receta divide primero uno de los factores para reducir el tamaño del producto intermedio.

```
long long n = 100000;
long long a = n, b = n - 1;
if (a % 2 == 0) a /= 2;
else b /= 2;
long long pares = a * b; // 4999950000
```

El cálculo usa `n >= 1`. Dividí el factor par antes de multiplicar y comprobá que el producto final quepa. Si el resultado lleva módulo, reducí los factores después de esa división entera.