RESUMEN

Acumuladores

Siempre se deben inicializar en cero.

División entre enteros

En C++, cuando dividimos int/int, el resultado también es int. Eso implica que el resultado se redondea descartando los decimales.
Si queremos que el resultado tenga decimales se puede hacer un cast de alguno de los dos valores:

```
float promedio;
int notas[] = {8, 7, 6};
const int CANTIDAD_NOTAS= sizeof(notas) / sizeof(notas[0]);

int acumulador = 0;

for(int i = 0; i < CANTIDAD_NOTAS; i++)
{
	acumulador += notas[i];
}

promedio = acumulador / CANTIDAD_NOTAS; // incorrecto, int/int redondea
promedio = acumulador / (float)CANTIDAD_NOTAS; // correcto, int/float
```

Promedio

Es la sumatoria de un conjunto de elementos, dividido la cantidad de elementos. Si estamos calculando el promedio de valores cargados en un array, se divide por la cantidad de elementos cargados, no por el tamaño del array. Siempre se debe validar que el divisor no sea cero.

```
//declaro array
const int TAMANIO_ARRAY = 10000;
float precios[TAMANIO_ARRAY];
int elementos_cargados = 0;
float precio_promedio;

//cargo elementos
precios[elementos_cargados++] = 3548,945;
precios[elementos_cargados++] = 15612,45;
precios[elementos_cargados++] = 985,5687;
```

```
float acumulador = 0.0;

//for(int i = 0; i < TAMANIO_ARRAY ; i++)// incorrecto, accede a posiciones no inicializadas
for(int i = 0; i < elementos_cargados; i++)// correcto
{
	acumulador += precios[i];
}

//precio_promedio = acumulador / (float)elementos_cargados ; // incorrecto, no valida que el divisor sea mayor a cero

if(TAMANIO_ARRAY > 0) // incorrecto, no valida la cantidad de elementos cargados
{
	//precio_promedio = acumulador / TAMANIO_ARRAY; // incorrecto, accede a posiciones no inicializadas
}

if(elementos_cargados > 0) // correcto, valida que el divisor no sea cero
{
	precio_promedio = acumulador / (float)elementos_cargados ; // correcto
}
```

Pasaje por referencia

Cuando pasamos un parámetro por referencia la función llamada puede modificar ese valor. Si es un contador hay que definir dónde se inicializa, en la función que llama o en la llamada.

Tipo int

En C++, el tipo **int** tiene al menos **16 bits** de ancho, lo que garantiza al menos 65.536 valores distintos. El rango mínimo garantizado es:

- unsigned int: de 0 a 65.535
- signed int: de -32.767 a 32.767

Sin embargo, en la mayoría de las plataformas modernas (x86, x64), int suele tener 32 bits, con un rango de:

- unsigned int: 0 a 4.294.967.295
- signed int: -2.147.483.648 a 2.147.483.647

Por cuestiones de **portabilidad** es recomendable asumir que el int es de 16 bits porque es el ancho garantizado por el lenguaje C++:

```
unsigned long dni = 45123123; // funciona en todas las plataformas

int dni = 45123123; // excede el rango 0-65535, puede fallar en algunas plataformas
```

Búsqueda secuencial

Debe finalizar cuando encuentra el id, y solo deben leerse los elementos cargados:

```
int busqueda_secuencial(const int array[], int id, int size)
{
	for (int i = 0; i < size; ++i)
	{
		if (array[i] == id)
			return i;  // finaliza cuando encuentra el id
	}
	return -1;
}
```

Tamaño de array vs cantidad de elementos cargados

En un array no se deben leer los elementos que no fueron inicializados.

```
//declaro array
const int TAMANIO_ARRAY 10000;
long serial_numbers [TAMANIO_ARRAY];
int elementos_cargados = 0;

//cargo elementos
serial_numbers[elementos_cargados++] = 3548945;
serial_numbers[elementos_cargados++] = 1561245;
serial_numbers[elementos_cargados++] = 9855687;
```

En este ejemplo bajo ninguna circunstancia se debe acceder a posiciones mayores a (TAMANIO_ARRAY-1). Dado que solo se cargaron 3 posiciones (0, 1 y 2), el resto no contiene datos y es incorrecto buscar datos en las posiciones no inicializadas. Los elementos "no inicializados" pueden contener basura, por lo que jamás deben usarse en búsquedas o cálculos. Si realizamos una búsqueda solo se deben recorrer los elementos cargados:

```
busqueda_secuencial(serial_numbers, 932422, elementos_cargados);  //Correcto
busqueda_secuencial(serial_numbers, 932422, TAMANIO_ARRAY);  //Incorrecto
```

Swap

Si estamos ordenando un array de structs, entonces la variable aux en el swap tiene que ser un struct. El swap debe copiar la estructura completa, no solo un campo, por eso el tipo de la variable aux debe ser el mismo que los elementos del array que estamos ordenando. Ejemplo:

```
struct Conductor {
    int id;
    string nombre;
};
Conductor lista[3] = {
    {102, "Ana"},
    {101, "Luis"},
    {103, "Zoe"}
};

Conductor aux; //correcto
aux = lista[0]; //correcto
lista[0] = lista[1]; //correcto
lista[1] = aux; //correcto

int aux; //incorrecto
aux = lista[0].id; //incorrecto
lista[0].id = lista[1].id; //incorrecto
lista[1].id = aux; //incorrecto
```

Valores enteros mapeados a strings

Cuando tenemos valores enteros correlativos que están asociados a nombres, por ejemplo:
1 -> “Centro”
2 -> “Norte”
3 -> “Sur”

Dado el valor entero de la izquierda, para obtener el string asociado no es lo mejor hacerlo con switch-case, en cambio se puede hacer con un simple array. En algunos casos puede ser necesario validar que el valor de la izquierda esté dentro del rango válido.

```
string nombres_zonas[] = {“”, “Centro”, “Norte”, “Sur”};
int codigo_zona;

codigo_zona = 2;
cout << nombres_zonas[codigo_zona];

codigo_zona = 1;
cout << nombres_zonas[codigo_zona];

codigo_zona = 3;
cout << nombres_zonas[codigo_zona];
```