VECTORES

1. Creación y Modificación

- `vector<T> v;` → Declara un vector vacío.
- `vector<T> v(n);` → Declara un vector de tamaño `n`, inicializado con valores por defecto.
- `vector<T> v(n, val);` → Declara un vector de tamaño `n`, inicializado con `val`.
- `vector<T> v2(v1);` → Crea una copia de otro vector `v1`.
- `vector<T> v2(v1.begin(), v1.begin() + k);` → Crea un vector con los primeros `k` elementos de `v1`.

---

2. Agregar y Eliminar Elementos

- `v.push_back(val);` → Agrega `val` al final del vector.
- `v.pop_back();` → Elimina el último elemento.
- `v.insert(it, val);` → Inserta `val` en la posición `it` (iterador).
- `v.insert(it, n, val);` → Inserta `n` copias de `val` en `it`.
- `v.erase(it);` → Elimina el elemento en la posición `it`.
- `v.erase(it1, it2);` → Elimina los elementos en el rango `[it1, it2)`.
- `v.clear();` → Borra todos los elementos del vector.

---

3. Acceso a Elementos

- `v[i];` → Accede al elemento en la posición `i`.
- `v.at(i);` → Similar a `v[i]`, pero con verificación de límites.
- `v.front();` → Devuelve el primer elemento.
- `v.back();` → Devuelve el último elemento.
- `v.data();` → Devuelve un puntero al primer elemento del vector.

---

4. Información del Vector

- `v.size();` → Devuelve el número de elementos.
- `v.capacity();` → Devuelve la capacidad actual (memoria reservada).
- `v.empty();` → Devuelve `true` si el vector está vacío.
- `v.max_size();` → Devuelve el número máximo de elementos posibles.

---

5. Ordenación y Búsqueda

- `sort(v.begin(), v.end());` → **Ordena el vector en orden ascendente.**
- `sort(v.begin(), v.end(), greater<T>());` → **Ordena en orden descendente.**
- `reverse(v.begin(), v.end());` → **Invierte el orden de los elementos.**
- `binary_search(v.begin(), v.end(), val);` → **Devuelve `true` si `val` está en el vector (requiere ordenación previa).**
- `find(v.begin(), v.end(), val);` → **Devuelve un iterador al primer elemento que coincide con `val`.**

---

6. Otras Operaciones Útiles

- `v.swap(v2);` → **Intercambia los elementos de `v` con `v2`.**
- `v.resize(n);` → **Cambia el tamaño del vector a `n`.**
- `v.resize(n, val);` → **Cambia el tamaño del vector a `n`, rellenando con `val` si es necesario.**
- `v.shrink_to_fit();` → **Reduce la capacidad del vector para ajustarse a su tamaño real.**

Ejemplo de Uso

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

int main() {
    vector<int> v = {5, 2, 8, 1};

    v.push_back(10); // Agrega 10 al final
    sort(v.begin(), v.end()); // Ordena el vector

    cout << "Tamaño: " << v.size() << endl;
    cout << "Primer elemento: " << v.front() << endl;
    cout << "Último elemento: " << v.back() << endl;

    return 0;
}
```

STRINGS

1. Creación y Modificación

- `string s;` → Declara un string vacío.
- `string s = "Hola";` → Declara un string con contenido.
- `string s(n, 'x');` → Crea un string de `n` caracteres con `'x'`.
- `string s1(s2);` → Crea una copia de otro string `s2`.
- `s += " mundo";` → Concatenación rápida.

---

2. Acceso a Caracteres

- `s[i];` → Accede al carácter en la posición `i`.
- `s.at(i);` → Similar a `s[i]`, pero con verificación de límites.
- `s.front();` → Devuelve el primer carácter.
- `s.back();` → Devuelve el último carácter.
- `s.data();` → Devuelve un puntero al primer carácter del string.

---

3. Información del String

- `s.size();` / `s.length();` → Devuelve el número de caracteres.
- `s.empty();` → Devuelve `true` si el string está vacío.
- `s.capacity();` → Devuelve la memoria reservada.
- `s.max_size();` → Devuelve el número máximo de caracteres posibles.

---

4. Modificación

- `s.clear();` → Borra todo el contenido.
- `s.insert(pos, str);` → Inserta `str` en la posición `pos`.
- `s.erase(pos, len);` → Elimina `len` caracteres desde `pos`.
- `s.replace(pos, len, str);` → Reemplaza `len` caracteres desde `pos` con `str`.
- `s.push_back('c');` → Agrega un carácter al final.
- `s.pop_back();` → Elimina el último carácter.

---

5. Búsqueda

- `s.find("mundo");` → Devuelve la posición de la primera aparición de `"mundo"` o `npos` si no se encuentra.
- `s.rfind("mundo");` → Similar a `find()`, pero busca desde el final.
- `s.find_first_of("aeiou");` → Encuentra la primera vocal en `s`.
- `s.find_last_of("aeiou");` → Encuentra la última vocal en `s`.
- `s.find_first_not_of("aeiou");` → Encuentra la primera consonante.
- `s.find_last_not_of("aeiou");` → Encuentra la última consonante.

---

6. Comparación

- `s1 == s2;` → Devuelve `true` si `s1` y `s2` son iguales.
- `s1 != s2;` → Devuelve `true` si `s1` y `s2` son diferentes.
- `s1 < s2;` → Compara lexicográficamente.
- `s1.compare(s2);` → Devuelve `<0` si `s1` < `s2`, `0` si son iguales, `>0` si `s1` > `s2`.

---

7. Substrings

- `s.substr(pos, len);` → Extrae un substring desde `pos` de longitud `len`.

```
string s = "Hola mundo";
string sub = s.substr(5, 5); // "mundo"
```

---

8. Conversión

- `stoi(s);` → Convierte `s` a `int`.
- `stol(s);` → Convierte `s` a `long`.
- `stoll(s);` → Convierte `s` a `long long`.
- `stof(s);` → Convierte `s` a `float`.
- `stod(s);` → Convierte `s` a `double`.
- `to_string(x);` → Convierte `x` a `string`.
- `transform(s1.begin(),s1.end(),s1.begin(), :: tolower);` → Convierte todo `s1` a `minúsculas`.
- transform(s2.begin(), s2.end() ,s2.begin(), :: tolower); → Conviertetodo `s2` a `mayúsculas`.

```
int num = stoi("1234");  // 1234
string s = to_string(567);  // "567"
```

---

9. Transformaciones

- `s.append(str);` → Agrega `str` al final de `s`.
- `s.insert(pos, str);` → Inserta `str` en `pos`.
- `s.erase(pos, len);` → Elimina `len` caracteres desde `pos`.
- `s.replace(pos, len, str);` → Reemplaza `len` caracteres desde `pos` con `str`.

```
string s = "Hola";
s.replace(1, 2, "ey"); // "Heya"
```

---

10. Iteradores

- `s.begin();` → Iterador al inicio.
- `s.end();` → Iterador al final.
- `s.rbegin();` → Iterador inverso al final.
- `s.rend();` → Iterador inverso al inicio.

Ejemplo Completo

```
string s = "Hola mundo";

cout << "Tamaño: " << s.size() << endl;
cout << "Subcadena: " << s.substr(5, 5) << endl;  // "mundo"

s.replace(5, 5, "amigo");  // "Hola amigo"
cout << "Después de reemplazo: " << s << endl;

reverse(s.begin(), s.end());
cout << "Invertido: " << s << endl;  // "ogima aloH"
```

SET

1. Creación y Declaración

```
set<int> s;  // Crea un set vacío
set<int> s = {3, 1, 4, 1, 5, 9, 2};  // Crea un set con elementos únicos
set<int, greater<int>> s;  // Set en orden descendente
```

---

2. Inserción y Eliminación

- `s.insert(x);` → Inserta `x` en el set.
- `s.emplace(x);` → Inserta `x` (más eficiente que `insert`).
- `s.erase(x);` → Elimina el elemento `x`.
- `s.erase(it);` → Elimina el elemento en la posición del iterador `it`.
- `s.clear();` → Vacía el set.

```
s.insert(10);  // {10}
s.insert(5);   // {5, 10}
s.insert(5);   // No se inserta de nuevo porque los sets no permiten duplicados
s.erase(5);    // {10}
```

---

3. Búsqueda

- `s.find(x);` → Devuelve un iterador al elemento `x`, o `s.end()` si no existe.
- `s.count(x);` → Devuelve `1` si `x` está en el set, `0` si no.

```
if (s.find(10) != s.end()) cout << "Encontrado\n";
if (s.count(5)) cout << "5 está en el set\n";
```

---

4. Tamaño e Información

- `s.size();` → Número de elementos en el set.
- `s.empty();` → `true` si el set está vacío.

```
cout << "Tamaño: " << s.size() << endl;
```

---

5. Mínimo y Máximo

- `s.begin();` → **Primer elemento (mínimo si está en orden ascendente).**
- `s.rbegin();` → **Último elemento (máximo si está en orden ascendente).**
- `s.end();` → **Devuelve iterador al final (no es un elemento válido).**
- `s.rend();` → **Devuelve iterador al inicio en reversa.**

```
cout << "Mínimo: " << *s.begin() << endl;
cout << "Máximo: " << *s.rbegin() << endl;
```

---

6. Rango de Búsqueda

- `s.lower_bound(x);` → **Devuelve el primer elemento** `≥ x`**.**
- `s.upper_bound(x);` → **Devuelve el primer elemento** `> x`**.**

```
auto it = s.lower_bound(4);  // Primer elemento >= 4
if (it != s.end()) cout << "Lower bound: " << *it << endl;

auto it2 = s.upper_bound(4);  // Primer elemento > 4
if (it2 != s.end()) cout << "Upper bound: " << *it2 << endl;
```

---

7. Recorrido

Iteradores

```
for (auto it = s.begin(); it != s.end(); it++)
    cout << *it << " ";
```

For-each

```
for (int x : s)
    cout << x << " ";
```

Recorrido Inverso

```
for (auto it = s.rbegin(); it != s.rend(); it++)
    cout << *it << " ";
```

Ejemplo Completo

```
    set<int> s = {4, 2, 8, 5, 3, 7};

    s.insert(10);
    s.erase(3);

    cout << "Elementos del set: ";
    for (int x : s) cout << x << " ";
    cout << endl;

    if (s.find(5) != s.end()) cout << "5 está en el set\n";

    auto lb = s.lower_bound(4);
    if (lb != s.end()) cout << "Lower bound de 4: " << *lb << endl;
```

BÚSQUEDA Y ORDENAMIENTO

1. `sort` (Ordenar un arreglo o contenedor)

La función `sort` se usa para ordenar elementos en un rango.

Sintaxis básica:

```
sort(inicio, fin);
```

- **Ordena de menor a mayor por defecto (ascendente).**
- **`inicio` y `fin` son iteradores o punteros al rango que quieres ordenar.**
- **`fin` NO está incluido en la ordenación.**

Ejemplo en un vector:

```
    vector<int> v = {4, 2, 8, 5, 1};

    sort(v.begin(), v.end());  // Orden ascendente

    for (int x : v) cout << x << " ";  // Salida: 1 2 4 5 8
}
```

Orden descendente:

```
sort(v.begin(), v.end(), greater<int>());
```

Orden con función personalizada:

```
bool comparar(int a, int b) {
    return a > b;  // Ordena de mayor a menor
}

sort(v.begin(), v.end(), comparar);
```

---

2. `binary_search` (Búsqueda Binaria)

Fórmula segura para evitar overflow:

```
int mid = l + (r - l) / 2;
```

- 🔹 **¿Por qué?**
  - `(r - l) / 2` **calcula la mitad del segmento.**
  - `l +` **evita la suma directa de** `l + r`**, que podría desbordarse si** `l` **y** `r` **son grandes.**

Ejemplo en Búsqueda Binaria Normal:

```
while (l <= r) {
    int mid = l + (r - l) / 2;

    if (arr[mid] == target)
        return mid;
    else if (arr[mid] < target)
        l = mid + 1;
    else
        r = mid - 1;
}
```

Se usa para buscar un elemento en un rango ORDENADO.

Sintaxis:

```
binary_search(inicio, fin, valor);
```

- **Devuelve `true` si el `valor` está en el rango, `false` si no.**
- **El rango debe estar ORDENADO.**

Ejemplo en un vector:

```
    vector<int> v = {1, 3, 5, 7, 9};

    if (binary_search(v.begin(), v.end(), 5))
        cout << "5 está en el vector" << endl;
    else
        cout << "5 NO está en el vector" << endl;
```

MAX Y MIN

`max` y **`min`** están en el encabezado `<algorithm>` o `<bits/stdc++.h>` (si usas ese).

Sintaxis básica:

```
int mayor = max(a, b);
int menor = min(a, b);
```

Para más de dos valores:
Si tienes tres o más valores:

```
int x = 15, y = 8, z = 25;
int mayor = max({x, y, z});  // 25
int menor = min({x, y, z});  // 8
```

Esto funciona desde C++11 gracias a las listas de inicialización (`{}`).

En contenedores (vectores, arrays):

```
vector<int> v = {5, 3, 9, 1, 7};

int max_val = *max_element(v.begin(), v.end());  // 9
int min_val = *min_element(v.begin(), v.end());  // 1
```

¿Necesarias?

- Si solo comparas dos valores, podrías usar un simple `if`, pero `max` y `min` son más elegantes y eficientes.
- Para rangos o múltiples valores, son casi imprescindibles.

BITS

Operaciones bit a bit (Bitwise operators)

Son operadores básicos para trabajar con bits:

Operador	Descripción	Ejemplo
`&`	AND bit a bit	`a & b`
`` ` ``	`` ` ``	OR bit a bit
`^`	XOR bit a bit	`a ^ b`
`~`	NOT bit a bit	`~a`
`<<`	Desplazamiento a la izquierda	`a << n`
`>>`	Desplazamiento a la derecha	`a >> n`

Funciones de `bitset`

Función	Descripción
`set(pos)`	Activa el bit en la posición `pos`.
`reset(pos)`	Apaga el bit en la posición `pos`.
`flip(pos)`	Invierte el bit en la posición `pos`.
`count()`	Cuenta los bits encendidos (1s).
`size()`	Devuelve el tamaño del bitset.
`test(pos)`	Devuelve `true` si el bit en `pos` es 1.

Funciones en `<bit>`

Función	Descripción
`popcount(x)`	Cuenta los bits en 1 de `x`.
`countr_zero(x)`	Cuenta ceros a la derecha.
`countl_zero(x)`	Cuenta ceros a la izquierda.