



Vectores cont

Universidad Católica Boliviana

MSc, José Jesús Cabrera Pantoja

Outline

- Introduccion a vectores
- Mas sobre vectores :D

Vector

- Para trabajar con un vector, incluya la librería `<vector>`.
- Al declarar una variable, los corchetes angulares indican el tipo de elementos que estarán contenidos en el contenedor.
- Por ejemplo,
 - **`vector<int>`**
 - **`vector<string>`**
 - **`vector<char>`**



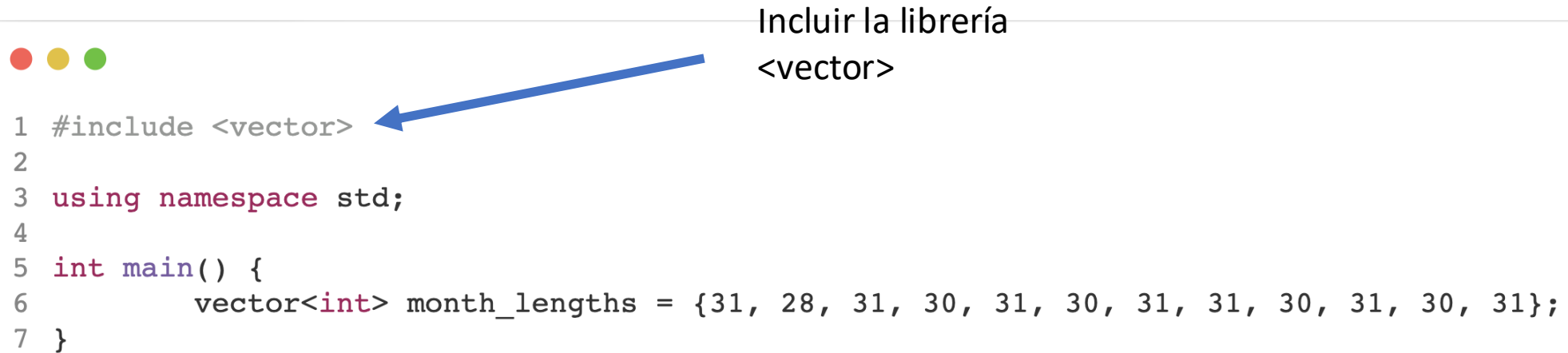
Vector

- Imagina que quieres almacenar en un programa el número de días de cada mes del año.
- Un vector será útil para esto.
- La cantidad de días es un número, es decir, almacenaremos valores del tipo **int** en el **vector**.
- Declaremos la variable **month_lengths**. Su tipo es **vector<int>**.
- El contenido del vector se escribe entre llaves durante la inicialización:



Vector

- El contenido del vector se escribe entre llaves durante la inicialización:



```
1 #include <vector>
2
3 using namespace std;
4
5 int main() {
6     vector<int> month_lengths = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
7 }
```


Incluir la librería
<vector>

Vector

- El contenido del vector se escribe entre llaves durante la inicialización:



```
1 #include <vector>
2
3 using namespace std;
4
5 int main() {
6     vector<int> month_lengths = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
7 }
```



Nombre de
la variable

Vector

- El contenido del vector se escribe entre llaves durante la inicialización:



```
1 #include <vector>
2
3 using namespace std;
4
5 int main() {
6     vector<int> month_lengths = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
7 }
```

Tipo de la
variable

Nombre de
la variable

Vector

- El contenido del vector se escribe entre llaves durante la inicialización:



```
1 #include <vector>
2
3 using namespace std;
4
5 int main() {
6     vector<int> month_lengths = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
7 }
```

Tipo de los
elementos
del vector


Nombre de
la variable

Vector

- El contenido del vector se escribe entre llaves durante la inicialización:



```
1 #include <vector>
2
3 using namespace std;
4
5 int main() {
6     vector<int> month_lengths = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
7 }
```



Elementos del vector, en este caso enteros (int)

Vector

- Cualquier vector tiene un tamaño: la cantidad de valores en el contenedor.
- El tamaño puede ser de uno o diez millones: C++ puede funcionar de manera eficiente incluso con vectores muy grandes.
- Supongamos también un vector de tamaño 0: se llama **vacío**.

Vector

- Cualquier vector tiene un tamaño: la cantidad de valores en el contenedor.



```
1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 int main() {
7     vector<int> month_lengths = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
8     cout << month_lengths.size() << endl;
9 }
```

- Cual es la salida del programa? 12!

Vector

- Cada elemento del vector tiene un **índice**, un número de serie desde el comienzo del vector.
- Además, los elementos están numerados desde **cero**: el índice del primer elemento es **cero** y el índice del último elemento **es uno menos que el tamaño del vector**.

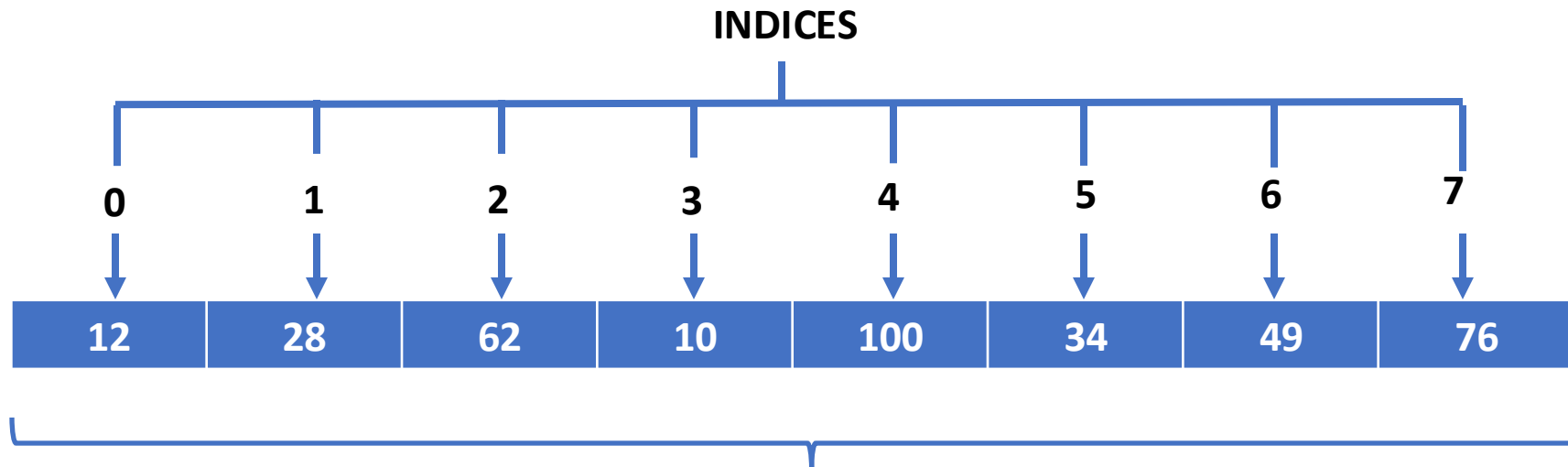


```
1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 int main() {
7     vector<int> month_lengths = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
8     cout << month_lengths.size() << endl;
9 }
```

Vector




```
1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 int main() {
7     vector<int> some_numbers = {12, 28, 62, 10, 100, 34, 49, 76};
8     cout << some_numbers.size() << endl;
9 }
```



- TAMANO DEL VECTOR = 8

Vector


- Para referirse a un elemento vectorial específico, debe escribir el índice entre corchetes.
- El índice puede ser un número, una variable o una expresión numérica arbitraria:



```
1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 int main() {
7     vector<int> month_lengths = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
8     cout << month_lengths.size() << endl;
9
10    int month_index;
11    cin >> month_index;
12
13    // Pedir el elemento del vector month-lengths con el indice month_index
14    cout << "There are "s << month_lengths[month_index] << " days"s << endl;
15 }
```


Crear vectores

- Existen varias formas de crear e inicializar un vector:




```
1 #include <iostream>
2 #include <vector>
3 using std::vector;
4 using std::cout;
5
6 int main() {
7     // Three ways of declaring and initializing vectors.
8     vector<int> v_1{0, 1, 2};
9     vector<int> v_2 = {3, 4, 5};
10    vector<int> v_3;
11    v_3 = {6};
12    cout << "Everything worked!" << "\n";
13 }
```

Crear vectores

- Un vector vacío no tiene un último elemento. Cuando vuelva a llamar para solicitarlo, el programa se comportará de manera impredecible. Además, C++ no informará un error por adelantado. Depende del programador asegurarse de que el vector no esté vacío antes de volver a llamar.
- Para verificar si un vector está vacío, use el método `empty`. Por ejemplo, tiene un vector que almacena los nombres de animales perdidos. Mostrar el apodo de este último le permitirá volver. Pero primero verifica si el vector no está vacío.

Empty

- Para verificar si un vector está vacío, use el método `empty`. Por ejemplo, tiene un vector que almacena los nombres de animales perdidos. Mostrar el apodo de este último le permitirá volver. Pero primero verifica si el vector no está vacío.

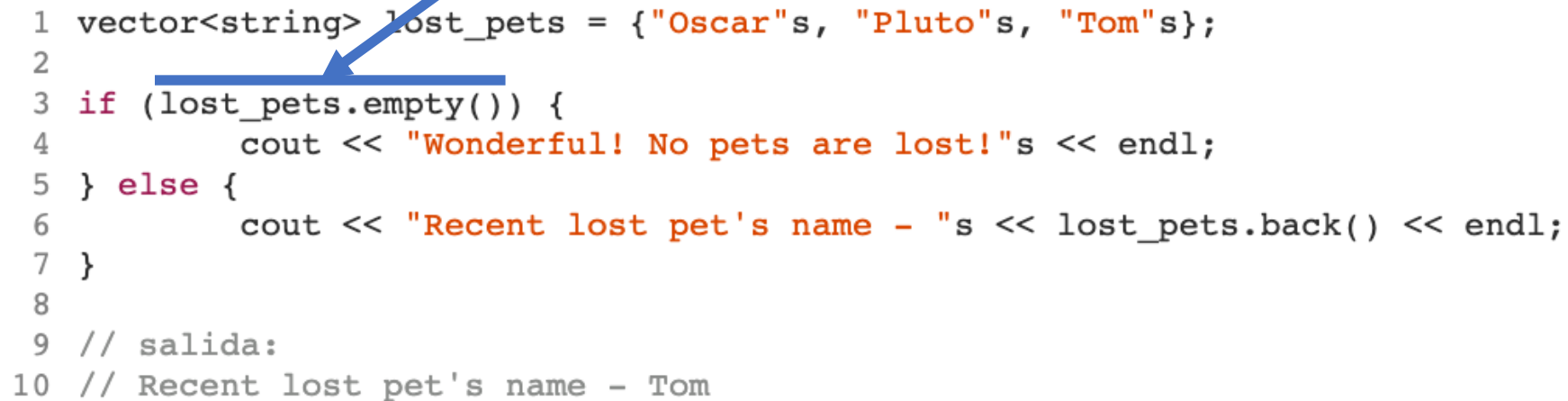


```
1 vector<string> lost_pets = {"Oscar"s, "Pluto"s, "Tom"s};
2
3 if (lost_pets.empty()) {
4     cout << "Wonderful! No pets are lost!"s << endl;
5 } else {
6     cout << "Recent lost pet's name - "s << lost_pets.back() << endl;
7 }
8
9 // salida:
10 // Recent lost pet's name - Tom
```

Empty

- Para verificar si un vector está vacío, use el método `empty`. Por ejemplo, tiene un vector que almacena los nombres de animales perdidos. Mostrar el apodo de este último le permitirá volver. Pero primero verifica si el vector no está vacío.

Verificamos si el vector esta vacio




```
1 vector<string> lost_pets = {"Oscar"s, "Pluto"s, "Tom"s};
2
3 if (lost_pets.empty()) {
4     cout << "Wonderful! No pets are lost!"s << endl;
5 } else {
6     cout << "Recent lost pet's name - "s << lost_pets.back() << endl;
7 }
8
9 // salida:
10 // Recent lost pet's name - Tom
```

Empty

- Hay otra forma de verificar el vector para ver si está vacío: llamar al `size` y comparar el resultado con 0.
- Pero usar `empty` es más corto y más expresivo: inmediatamente queda claro que lo que interesa es el vacío del vector.

Vectores 2D

- Un vector puede contener valores de cualquier tipo, incluidos otros vectores. Digamos que el refugio de animales tiene un horario de caminatas entre semana y queremos mantener ese horario. Un vector de vectores nos será de utilidad!:



```
1 // En el vector guardamos el grafico de paseos
2 // por dias en la semana
3 vector<vector<string>> walking_schedule = {
4     {"Ludwig"s, "Karl"s, "Johann"s},
5     {"Felix"s},
6     {"Karl"s, "Gustav"s, "Richard"s, "Wolfgang"s, "Johann"s},
7     {"Gustav"s, "Modest"s},
8     {"Sebastian"s, "Wolfgang"s, "Modest"s},
9     {"Rihard"s, "Sebastian"s, "Ludwig"s},
10    {"Felix"s},
11 };
```

Vectores 2D

- Acceder a los elementos de estos vectores se puede hacer con los corchetes:



```
1 // Seleccionamos del vector, el miercoles, y el elemento de este dia
2 // con el indice igual a 1
3 cout << "El miercoles el segundo en pasear es "s << walking_schedule[2][1] << endl;
4
5 // Salida: El miercoles el segundo en pasear es Gustav.
```


Vectores 2D

- Acceder a los elementos de estos vectores se puede hacer con los corchetes:



```
1 // Seleccionamos del vector, el miercoles, y el elemento de este dia
2 // con el indice igual a 1
3 cout << "El miercoles el segundo en pasear es "s << walking_schedule[2][1] << endl;
4
5 // Salida: El miercoles el segundo en pasear es Gustav.
```



La fila

Vectores 2D

- Acceder a los elementos de estos vectores se puede hacer con los corchetes:



```
1 // Seleccionamos del vector, el miercoles, y el elemento de este dia
2 // con el indice igual a 1
3 cout << "El miercoles el segundo en pasear es "s << walking_schedule[2][1] << endl;
4
5 // Salida: El miercoles el segundo en pasear es Gustav.
```



La Columna

Vectores 2D



```
1 // Seleccionamos del vector, el miercoles, y el elemento de este dia
2 // con el indice igual a 1
3 cout << "El miercoles el segundo en pasear es "s << walking_schedule[2][1] << endl;
4
5 // Salida: El miercoles el segundo en pasear es Gustav.
```



```
1 // En el vector guardamos el grafico de paseos
2 // por dias en la semana
3 vector<vector<string>> walking_schedule = {
4     {"Ludwig"s, "Karl"s, "Johann"s},
5     {"Felix"s},
6     {"Karl"s, "Gustav"s, "Richard"s, "Wolfgang"s, "Johann"s},
7     {"Gustav"s, "Modest"s},
8     {"Sebastian"s, "Wolfgang"s, "Modest"s},
9     {"Rihard"s, "Sebastian"s, "Ludwig"s},
10    {"Felix"s},
11 };
```

Vectores 2D



```
1 // Seleccionamos del vector, el miercoles, y el elemento de este dia
2 // con el indice igual a 1
3 cout << "El miercoles el segundo en pasear es "s << walking_schedule[2][1] << endl;
4
5 // Salida: El miercoles el segundo en pasear es Gustav.
```



```
1 // En el vector guardamos el grafico de paseos
2 // por dias en la semana
3 vector<vector<string>> walking_schedule = {
4     {"Ludwig"s, "Karl"s, "Johann"s},
5     {"Felix"s, "Karl"s, "Gustav"s, "Richard"s, "Wolfgang"s, "Johann"s},
6     {"Gustav"s, "Modest"s},
7     {"Sebastian"s, "Wolfgang"s, "Modest"s},
8     {"Rihard"s, "Sebastian"s, "Ludwig"s},
9     {"Felix"s},
10
11 };
```

La columna 1

La fila 2

For loops en Vectores

- Al igual que en las strings podemos pasarnos elemento por elemento de los vectores utilizando un loop



```
1 for (int month_index = 0; month_index < month_lengths.size(); ++month_index) {  
2     cout << "There are "s << month_lengths[month_index]  
3         << " days "s  
4         << " in month "s << (month_index + 1) << endl;  
5 }
```

For loops en Vectores

- Al igual que en las strings podemos pasarnos elemento por elemento de los vectores utilizando un loop
- Este ciclo se ve feo, se puede cometer errores, se repite varias veces la variable month_index, etc.



```
1 for (int month_index = 0; month_index < month_lengths.size(); ++month_index) {  
2     cout << "There are "s << month_lengths[month_index]  
3         << " days "s  
4         << " in month "s << (month_index + 1) << endl;  
5 }
```

For loops en Vectores

- Al igual que en las strings podemos pasarnos elemento por elemento de los vectores utilizando un loop
- Este ciclo se ve feo, se puede cometer errores, se repite varias veces la variable month_index, etc.



```
1 for (int month_index = 0; month_index < month_lengths.size(); ++month_index) {  
2     cout << "There are "s << month_lengths[month_index]  
3         << " days "s  
4         << " in month "s << (month_index + 1) << endl;  
5 }
```

- Buenas noticias! Hay una mejor forma de hacerlo 😊. Se llama **range-based for**

Range-based for

- El ejemplo utiliza un **range-based for** para recorrer el vector. La variable de **length** se ejecutará a través de todos sus elementos en secuencia:



```
1 cout << "Month lengths are:"s;  
2  
3 // Los valores de la variable length por orden  
4 // tendra los elementos del vector month_length  
5 for (int length : month_lengths) {  
6     cout << " "s << length;  
7 }  
8 cout << endl;
```

Range-based for

- El ejemplo utiliza un **range-based for** para recorrer el vector. La variable de **length** se ejecutará a través de todos sus elementos en secuencia:



```
1 cout << "Month lengths are:"s;  
2  
3 // Los valores de la variable length por orden  
4 // tendra los elementos del vector month_length  
5 for (int length : month_lengths) {  
6     cout << " "s << length;  
7 }  
8 cout << endl;
```

Variable
length que
guardara los
elementos
del vector

Range-based for

- El ejemplo utiliza un **range-based for** para recorrer el vector. La variable de **length** se ejecutará a través de todos sus elementos en secuencia:

```
1 cout << "Month lengths are:"s;
2
3 // Los valores de la variable length por orden
4 // tendra los elementos del vector month_length
5 for (int length : month_lengths) {
6     cout << " "s << length;
7 }
8 cout << endl;
```

Variable
length que
guardara los
elementos
del vector

Vector
utilizado

Range-based for

- El ejemplo utiliza un **range-based for** para recorrer el vector. La variable de **length** se ejecutará a través de todos sus elementos en secuencia:



```
1 cout << "Month lengths are:"s;  
2  
3 // Los valores de la variable length por orden  
4 // tendra los elementos del vector month_length  
5 for (int length : month_lengths) {  
6     cout << " "s << length;  
7 }  
8 cout << endl;
```

- Salida:



```
1 Month lengths are: 31 28 31 30 31 30 31 31 30 31 30 31
```

Agregar elementos al vector

- Al vector se le puede agregar elementos al final de forma rápida. Para ello se utiliza el método **push_back**



```
1 // Preguntamos la cantidad de animales con cin.
2 int pet_count;
3 cin >> pet_count;
4
5 // Creamos un vector vacio
6 vector<int> lost_pet_ages;
7
8 // El vector aún no se ha formado. Usamos el normal for para leer
9 // algunos números de cin y agregar los al vector lost_pet_ages.
10 for (int i = 0; i < pet_count; ++i) {
11     int age;
12     cin >> age;
13     lost_pet_ages.push_back(age);
14 }
```

Cambiar el tamaño

- Para cambiar el tamaño de un vector, utilice el método de **resize**. En su forma más simple, redimensionar toma un **argumento**, el nuevo tamaño del vector. Si el valor de este **argumento** es menor que el valor actual, los elementos adicionales se descartan:



```
1 vector<int> car_velocities = {60, 53, 67, 19, 77, 59};  
2 car_velocities.resize(4);  
3 // Ahora el tamaño del vector es 4, contenido: 60, 53, 67, 19
```

- **Argumento** – por ahora, tómelo como lo que esta entre paréntesis

Cambiar el tamaño

- Para cambiar el tamaño de un vector, utilice el método de **resize**. En su forma más simple, redimensionar toma un **argumento**, el nuevo tamaño del vector. Si el valor de este **argumento** es menor que el valor actual, los elementos adicionales se descartan:



```
1 vector<int> car_velocities = {60, 53, 67, 19, 77, 59};  
2 car_velocities.resize(4);  
3 // Ahora el tamaño del vector es 4, contenido: 60, 53, 67, 19
```

- **Argumento** – por ahora, tómelo como lo que esta entre paréntesis

Cambiar el tamaño

- Que pasa si el nuevo tamaño es mayor al actual?



```
1 vector<int> lost_pet_ages = {1, 8, 2, 1, 3, 10};  
2 lost_pet_ages.resize(10);  
3 // Ahora el tamaño del vector — 10, contenido: 1, 8, 2, 1, 3, 10, 0, 0, 0, 0
```

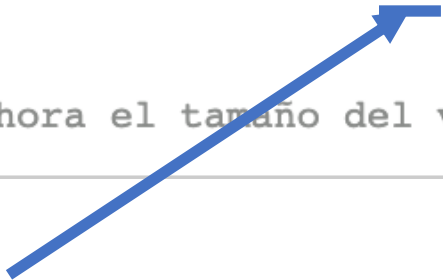
- Argumento — por ahora, tómelo como lo que esta entre paréntesis

Cambiar el tamaño

- Que pasa si el nuevo tamaño es mayor al actual?



```
1 vector<int> lost_pets_age = {1, 8, 2, 1, 3, 10};  
2 lost_pets_age.resize(10, -1);    // Se lee así: el vector debe tener un tamaño de 10.  
3                                // Si necesita agregar nuevos elementos,  
4                                // estos deben ser números -1.  
5 // Ahora el tamaño del vector es 10, contenido: 1, 8, 2, 1, 3, 10, -1, -1, -1, -1.
```



- Argumentos – nuevo tamaño del vector

Cambiar el tamaño

- Que pasa si el nuevo tamaño es mayor al actual?



```
1 vector<int> lost_pets_age = {1, 8, 2, 1, 3, 10};  
2 lost_pets_age.resize(10, -1);    // Se lee así: el vector debe tener un tamaño de 10.  
3                                // Si necesita agregar nuevos elementos,  
4                                // estos deben ser números -1.  
5 // Ahora el tamaño del vector es 10, contenido: 1, 8, 2, 1, 3, 10, -1, -1, -1, -1.
```

- Argumentos – nuevo tamaño del vector

Valores con
los que se
rellenara

Cambiar el tamaño

- Sirve para iniciar un vector con valores predeterminados!



```
1 size_t size;
2 cin >> size;
3 vector<string> lost_pet_names(size, "N/A"s);
4 // Vector del se rellena con N/A, y el tamaño esta dado por size
5 // Esta abreviatura se utiliza para reemplazar los datos que faltan.
6 // Puede configurarlos más tarde.
```

Cambiar el tamaño

- Ejemplo de uso:


```
1 vector<string> elephant_friends;
2
3 string friend_name;
4 cout << "Enter friend names or \"end\" to exit:"s << endl;
5
6 // Usamos un truco especial: leer desde cin como una condición de bucle.
7 // Esto mantendrá el ciclo en ejecución mientras las lecturas sean exitosas.
8 // Si los datos han terminado o no pueden ser leídos por otro
9 // motivo, el bucle se detendrá.
10 // Se puede usar una expresión similar con cin en una condición if. Significado
11 // la condición será verdadera si la lectura fue exitosa. Y si no, falso.
12 while (cin >> friend_name) {
13     // Finaliza el bucle si se lee end.
14     if (friend_name == "end"s) {
15         break;
16     }
17
18     // Agrega el nombre de lectura al vector.
19     elephant_friends.push_back(friend_name);
20 }
21
22 // Mostrar información sobre cuántos elementos se agregaron al vector
23 cout << "Elephant has "s << elephant_friends.size() << " friends."s << endl;
```

Keyword **auto**

- Hasta ahora se ha visto cómo almacenar tipos básicos y vectores que contienen esos tipos.
- Mientras practicaba la declaración de variables, en cada caso indicó el tipo de variable.
- Sin embargo, es posible que C++ haga una inferencia de tipo automática, usando la palabra clave **auto**.

Keyword **auto**

- Sin embargo, es posible que C++ haga una inferencia de tipo automática, usando la palabra clave **auto**.



```
1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 int main() {
7     auto i = 5;
8     auto v_6 = {1, 2, 3};
9     cout << "Variables declared and initialized without explicitly stating type!" << "\n";
10 }
```

Keyword **auto**

- Sin embargo, es posible que C++ haga una inferencia de tipo automática, usando la palabra clave **auto**.

```
1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 int main() {
7     auto i = 5;
8     auto v = {1, 2, 3};
9     cout << "Variables declared and initialized without explicitly stating type!" << "\n";
10 }
```

Le decimos al compilador que adivine que tipo de dato es esta variable (en este caso entero, int)

Keyword **auto**

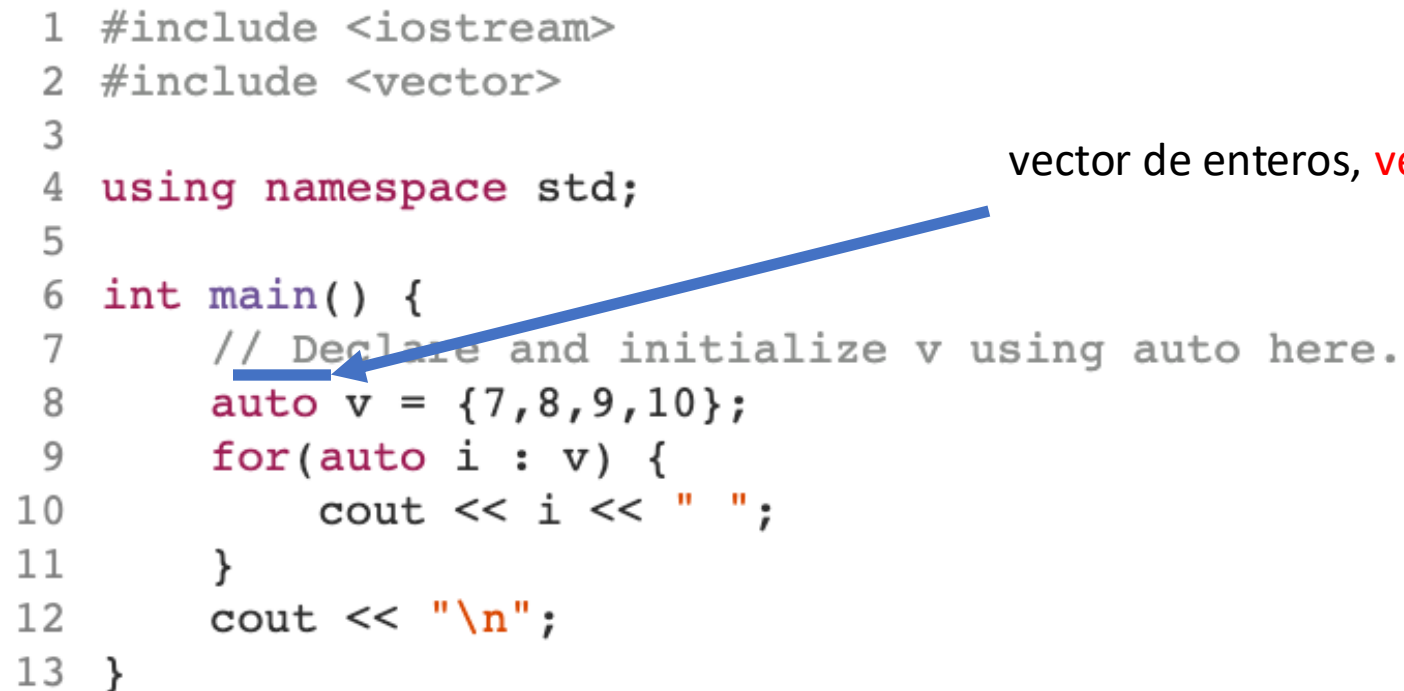
- Sin embargo, es posible que C++ haga una inferencia de tipo automática, usando la palabra clave **auto**.

```
1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 int main() {
7     auto i = 5;
8     auto v_6 = {1, 2, 3};
9     cout << "Variables declared and initialized without explicitly stating type!" << "\n";
10 }
```

Le decimos al compilador que adivine que tipo de dato es esta variable (en este caso vector de enteros, `vector<int>`)

Keyword **auto**

- Incluso se puede utilizar **auto** en el **range-based for** ciclo



```
1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 int main() {
7     // Declare and initialize v using auto here.
8     auto v = {7,8,9,10};
9     for(auto i : v) {
10         cout << i << " ";
11     }
12     cout << "\n";
13 }
```

vector de enteros, **vector<int>**

Keyword **auto**

- Incluso se puede utilizar **auto** en el **range-based for** ciclo

```
1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 int main() {
7     // Declare and initialize v using auto here.
8     auto v = {7,8,9,10};
9     for(auto i : v) {
10         cout << i << " ";
11     }
12     cout << "\n";
13 }
```

vector de enteros, **vector<int>**

Que tipo de dato tendrá la variable **i**?

Keyword **auto**

- Es útil declarar manualmente el tipo de una variable si desea que el tipo de variable sea claro para el lector de su código, o si desea ser explícito sobre la precisión numérica que se utiliza.
- C++ tiene varios tipos de números con diferentes niveles de precisión, y es posible que esta precisión no quede clara a partir del valor que se asigna.
- No es bueno abusar del uso de auto, se debe utilizar lo menos posible o únicamente cuando la precisión no es necesaria.