



# More vectors

Universidad Católica Boliviana

MSc, José Jesús Cabrera Pantoja

# Outline

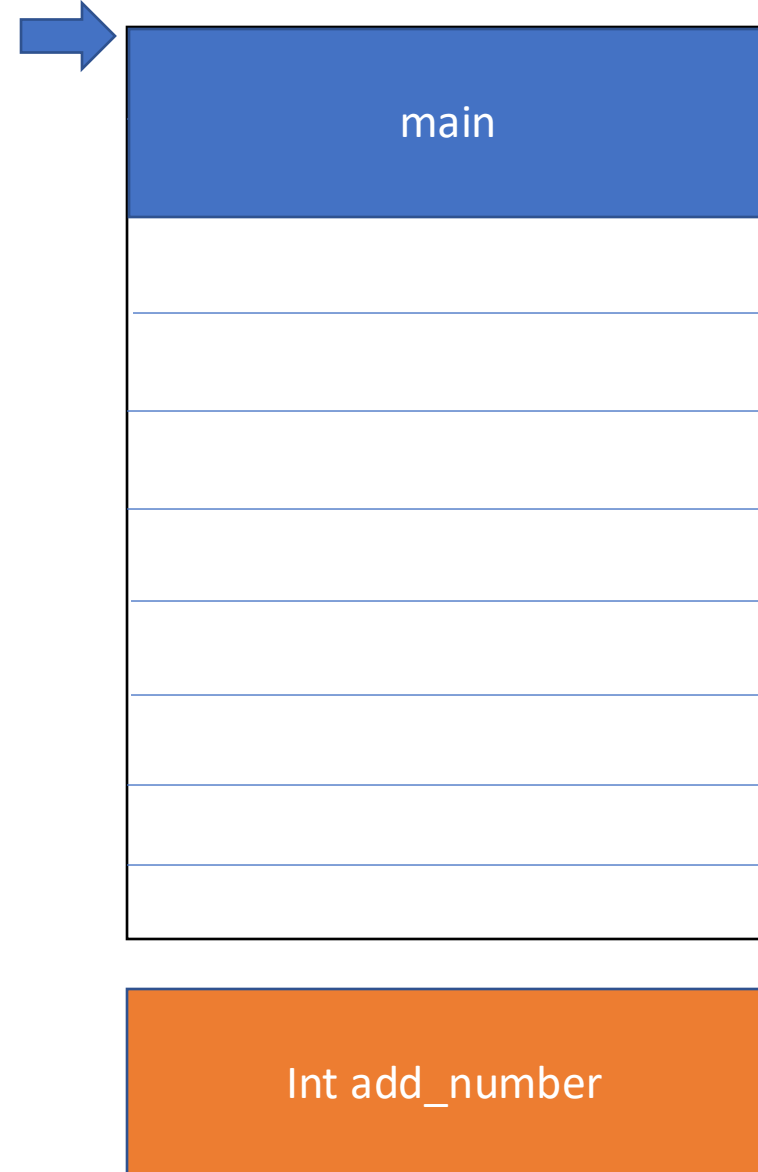
- Repaso Vectores
- Paso por valor vs paso por referencia
- Mas ejemplos de vectores
- Estructuras

# Memoria



```
1 int add_number(int a, int b);  
2  
3 int main() {  
4     cout << "From main!" << endl;  
5  
6     // llamada a la funcion  
7     // se le pasa dos valores int  
8     cout << add_number(4, 4);  
9  
10    // tambien se le puede pasar variables  
11    // de tipo int  
12    int x = 10;  
13    int x = 11;  
14    cout << add_number(x, y);  
15 }  
16  
17 int add_number(int a, int b) {  
18     int var = 0;  
19     // some code  
20  
21     // end of code  
22     return var;  
23 }
```

Memoria RAM



# Memoria



```
1 int add_number(int a, int b);
2
3 int main() {
4     cout << "From main!" << endl;
5
6     // llamada a la funcion
7     // se le pasa dos valores int
8     cout << add_number(4, 4);
9
10    // tambien se le puede pasar variables
11    // de tipo int
12    int x = 10;
13    int x = 11;
14    cout << add_number(x, y);
15 }
16
17 int add_number(int a, int b) {
18     int var = 0;
19     // some code
20
21     // end of code
22     return var;
23 }
```

Memoria RAM

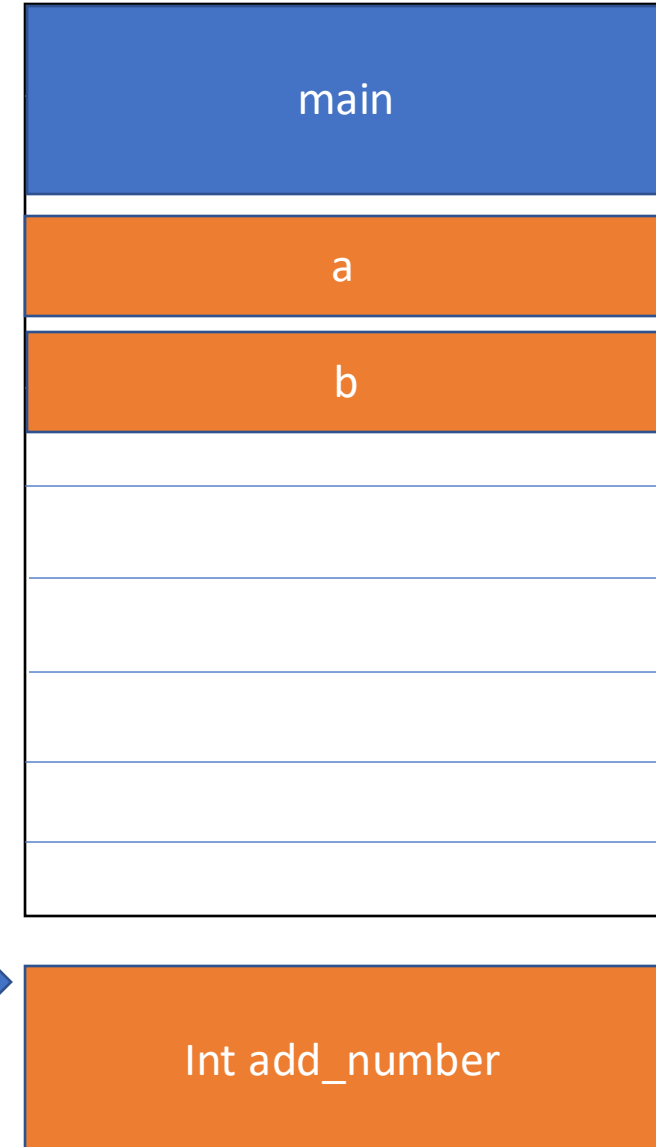


# Memoria



```
1 int add_number(int a, int b);
2
3 int main() {
4     cout << "From main!" << endl;
5
6     // llamada a la funcion
7     // se le pasa dos valores int
8     cout << add_number(4, 4);
9
10    // tambien se le puede pasar variables
11    // de tipo int
12    int x = 10;
13    int x = 11;
14    cout << add_number(x, y);
15 }
16
17 int add_number(int a, int b) {
18     int var = 0;
19     // some code
20
21     // end of code
22     return var;
23 }
```

Memoria RAM



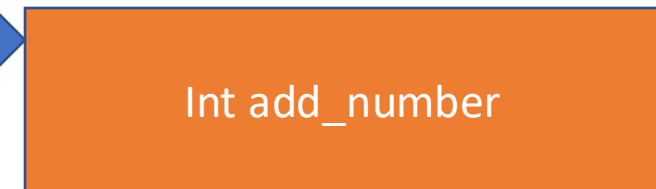
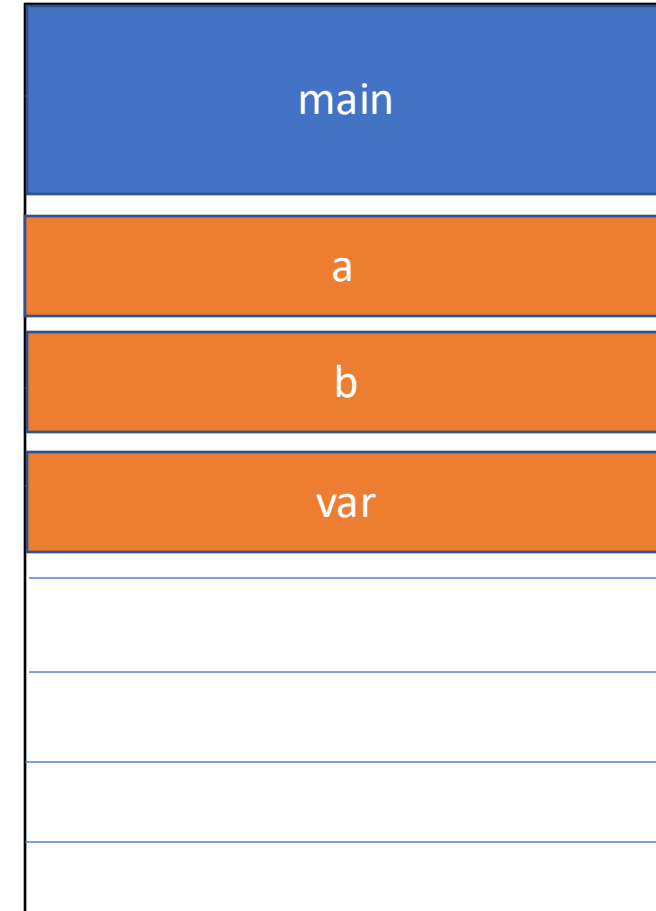
# Memoria



```
1 int add_number(int a, int b);
2
3 int main() {
4     cout << "From main!" << endl;
5
6     // llamada a la funcion
7     // se le pasa dos valores int
8     cout << add_number(4, 4);
9
10    // tambien se le puede pasar variables
11    // de tipo int
12    int x = 10;
13    int x = 11;
14    cout << add_number(x, y);
15 }
16
17 int add_number(int a, int b) {
18     int var = 0;
19     // some code
20
21     // end of code
22     return var;
23 }
```



Memoria RAM



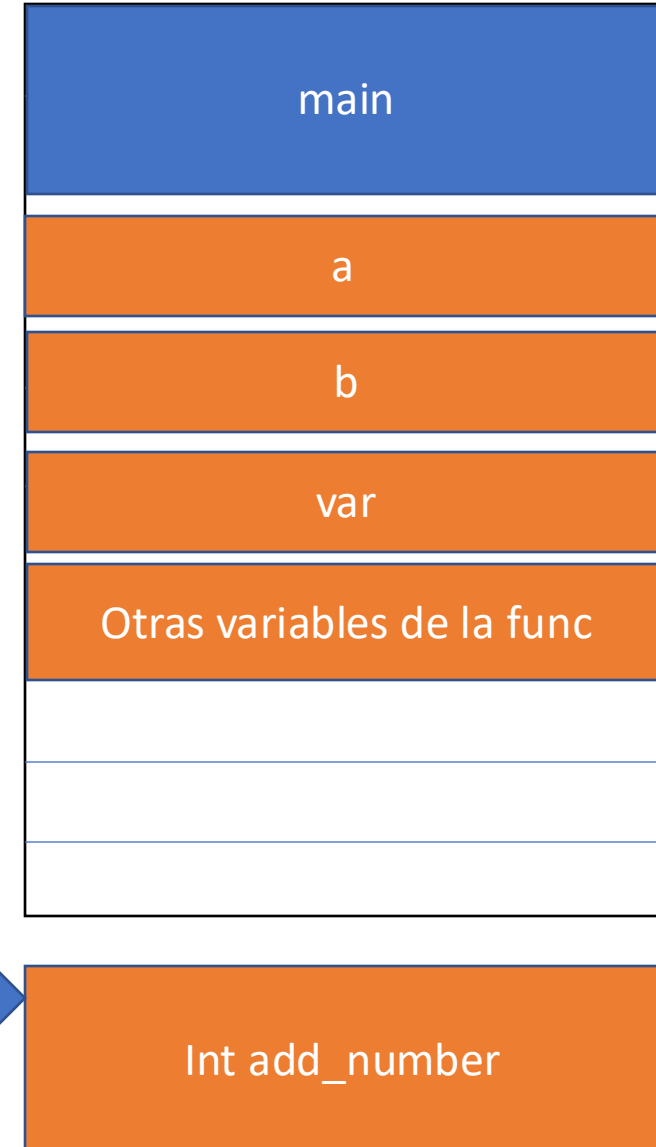
# Memoria



```
1 int add_number(int a, int b);
2
3 int main() {
4     cout << "From main!" << endl;
5
6     // llamada a la funcion
7     // se le pasa dos valores int
8     cout << add_number(4, 4);
9
10    // tambien se le puede pasar variables
11    // de tipo int
12    int x = 10;
13    int x = 11;
14    cout << add_number(x, y);
15 }
16
17 int add_number(int a, int b) {
18     int var = 0;
19     // some code
20
21     // end of code
22     return var;
23 }
```



Memoria RAM



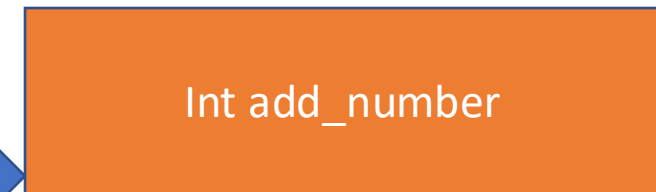
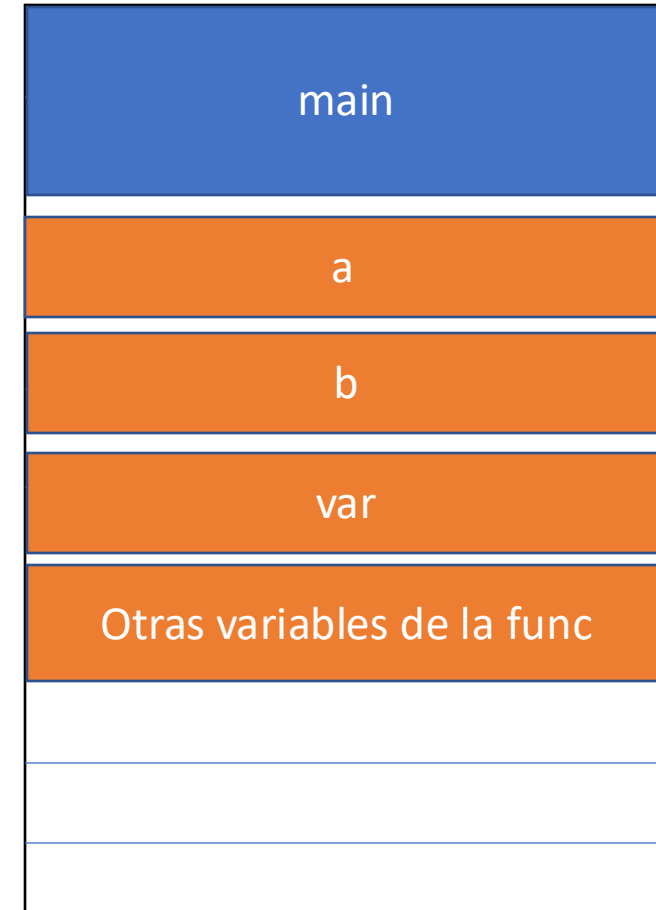
# Memoria



```
1 int add_number(int a, int b);
2
3 int main() {
4     cout << "From main!" << endl;
5
6     // llamada a la funcion
7     // se le pasa dos valores int
8     cout << add_number(4, 4);
9
10    // tambien se le puede pasar variables
11    // de tipo int
12    int x = 10;
13    int x = 11;
14    cout << add_number(x, y);
15 }
16
17 int add_number(int a, int b) {
18     int var = 0;
19     // some code
20
21     // end of code
22     return var;
23 }
```



Memoria RAM





# Memoria



```
1 int add_number(int a, int b);
2
3 int main() {
4     cout << "From main!" << endl;
5
6     // llamada a la funcion
7     // se le pasa dos valores int
8     cout << add_number(4, 4);
9
10    // tambien se le puede pasar variables
11    // de tipo int
12    int x = 10;
13    int x = 11;
14    cout << add_number(x, y);
15 }
16
17 int add_number(int a, int b) {
18     int var = 0;
19     // some code
20
21     // end of code
22     return var;
23 }
```



Memoria RAM



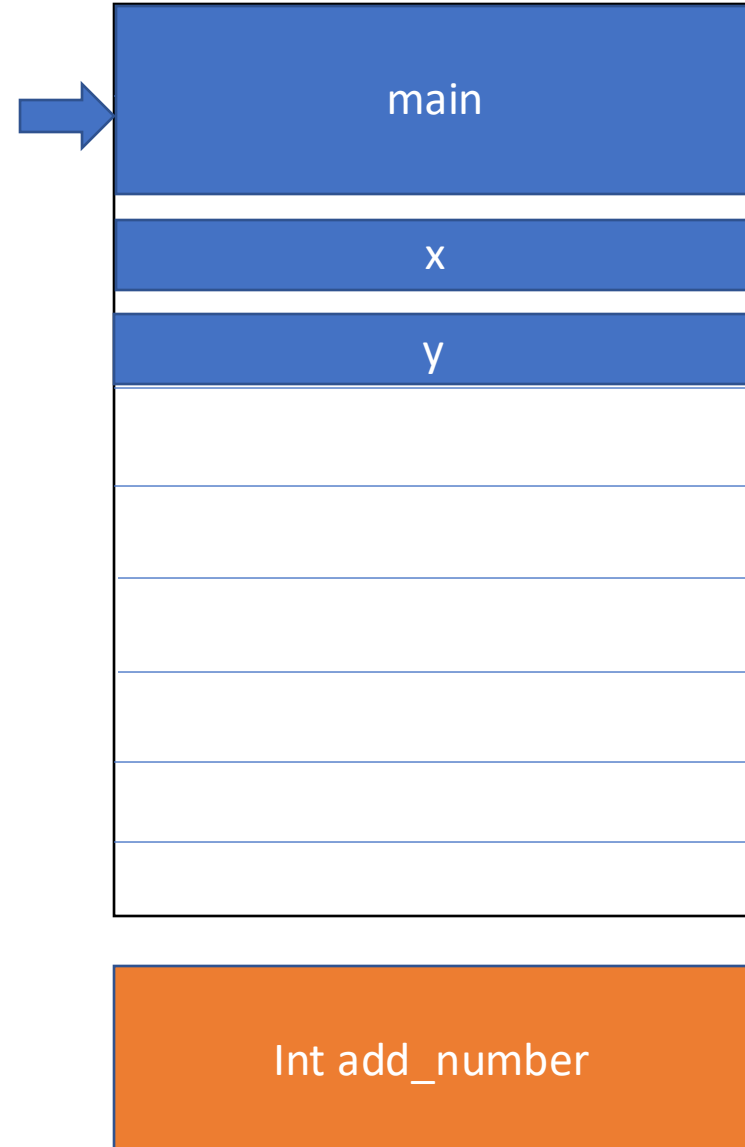
# Memoria



```
1 int add_number(int a, int b);
2
3 int main() {
4     cout << "From main!" << endl;
5
6     // llamada a la funcion
7     // se le pasa dos valores int
8     cout << add_number(4, 4);
9
10    // tambien se le puede pasar variables
11    // de tipo int
12    int x = 10;
13    int x = 11;
14    cout << add_number(x, y);
15 }
16
17 int add_number(int a, int b) {
18     int var = 0;
19     // some code
20
21     // end of code
22     return var;
23 }
```



Memoria RAM



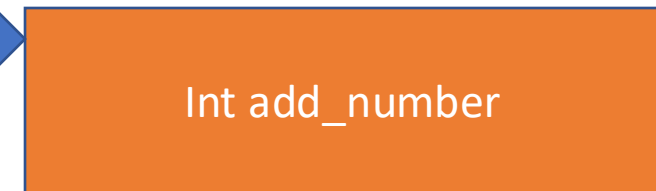
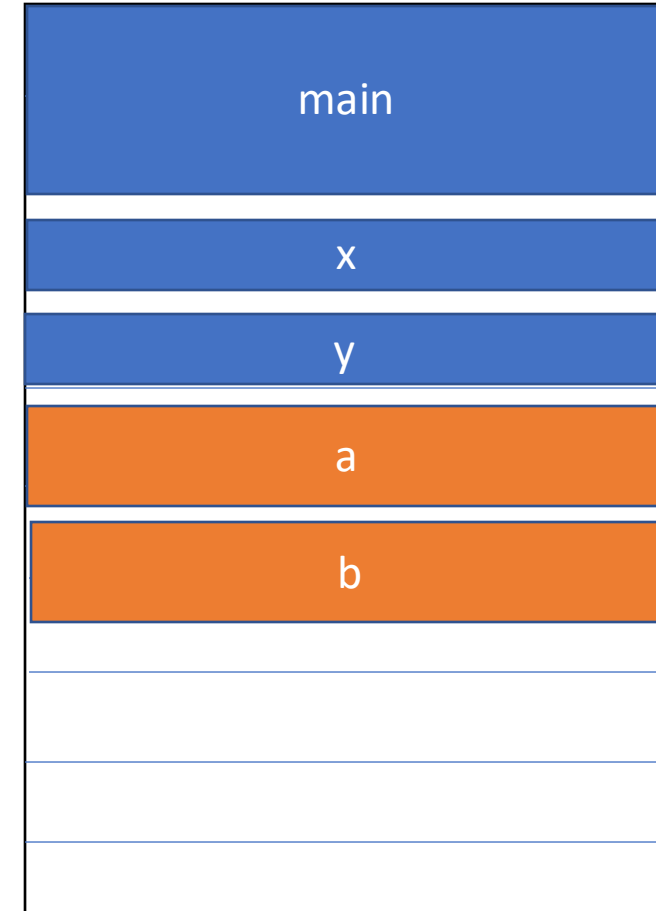
# Memoria



```
1 int add_number(int a, int b);
2
3 int main() {
4     cout << "From main!" << endl;
5
6     // llamada a la funcion
7     // se le pasa dos valores int
8     cout << add_number(4, 4);
9
10    // tambien se le puede pasar variables
11    // de tipo int
12    int x = 10;
13    int x = 11;
14    cout << add_number(x, y);
15 }
16
17 int add_number(int a, int b) {
18     int var = 0;
19     // some code
20
21     // end of code
22     return var;
23 }
```



Memoria RAM



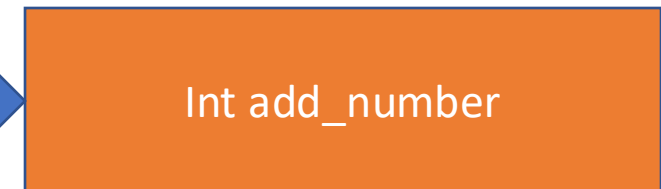
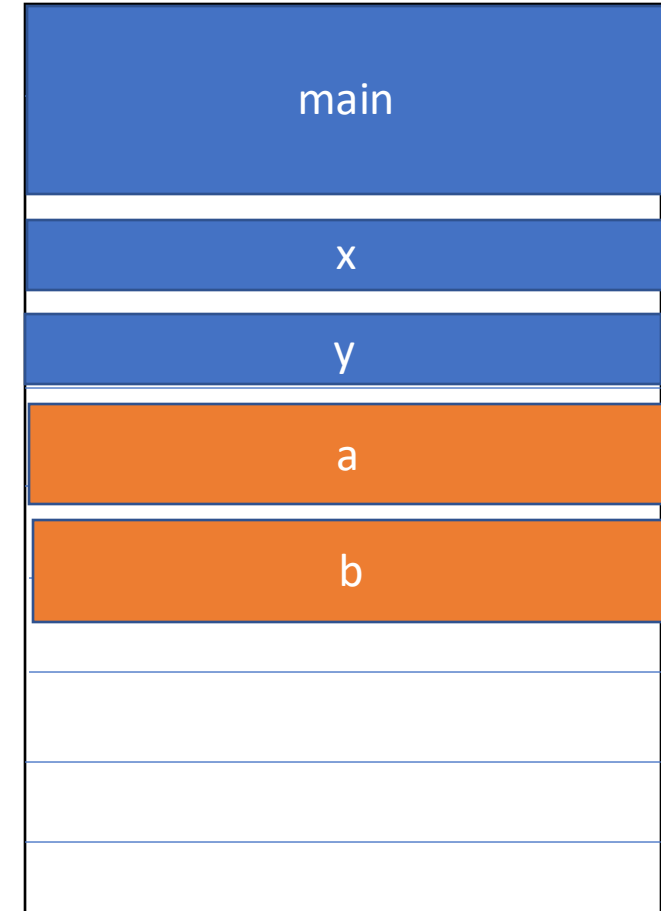
# Memoria



```
1 int add_number(int a, int b);
2
3 int main() {
4     cout << "From main!" << endl;
5
6     // llamada a la funcion
7     // se le pasa dos valores int
8     cout << add_number(4, 4);
9
10    // tambien se le puede pasar variables
11    // de tipo int
12    int x = 10;
13    int x = 11;
14    cout << add_number(x, y);
15 }
16
17 int add_number(int a, int b) {
18     int var = 0;
19     // some code
20
21     // end of code
22     return var;
23 }
```



Memoria RAM



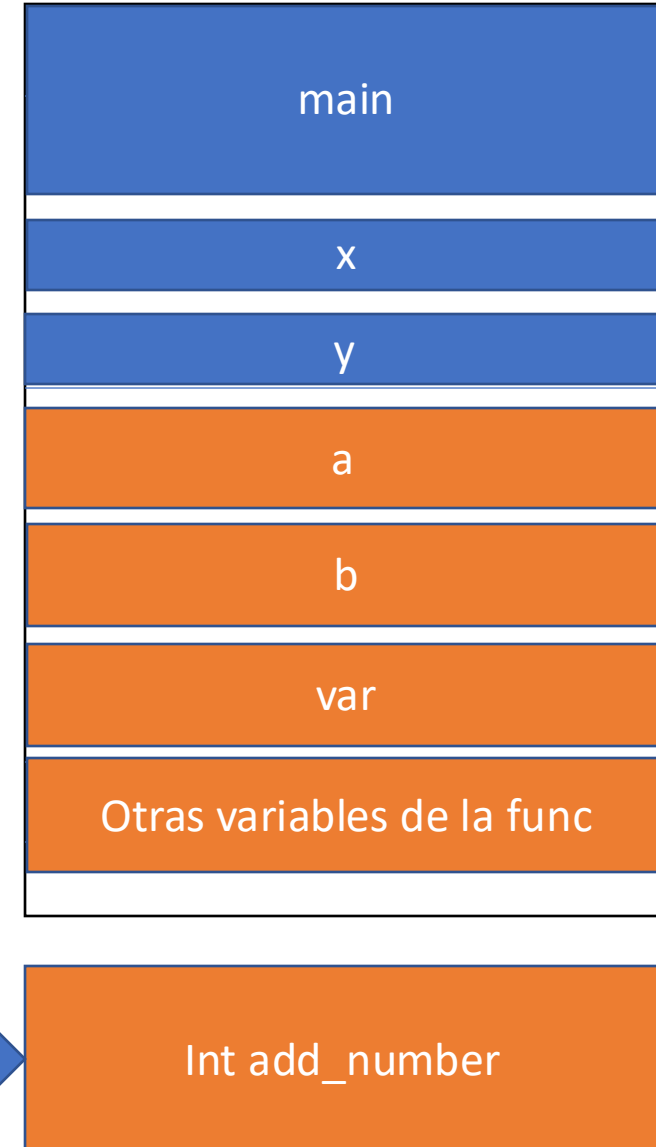
# Memoria



```
1 int add_number(int a, int b);
2
3 int main() {
4     cout << "From main!" << endl;
5
6     // llamada a la funcion
7     // se le pasa dos valores int
8     cout << add_number(4, 4);
9
10    // tambien se le puede pasar variables
11    // de tipo int
12    int x = 10;
13    int x = 11;
14    cout << add_number(x, y);
15 }
16
17 int add_number(int a, int b) {
18     int var = 0;
19     // some code
20
21     // end of code
22     return var;
23 }
```



Memoria RAM



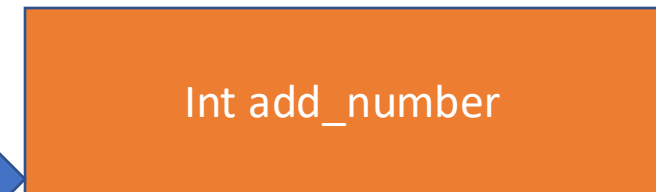
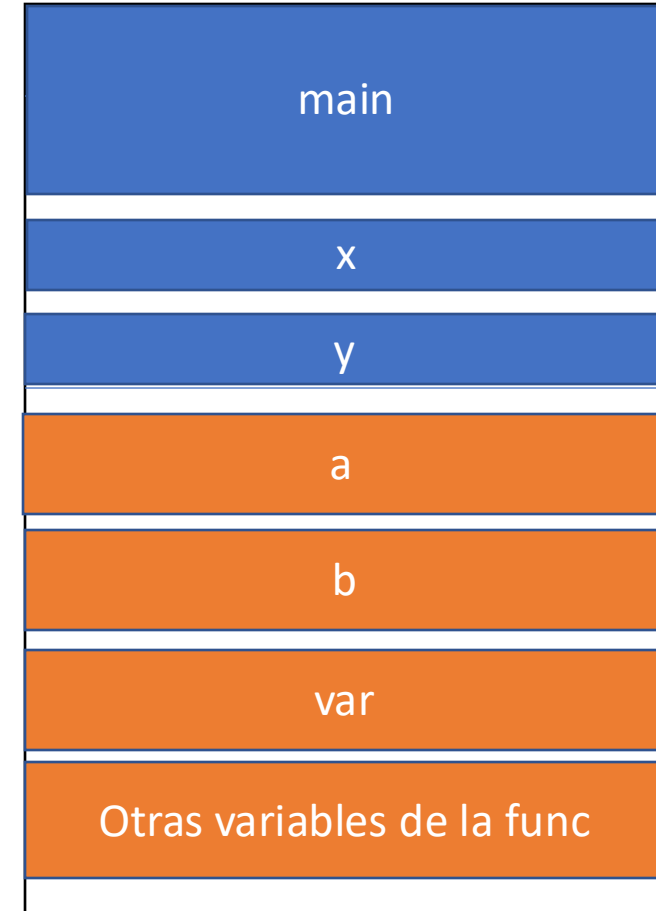
# Memoria



```
1 int add_number(int a, int b);
2
3 int main() {
4     cout << "From main!" << endl;
5
6     // llamada a la funcion
7     // se le pasa dos valores int
8     cout << add_number(4, 4);
9
10    // tambien se le puede pasar variables
11    // de tipo int
12    int x = 10;
13    int x = 11;
14    cout << add_number(x, y);
15 }
16
17 int add_number(int a, int b) {
18     int var = 0;
19     // some code
20
21     // end of code
22     return var;
23 }
```



Memoria RAM

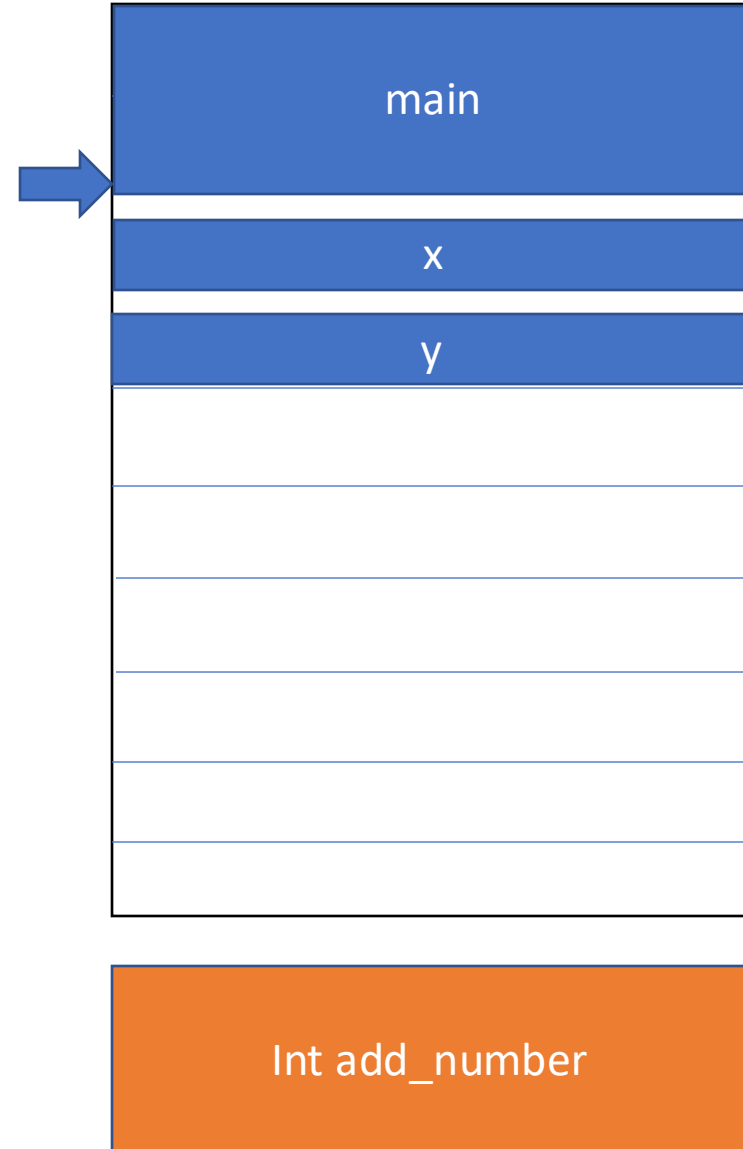


# Memoria



```
1 int add_number(int a, int b);
2
3 int main() {
4     cout << "From main!" << endl;
5
6     // llamada a la funcion
7     // se le pasa dos valores int
8     cout << add_number(4, 4);
9
10    // tambien se le puede pasar variables
11    // de tipo int
12    int x = 10;
13    int x = 11;
14    cout << add_number(x, y);
15 }
16
17 int add_number(int a, int b) {
18     int var = 0;
19     // some code
20
21     // end of code
22     return var;
23 }
```

Memoria RAM



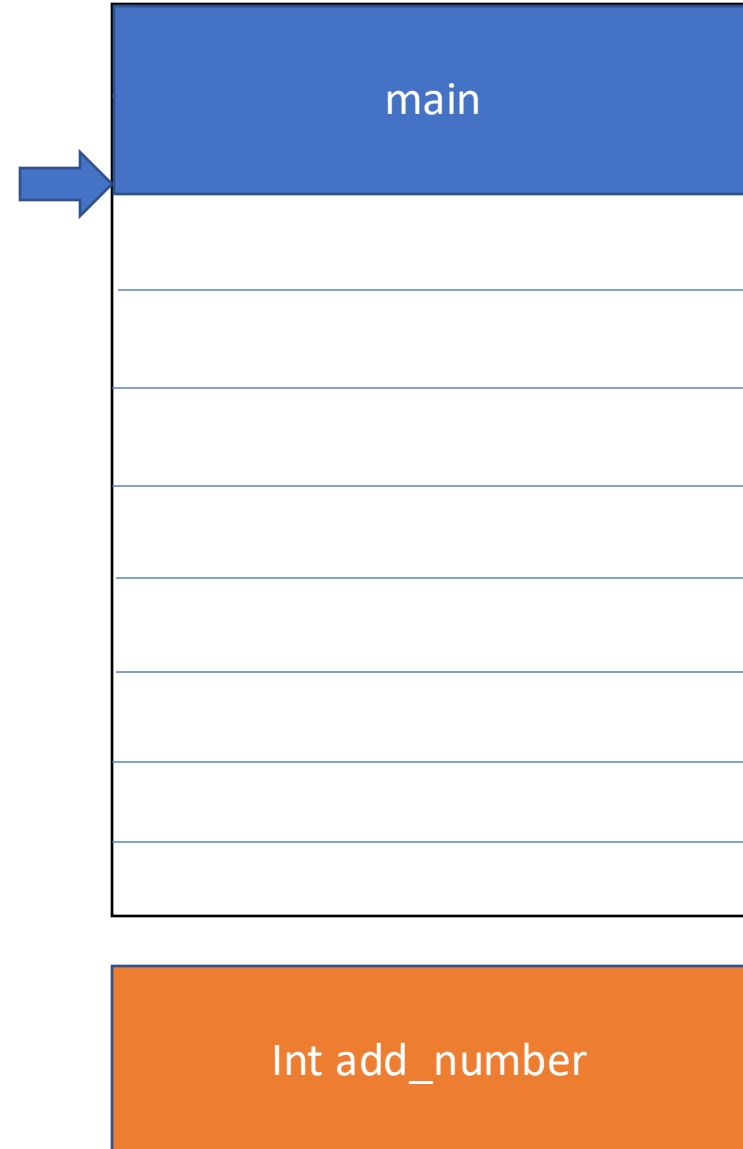
# Memoria



```
1 int add_number(int a, int b);
2
3 int main() {
4     cout << "From main!" << endl;
5
6     // llamada a la funcion
7     // se le pasa dos valores int
8     cout << add_number(4, 4);
9
10    // tambien se le puede pasar variables
11    // de tipo int
12    int x = 10;
13    int x = 11;
14    cout << add_number(x, y);
15 }
16
17 int add_number(int a, int b) {
18     int var = 0;
19     // some code
20
21     // end of code
22     return var;
23 }
```



Memoria RAM





# Scope (alcance)

La variable sum  
únicamente existe en  
la función  
(add\_numbers)  
Afuera de esta función  
ya no existe

```
1 #include <iostream>
2
3 using namespace std;
4
5 double add_numbers(int a, int b);
6
7 int main() {
8     cout << "Hello world!" << endl;
9
10    double result = add_numbers(1, 2);
11
12    cout << "Result = " << result << endl;
13
14    // No podemos acceder a variables internas de
15    // la funcion. Por ejemplo, sum unicamente existe en
16    // la funcion add_numbers. La siguiente linea devolvera
17    // un error:
18    // cout << sum << endl;
19
20    // Incluso los argumentos: (la siguiente linea es un error)
21    // cout << a;
22    // cout << b;
23 }
24
25 double add_numbers(int a, int b) {
26     double sum = a + b;
27     return sum;
28 }
```

# Scope (alcance)



```
1 #include <iostream>
2
3 int globalVar = 10; // global variable
4
5 int main() {
6     int localVar = 5; // local variable
7     std::cout << "Local variable value: " << localVar << std::endl;
8     std::cout << "Global variable value: " << globalVar << std::endl;
9     return 0;
10 }
```

Variable global – es una variable que se puede acceder y modificar desde cualquier parte del programa

Paso por valor vs Paso por referencia

# Paso por valor



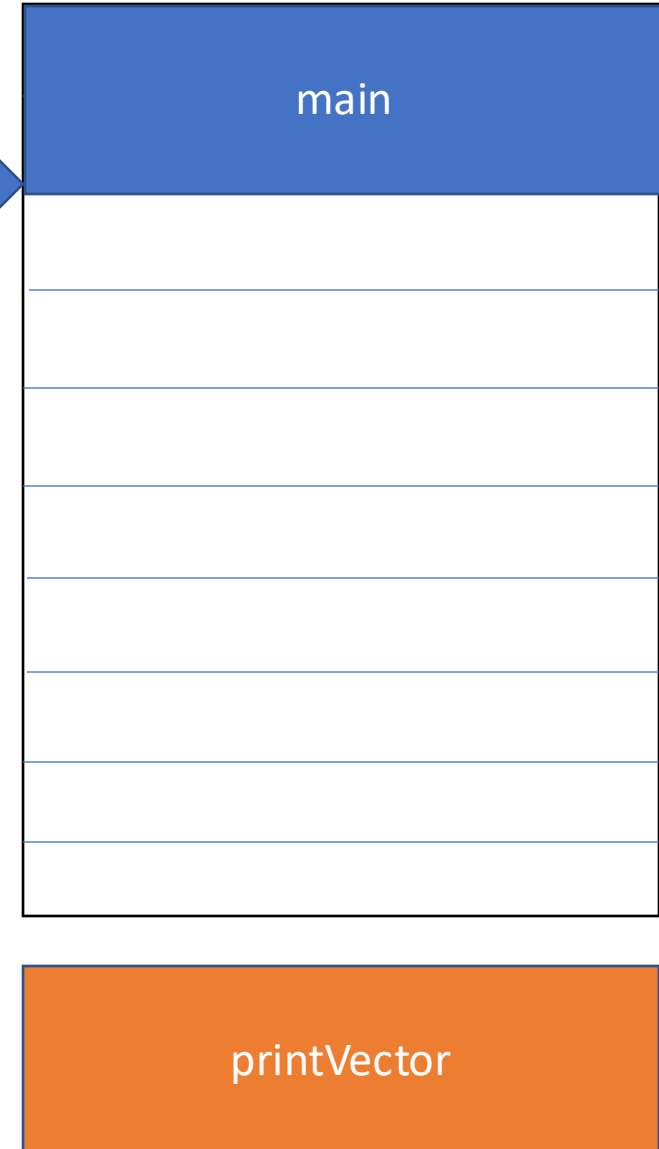
```
1 void printVector(vector<int> v);  
2  
3 int main() {  
4     vector<int> big_vector;  
5  
6     printVector(big_vector);  
7 }
```

# Paso por valor




```
1 void printVector(vector<int> v);  
2  
3 int main() {  
4     vector<int> big_vector;  
5  
6     printVector(big_vector);  
7 }
```

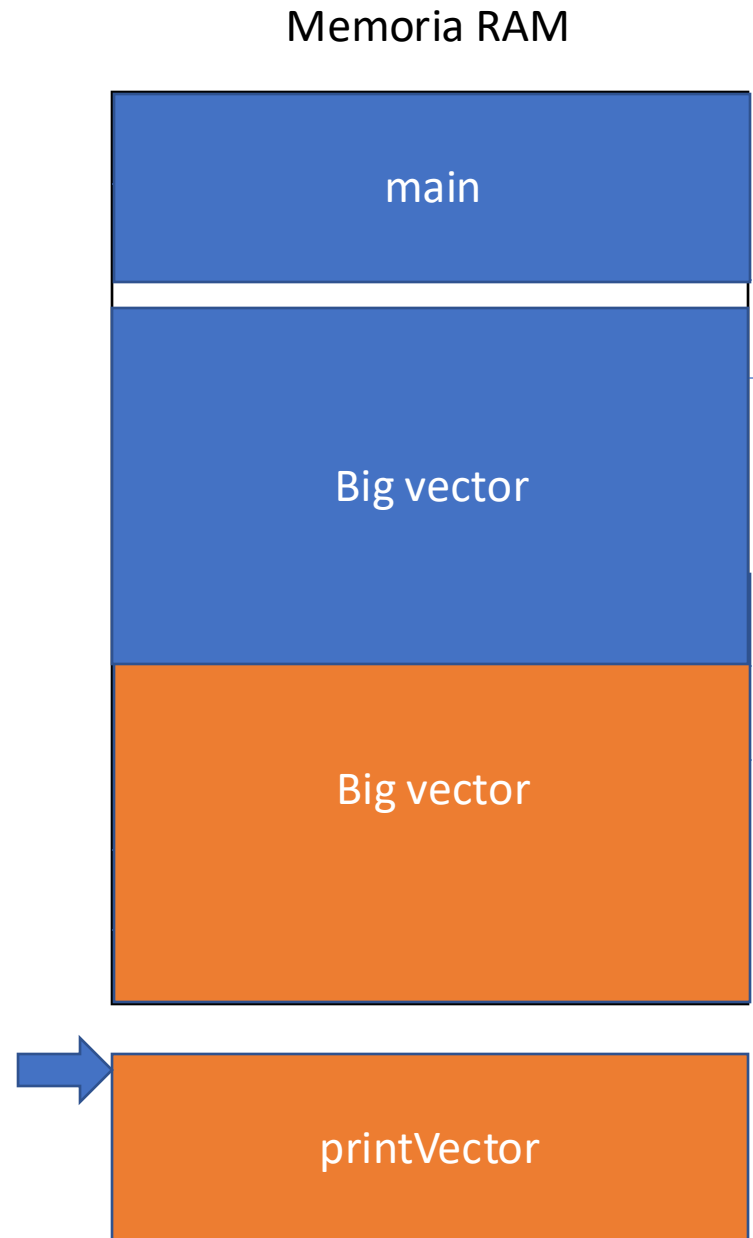

Memoria RAM



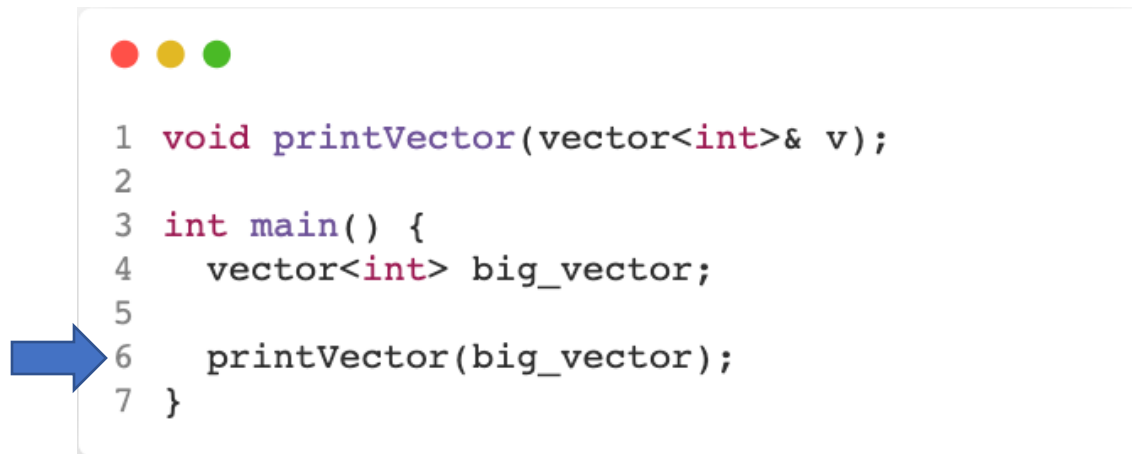
# Paso por valor



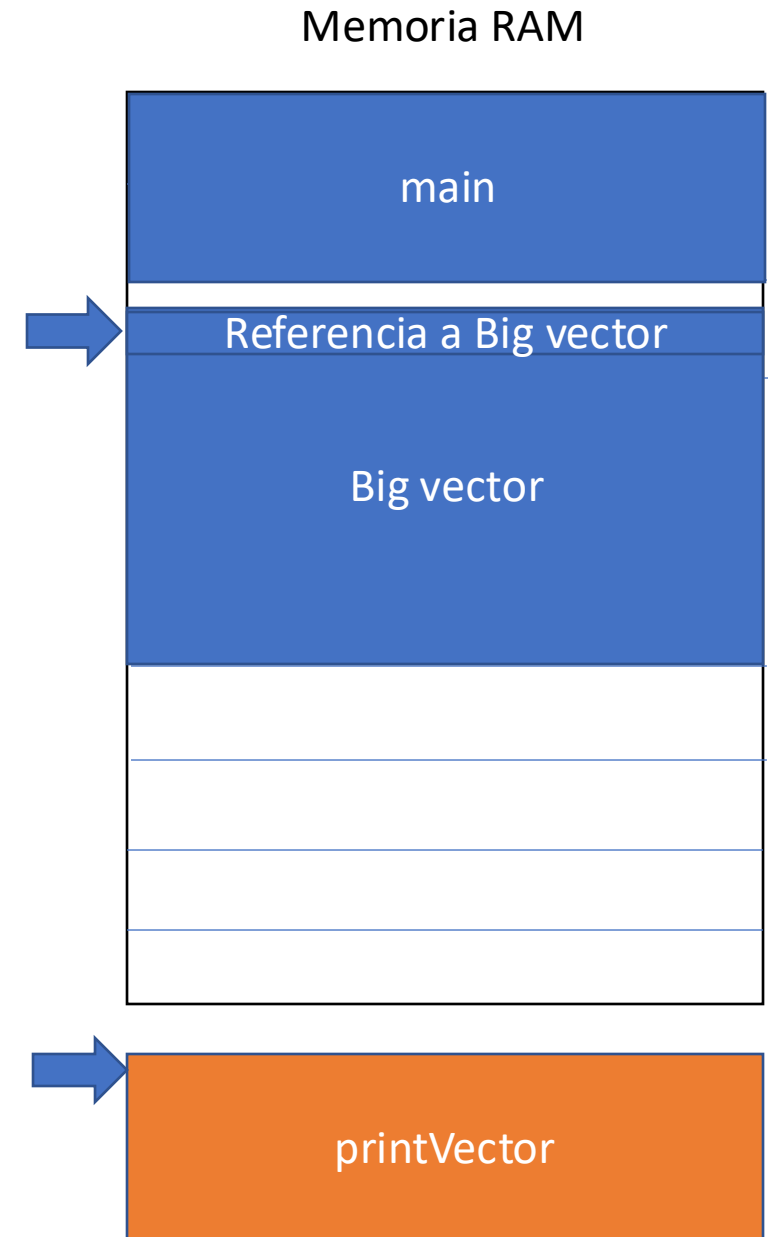
```
1 void printVector(vector<int> v);  
2  
3 int main() {  
4     vector<int> big_vector;  
5  
6     printVector(big_vector);  
7 }
```



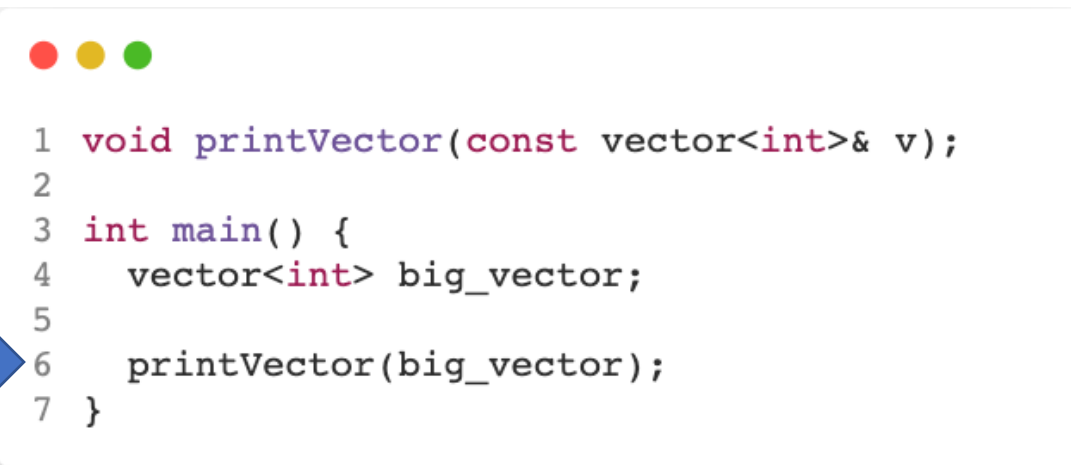
# Paso por referencia




```
1 void printVector(vector<int>& v);  
2  
3 int main() {  
4     vector<int> big_vector;  
5  
6     printVector(big_vector);  
7 }
```



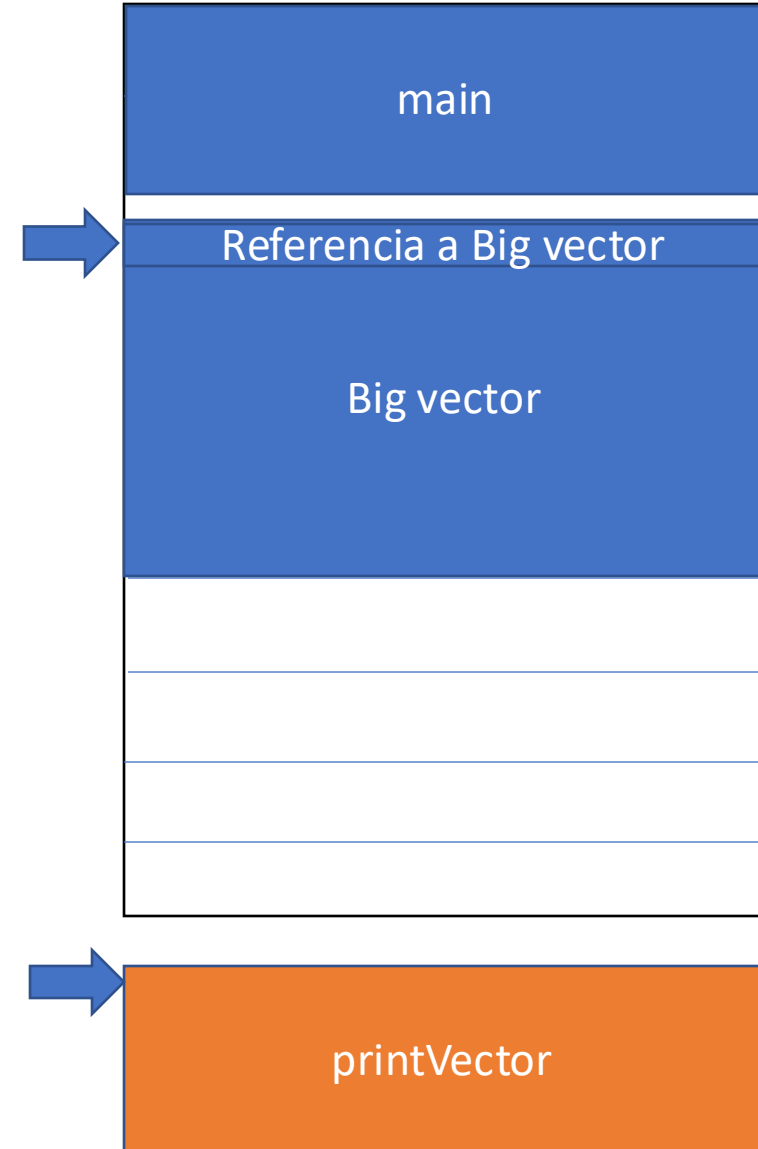
# Paso por referencia const



```
1 void printVector(const vector<int>& v);  
2  
3 int main() {  
4     vector<int> big_vector;  
5  
6     printVector(big_vector);  
7 }
```



Memoria RAM





Mas sobre vectores

# Mas sobre vectores

- Queremos crear un vector de strings.
- Los valores del vector son introducidos por el usuario.
- Se sabe el tamaño del vector al inicio.

# Mas sobre vectores

```
1 int main() {  
2     int n;  
3     cin >> n;  
4  
5     vector<string> v;  
6  
7     int i = 0;  
8     while (i < n) {  
9         string s;  
10        cin >> s;  
11        v.push_back(s);  
12        ++i;  
13    }  
14    printVector(v);  
15 }
```

- Queremos crear un vector.
- Los valores del vector son introducidos por el usuario.
- Se sabe el tamaño del vector al inicio.

# Mas sobre vectores

```
1 int main() {  
2     int n;  
3     cin >> n;  
4  
5     vector<string> v;  
6  
7     int i = 0;  
8     while (i < n) {  
9         string s;  
10        cin >> s;  
11        v.push_back(s);  
12        ++i;  
13    }  
14    printVector(v);  
15 }
```

- Queremos crear un vector.
- Los valores del vector son introducidos por el usuario.
- Se sabe el tamaño del vector al inicio.

En este vector se  
almacena los valores  
introducidos por el  
usuario

# Mas sobre vectores

```
1 int main() {  
2     int n;  
3     cin >> n;  
4  
5     vector<string> v;  
6  
7     int i = 0;  
8     while (i < n) {  
9         string s;  
10        cin >> s;  
11        v.push_back(s);  
12        ++i;  
13    }  
14    printVector(v);  
15 }
```

- Queremos crear un vector.
- Los valores del vector son introducidos por el usuario.
- Se sabe el tamaño del vector al inicio.

Mientras i sea menor a n se sigue solicitando valores al usuario

# Mas sobre vectores

```
1  int main() {  
2      int n;  
3      cin >> n;  
4  
5      vector<string> v;  
6  
7      int i = 0;  
8      while (i < n) {  
9          string s;  
10         cin >> s;  
11         v.push_back(s);  
12         ++i;  
13     }  
14     printVector(v);  
15 }
```

- Queremos crear un vector.
- Los valores del vector son introducidos por el usuario.
- Se sabe el tamaño del vector al inicio.

Se guarda el valor  
introducido por el  
usuario en el vector

# Mas sobre vectores

- Queremos crear un vector de strings.
- Los valores del vector son introducidos por el usuario.
- Se sabe el tamaño del vector al inicio.
- De que otra forma lo haría?

# Mas sobre vectores



```
1  int main() {  
2      int n;  
3      cin >> n;  
4  
5      vector<string> v(n);  
6      for(string& s : v) {  
7          cin >> s;  
8      }  
9  
10     printVector(v);  
11 }
```

- Queremos crear un vector.
- Los valores del vector son introducidos por el usuario.
- Se sabe el tamaño del vector al inicio.



# Mas sobre vectores



```
1  int main() {  
2      int n;  
3      cin >> n;  
4  
5      vector<string> v(n);  
6      for(string& s : v) {  
7          cin >> s;  
8      }  
9  
10     printVector(v);  
11 }
```

- Queremos crear un vector.
- Los valores del vector son introducidos por el usuario.
- Se sabe el tamaño del vector al inicio.

Un vector de tipo  
string

# Mas sobre vectores



```
1  int main() {  
2      int n;  
3      cin >> n;  
4  
5      vector<string> v(n);  
6      for(string& s : v) {  
7          cin >> s;  
8      }  
9  
10     printVector(v);  
11 }
```

- Queremos crear un vector.
- Los valores del vector son introducidos por el usuario.
- Se sabe el tamaño del vector al inicio.

Creamos un vector de tamaño n (con elementos default)

# Mas sobre vectores



```
1  int main() {
2      int n;
3      cin >> n;
4
5      vector<string> v(n);
6      for(string& s : v) {
7          cin >> s;
8      }
9
10     printVector(v);
11 }
```

- Queremos crear un vector.
- Los valores del vector son introducidos por el usuario.
- Se sabe el tamaño del vector al inicio.

→ Ciclo range based for

# Mas sobre vectores



```
1  int main() {  
2      int n;  
3      cin >> n;  
4  
5      vector<string> v(n);  
6      for(string& s : v) {  
7          cin >> s;  
8      }  
9  
10     printVector(v);  
11 }
```

- Queremos crear un vector.
- Los valores del vector son introducidos por el usuario.
- Se sabe el tamaño del vector al inicio.

Tomamos una  
referencia al elemento  
del vector para  
modificarlo

# Mas sobre vectores



```
1  int main() {  
2      int n;  
3      cin >> n;  
4  
5      vector<string> v(n);  
6      for(string& s : v) {  
7          cin >> s;  
8      }  
9  
10     printVector(v);  
11 }
```

- Queremos crear un vector.
- Los valores del vector son introducidos por el usuario.
- Se sabe el tamaño del vector al inicio.

Solicitamos al usuario introducir un valor y se guarda en la referencia al elemento del vector

# Mas sobre vectores

- Tenemos un vector con los días que tiene cada mes
- Queremos modificar los días de febrero dependiendo del año

# Mas sobre vectores

- Tenemos un vector con los días que tiene cada mes
- Queremos modificar los días de febrero dependiendo del año
- Como lo haría?



```
1 int main() {  
2     vector<int> day_in_months = {31, 28, 31, 30, 31};  
3     printVector(v);  
4 }
```

# Mas sobre vectores

- Tenemos un vector con los días que tiene cada mes
- Queremos modificar los días de febrero dependiendo del año

Es año bisiesto?

```
1 int main() {  
2     vector<int> day_in_months = {31, 28, 31, 30, 31};  
3     if(true) { // if year is leap  
4         days_in_months[1]++;  
5     }  
6     printVector(v);  
7 }
```



# Mas sobre vectores

- Tenemos un vector con los días que tiene cada mes
- Queremos modificar los días de febrero dependiendo del año

Si es entonces  
aumentamos un día a  
febrero

```
1 int main() {  
2     vector<int> day_in_months = {31, 28, 31, 30, 31};  
3     if(true) { // if year is leap  
4         days_in_months[1]++;  
5     }  
6     printVector(v);  
7 }
```

# Mas sobre vectores

- Queremos crear un vector de booleans.
- Este vector tendrá valores de True en los días del mes que son feriados.

# Mas sobre vectores

- Queremos crear un vector de booleans.
- Este vector tendrá valores de True en los días del mes que son feriados.



```
1 int main() {  
2     vector<bool> is_holiday(28, false);  
3     is_holiday[21] = true;  
4     printVector(v);  
5 }
```

↓  
Creamos un vector de  
booleans

# Mas sobre vectores

- Queremos crear un vector de booleans.
- Este vector tendrá valores de True en los días del mes que son feriados.



```
1 int main() {  
2     vector<bool> is_holiday(28, false);  
3     is_holiday[21] = true;  
4     printVector(v);  
5 }
```

El tamaño del vector

# Mas sobre vectores

- Queremos crear un vector de booleans.
- Este vector tendrá valores de True en los días del mes que son feriados.



```
1 int main() {  
2     vector<bool> is_holiday(28, false);  
3     is_holiday[21] = true;  
4     printVector(v);  
5 }
```

Este vector con  
tamaño 28 se rellenara  
de false. (de ceros)

# Mas sobre vectores

- Queremos crear un vector de booleans.
- Este vector tendrá valores de True en los días del mes que son feriados.




```
1 int main() {  
2     vector<bool> is_holiday(28, false);  
3     is_holiday[21] = true;  
4     printVector(v);  
5 }
```

El 20 de febrero fue  
feriado. Por tanto, lo  
marcamos como  
verdadero.

# Mas sobre vectores

- Queremos crear un vector de booleans.
- Este vector tendrá valores de True en los días del mes que son feriados.



```
1 int main() {  
2     vector<bool> is_holiday(28, false);  
3     is_holiday[21] = true;  
4     printVector(v);  
5  
6     is_holiday.resize(31);  
7     printVector(v);  
8 }
```

# Mas sobre vectores

- Queremos crear un vector de booleans.
- Este vector tendrá valores de True en los días del mes que son feriados.



```
1 int main() {  
2     vector<bool> is_holiday(28, false);  
3     is_holiday[21] = true;  
4     printVector(v);  
5  
6     is_holiday.resize(31);  
7     printVector(v);  
8 }
```

Para irnos a otro mes  
debemos cambiar el  
tamaño del vector. Para  
ello se utiliza resize.





# 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
```

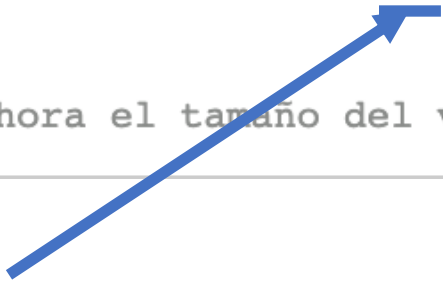
- Argumento — por ahora, tómelo como lo que esta entre paréntesis

# 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

# Mas sobre vectores

- Queremos crear un vector de booleans.
- Este vector tendrá valores de True en los días del mes que son feriados.



```
1 int main() {  
2     vector<bool> is_holiday(28, false);  
3     is_holiday[21] = true;  
4     printVector(v);  
5  
6     is_holiday.resize(31);  
7     printVector(v);  
8 }
```

Por tanto, esto rellenara  
los espacios inexistentes  
con ceros!



# Mas sobre vectores

- Cual es el error?



```
1 int main() {  
2     vector<bool> is_holiday(28, false);  
3     is_holiday[21] = true;  
4     printVector(v);  
5  
6     is_holiday.resize(31);  
7     printVector(v);  
8 }
```

Por tanto, esto rellenara  
los espacios inexistentes  
con ceros!



# Mas sobre vectores



```
1 int main() {  
2     vector<bool> is_holiday(28, false);  
3     is_holiday[21] = true;  
4     printVector(v);  
5  
6     is_holiday.assign(31, false);  
7     is_holiday[0] = true;  
8     printVector(v);  
9 }
```

# Mas sobre vectores



```
1 int main() {  
2     vector<bool> is_holiday(28, false);  
3     is_holiday[21] = true;  
4     printVector(v);  
5  
6     is_holiday.assign(31, false);  
7     is_holiday[0] = true;  
8     printVector(v);  
9 }
```

Cuando queremos cambiar el tamaño del vector y limpiarlo para reutilizarlo es mejor utilizar assign





# Mas sobre vectores



```
1 int main() {  
2     vector<bool> is_holiday(28, false);  
3     is_holiday[21] = true;  
4     printVector(v);  
5  
6     is_holiday.assign(31, false);  
7     is_holiday[0] = true;  
8     printVector(v);  
9  
10    is_holiday.clear();  
11 }
```

# Mas sobre vectores



```
1 int main() {  
2     vector<bool> is_holiday(28, false);  
3     is_holiday[21] = true;  
4     printVector(v);  
5  
6     is_holiday.assign(31, false);  
7     is_holiday[0] = true;  
8     printVector(v);  
9  
10    is_holiday.clear();  
11 }
```

Pero si solo  
queremos  
limpiarlo  
usamos  
clear



Estructuras

# Structure!

- Podemos crear nuestros tipos de variables (conjuntos de otras variables)
- Para ello requerimos la palabra clave `struct`

# Structure!



```
1 // Creamos nuestra estrucutra
2 // esta estrucutra tiene varios
3 // "campos" (name, surname, age)
4 struct Person {
5     string name;
6     string surname;
7     int age;
8 };
9
10 int main() {
11     // Creamos una variable de tipo Person
12     Person person1;
13
14     // Llenamos los campos de esta nueva variable
15     person1.name = "Jose Jesus";
16     person1.surname = "Cabrera";
17     person1.age = 28;
18
19     // Mostramos la informacion de la estructura
20     cout << "Person's name is " << person1.name << endl;
21     cout << "Person's surname is " << person1.surname << endl;
22     cout << "Age: " << person1.age << endl;
23 }
```

# Structure!



```
1 // Creamos nuestra estrucutra
2 // esta estrucutra tiene varios
3 // "campos" (name, surname, age)
4 struct Person {
5     string name;
6     string surname;
7     int age;
8 };
9
10 int main() {
11     // Creamos una variable de tipo Person
12     Person person1;
13
14     // Llenamos los campos de esta nueva variable
15     person1.name = "Jose Jesus";
16     person1.surname = "Cabrera";
17     person1.age = 28;
18
19     // Mostramos la informacion de la estructura
20     cout << "Person's name is " << person1.name << endl;
21     cout << "Person's surname is " << person1.surname << endl;
22     cout << "Age: " << person1.age << endl;
23 }
```

Creamos una estructura. El nombre de la estructura puede ser cualquiera

# Structure!



```
1 // Creamos nuestra estrucutra
2 // esta estrucutra tiene varios
3 // "campos" (name, surname, age)
4 struct Person {
5     string name;
6     string surname;
7     int age;
8 };
9
10 int main() {
11     // Creamos una variable de tipo Person
12     Person person1;
13
14     // Llenamos los campos de esta nueva variable
15     person1.name = "Jose Jesus";
16     person1.surname = "Cabrera";
17     person1.age = 28;
18
19     // Mostramos la informacion de la estructura
20     cout << "Person's name is " << person1.name << endl;
21     cout << "Person's surname is " << person1.surname << endl;
22     cout << "Age: " << person1.age << endl;
23 }
```

Campos de la  
estructura



# Structure!



```
1 // Creamos nuestra estrucutra
2 // esta estrucutra tiene varios
3 // "campos" (name, surname, age)
4 struct Person {
5     string name;
6     string surname;
7     int age;
8 };
9
10 int main() {
11     // Creamos una variable de tipo Person
12     Person person1;
13
14     // Llenamos los campos de esta nueva variable
15     person1.name = "Jose Jesus";
16     person1.surname = "Cabrera";
17     person1.age = 28;
18
19     // Mostramos la informacion de la estructura
20     cout << "Person's name is " << person1.name << endl;
21     cout << "Person's surname is " << person1.surname << endl;
22     cout << "Age: " << person1.age << endl;
23 }
```

↓  
Creamos una variable  
de tipo Person



# Structure!



```
1 // Creamos nuestra estrucutra
2 // esta estrucutra tiene varios
3 // "campos" (name, surname, age)
4 struct Person {
5     string name;
6     string surname;
7     int age;
8 };
9
10 int main() {
11     // Creamos una variable de tipo Person
12     Person person1;
13
14     // Llenamos los campos de esta nueva variable
15     person1.name = "Jose Jesus";
16     person1.surname = "Cabrera";
17     person1.age = 28;
18
19     // Mostramos la informacion de la estructura
20     cout << "Person's name is " << person1.name << endl;
21     cout << "Person's surname is " << person1.surname << endl;
22     cout << "Age: " << person1.age << endl;
23 }
```

Llenamos los campos  
de la estructura

# Structure!



```
1 // Creamos nuestra estrucutra
2 // esta estrucutra tiene varios
3 // "campos" (name, surname, age)
4 struct Person {
5     string name;
6     string surname;
7     int age;
8 };
9
10 int main() {
11     // Creamos una variable de tipo Person
12     Person person1;
13
14     // Llenamos los campos de esta nueva variable
15     person1.name = "Jose Jesus";
16     person1.surname = "Cabrera";
17     person1.age = 28;
18
19     // Mostramos la informacion de la estructura
20     cout << "Person's name is " << person1.name << endl;
21     cout << "Person's surname is " << person1.surname << endl;
22     cout << "Age: " << person1.age << endl;
23 }
```

Mostramos los datos  
de la estructura

# Structure!

Creamos un vector  
que contendrá  
variables de tipo  
Person



```
1 // Creamos nuestra estrucutra
2 // esta estrucutra tiene varios
3 // "campos" (name, surname, age)
4 struct Person {
5     string name;
6     string surname;
7     int age;
8 };
9
10 int main() {
11     // Creamos una variable de tipo Person
12     vector<Person> v;
13 }
```

# Structure!

Podemos devolver  
nuestro tipo de  
variable Person



```
1 // Creamos nuestra estrucutra
2 // esta estrucutra tiene varios
3 // "campos" (name, surname, age)
4 struct Person {
5     string name;
6     string surname;
7     int age;
8 };
9
10
11 Person getInfo();
12
13 void showInfo(Person p);
14
15 int main() {
16     // Creamos una variable de tipo Person
17     vector<Person> v;
18 }
```

# Structure!

Se puede aceptar este  
tipo de variable  
Person



```
1 // Creamos nuestra estrucutra
2 // esta estrucutra tiene varios
3 // "campos" (name, surname, age)
4 struct Person {
5     string name;
6     string surname;
7     int age;
8 };
9
10
11 Person getInfo();
12
13 void showInfo(Person p);
14
15 int main() {
16     // Creamos una variable de tipo Person
17     vector<Person> v;
18 }
```