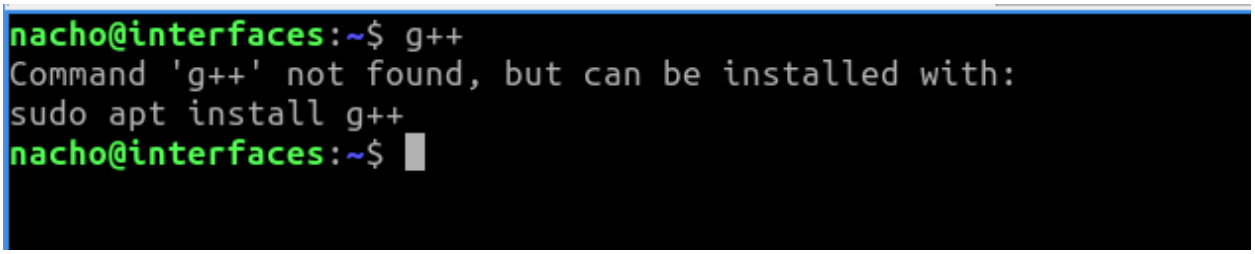
Guía rápida de adaptación a C y C++ para estudiantes que han aprendido a programar con Java

C es un lenguaje de programación algo antiguo. Llegado el momento, se amplió con muchas más capacidades, y se creó C++. Este lenguaje ha sido repetidamente ampliado conforme avanzan los tiempos y se introducen nuevas técnicas y paradigmas. En este curso NO distinguiremos apenas entre C y C++, en cualquier momento nos referiremos a ambos con cualquier nombre de "C" o "C++", aunque en general nuestro objetivo es aprender C++ (que incluye prácticamente C). Tampoco nos importará qué versión de C++ usamos a cada momento.

Instalar c++

En algunas distribuciones el compilador de c++ no viene instalado. Para verificar si está instalado prueba a ejecutar la orden "g++", si el mensaje de error indica que no existe el comando, es que no está instalado.



Si el compilador está instalado, el mensaje de error al ejecutar "g++" es otro y ya indica que le faltan archivos.

Para instalarlo sigue estos pasos

1. hacerse superusuario desde un usuario con permisos para ejecutar sudo

   ```
   su - lliurex
   (siendo usuario "lliurex") sudo su -
   ```

2. `apt-get update`
3. `apt-get upgrade`
4. `apt-get install g++`

    a. Alternativamente podemos instala un paquete bastante popular que incluye al compilador y otras herramientas de desarrollo "apt-get install build-essentials"

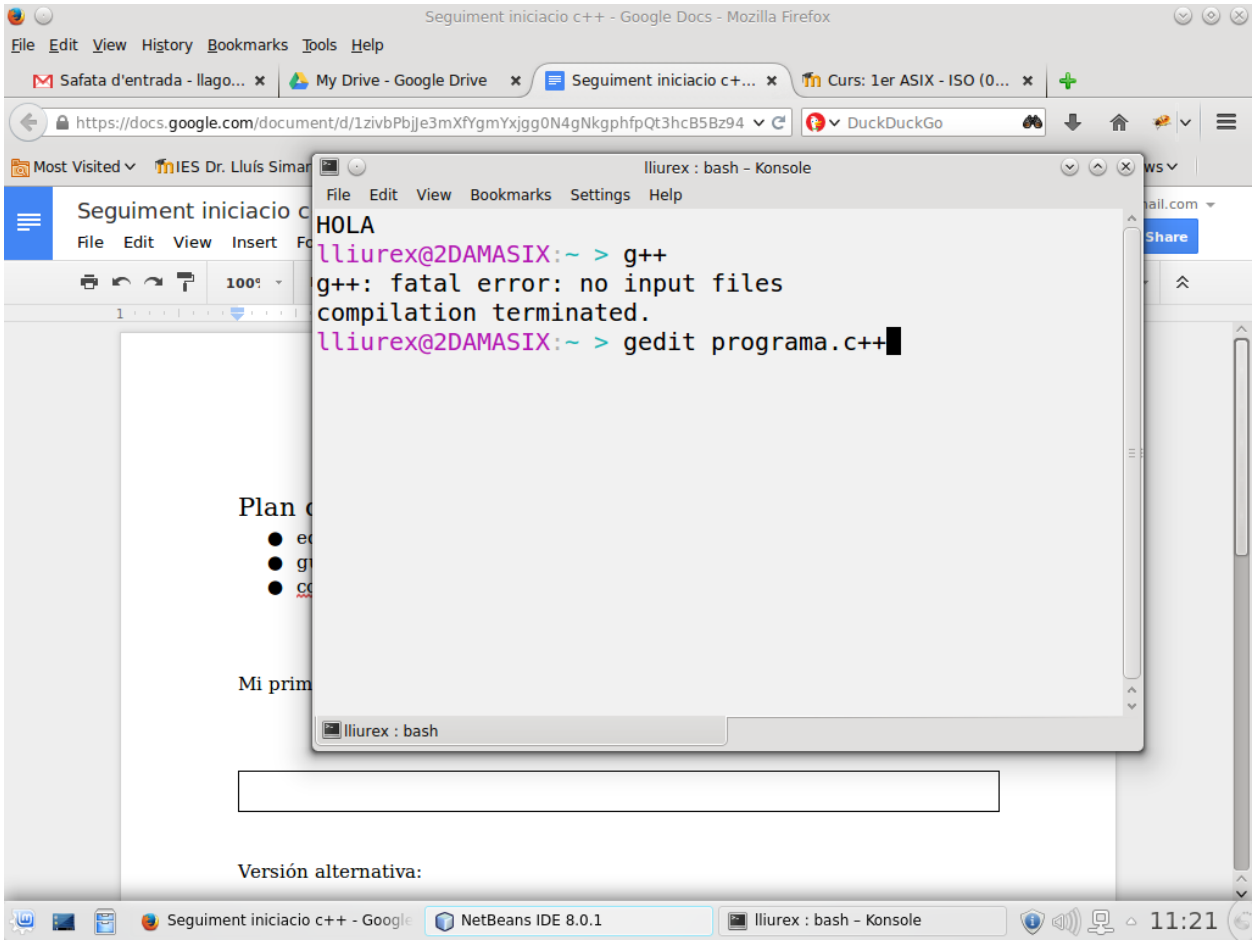
Plan de trabajo:

1. Usar un **terminal** desde el que realizar el trabajo
2. **Editar** un fichero. Necesitamos un editor de texto. Se recomienda uno que resalte con colores la sintaxis del lenguaje. "gedit" es un buen ejemplo de editor.
3. Guardar el fichero en un directorio de trabajo (vacío en principio), sin otros ficheros que molesten.
4. **Compilar** (es decir: generar un programa ejecutable a partir de nuestro fichero en C). Necesitamos el compilador de c++ (comando g++)
    - Observar y corregir posibles errores que impidan su compilación

    > Si has tenido clases de programación con "bash", esta es la parte nueva en C. Bash interpreta directamente el fichero que programas. En el caso de C, hay que transformar el fichero e programa… es lo que se conoce como : compilar

5. Ejecutar el programa compilado.
    - Observar el funcionamiento

Observa la captura de pantalla siguiente



La línea de comandos anterior "`gedit programa.c++`" abre el editor, pero deja "clavada" la consola. Es preferible abrir el editor en segundo plano y seguir usando la consola para compilar y ejecutar el programa. Esto se consigue escribiendo un "&" al final de la línea de comandos:

```
gedit programa.c++ &
```

Con la anterior línea de comandos lanzarás un editor que se quedará abierto sin bloquear el terminal desde el que lo has lanzado.

Mi primer programa:

¡Manos a la obra! Copia el siguiente programa al editor:

```
#include <stdio.h>

int main (int argc, char *argv[] ) {
```

```
    printf(" Hola mundo\n");

}
```

Guarda este fichero con el nombre de `programa.c++`. **Para compilarlo** ejecuta

```
$ g++ programa.c++
```

Seguramente no verás ninguna respuesta al compilar. Como es habitual, la ausencia de respuesta indica que todo ha ido bien. A continuación, puedes ejecutar un listado del contenido del directorio y observarás la existencia de un nuevo fichero llamado "a.out"

La siguiente captura de pantalla corresponde a estos pasos sugeridos

```
lliurex@2DAMASIX:~/Documentos/ejerciciosCPP > g++ programa.c++
lliurex@2DAMASIX:~/Documentos/ejerciciosCPP > ls
a.out  programa.c++
lliurex@2DAMASIX:~/Documentos/ejerciciosCPP >
```

Este fichero nuevo **`a.out`** es el programa ejecutable que ha generado el compilador de c++ a partir del fichero "**`programa.c++`**" que tú guardaste y compilaste

¿Por qué está en verdecito?

Con `ls -l` obtendrás más información que te permitirá descubrir por qué aparece en verde

```
lliurex@2DAMASIX:~/Documentos/ejerciciosCPP > ls -l
total 16
-rwxr-xr-x 1 lliurex lliurex 8518 mar 25 11:48 a.out
-rw-r--r-- 1 lliurex lliurex   85 mar 25 11:25 programa.c++
lliurex@2DAMASIX:~/Documentos/ejerciciosCPP >
```

¿Lo ves? Es un fichero **ejecutable**, tiene el atributo "x" activo. Puedes ejecutarlo invocándolo directamente (con una ruta absoluta o relativa) desde el terminal

```
lliurex@2DAMASIX:~/Documentos/ejerciciosCPP > ./a.out
 Hola mundolliurex@2DAMASIX:~/Documentos/ejerciciosCPP >
```

El nombre del nuevo programa es **`a.out`**, entre otros motivos, porque tú no le has dicho al compilador cómo quieres que sea nombrado dicho fichero que él genera. Prueba a volver a

compilar, pero añadiendo la opción `-o` seguida del nombre que deseas que tenga el programa generado.

```
g++ -o miprograma programa.c++
```

Observa el efecto

```
lliurex@2DAMASIX:~/Documentos/ejerciciosCPP > g++ programa.c++ -
o miprograma
lliurex@2DAMASIX:~/Documentos/ejerciciosCPP > ll
total 28
-rwxr-xr-x 1 lliurex lliurex 8518 mar 25 11:48 a.out*
-rwxr-xr-x 1 lliurex lliurex 8518 mar 25 11:53 miprograma*
-rw-r--r-- 1 lliurex lliurex   85 mar 25 11:25 programa.c++
lliurex@2DAMASIX:~/Documentos/ejerciciosCPP > ./miprograma
 Hola mundolliurex@2DAMASIX:~/Documentos/ejerciciosCPP >
```

¿Ves el nuevo ejecutable generado? Se llama como lo has indicado con la opción `-o` anterior. Sugiero que ahora borres el "a.out" generado anteriormente

Análisis línea a línea del ejemplo

El programa que hemos ejecutado está aquí

```
1    #include <stdio.h>
2
3    int main (int argc, char *argv[] ) {
4        printf(" Hola mundo");
5    }
```

LA explicación de cada línea es:

1. "`#include <stdio.h>`" Esta línea indica que el fichero "`stdio.h`" ha de ser incluido en la compilación. En las siguientes líneas vas a usar algo que está programado en ese fichero y por eso se hace necesario. No necesitas saber más detalles para empezar a programar. Llegado el momento verás la necesidad de incluir otros ficheros.
2. Las líneas en blanco son totalmente prescindibles. Las dejamos por claridad

3. "`int main (int argc, char *argv[] ) {`" Esta línea que es la más larga, es donde empieza una función llamada "`main`". Realmente es donde empieza el programa a ejecutarse. La función devuelve un número entero, de ahí el primer "int" y además, recoge dos parámetros ("`argc`", "`argv`"), que son el número de argumentos y un vector de cadenas de caracteres correspondientes a los argumentos. El símbolo "{" es importante, porque indica que empieza a continuación el *cuerpo* de la función, es decir las instrucciones que lleva dentro

4. `printf` es una función que tienes disponible para usar. En este miniejemplo, es la única instrucción real. Su tarea es imprimir un mensaje por pantalla. El mensaje es "Hola Mundo", que aparece entrecomillado y entre paréntesis.

    a. Entrecomillado porque es una cadena de caracteres literal, si no llevase comillas el compilador pensaría que son variables o funciones (o palabras reservadas)

    b. Entre paréntesis, porque todo argumento que se pase a una función debe ir entre paréntesis

5. En la última línea se termina un bloque de instrucciones … que resulta ser el de la función… que resulta ser la función principal. Por tanto, en esta línea termina el programa.

printf y cout

C++ es una evolución del lenguaje original de C. C++ añade algunas cosas. En este tutorial no seguimos una regla clara en qué cosas usar de C y cuáles de C++. A veces mezclamos. En especial, confundimos `printf` y `cout`: son dos maneras diferentes de mostrar texto por la salida estándar del programa.

```
#include <stdio.h>
#include <iostream>

using namespace std;

int main (int argc, char *argv[] ) {

   cout << "Hola Mundo!"<< endl;
   printf("Hola Mundo\n");
}
```

En el ejemplo anterior, cout y printf logran lo mismo. Lo que está en **negrita** es todo lo necesario para usar **cout** y lo que está resaltado es lo necesario para printf.

Como comparativa, si queremos mostrar el valor de una variable numérica printf y cout difieren bastante en la forma de hacerlo, mientras que printf es una función que puede llevar varios parámetros, cout usa operadores que permiten concatenar expresiones. Por ello puede haber fuertes diferencias en las preferencias entre programadores y nosotros vamos a conocer ambas posibilidades.

```
#include <stdio.h>
#include <iostream>

using namespace std;
int variable;

int main (int argc, char *argv[] ) {
   variable = 7:
   cout << "Variable vale :"<< variable << endl;
   printf("Variable vale %d\n",variable );
}
```

En ambos casos aparece antes que nada unas líneas `#include`. En ellas se solicita la inclusión de un "fichero de cabecera". Éstos son ficheros donde se describen funciones o clases en C o C++. "printf" y "cout" son dos elementos que están definidos dentro de esos ficheros y por tanto, necesitamos que el compilador los conozca. Para ello hay que hacer esas líneas de `#include`. No te preocupes ahora de saber cuándo y cómo hacer ésto ni que ficheros hay que incluir.

Variables

En C se debe declarar la variable. Esto es, se debe *anunciar* que deseamos usar una variable con un nombre que elegimos libremente. Además, se debe indicar qué tipo de dato va a contener dicha variable. Una variable puede contener:

- números enteros
- números con decimales
- letras o símbolos (uno sólo, no varios)
- cadenas de símbolos o cadenas de texto
- direcciones de memoria
- más cosas muy difíciles de entender ahora

Observa el siguiente código

```
1 #include <stdio.h>
2
3 int main (int argc, char *argv[] ) {
4
5   int veces;
6   veces=8;
7   printf(" Hola mundo\n");
}
```

En la línea 5 se le indica al compilador que deseamos usar una variable que contendrá un número.

En la línea 6 usamos dicha variable para almacenar un dato (el número 8)

Entrada y condicionales (if)

(vamos a ir viendo errores, tanto de compilación como de ejecución)

Problema de ejemplo:

Haz un programa que lea un número por el teclado e indique si dicho número es mayor o menor que 10.

Posible solución:

La siguiente solución tiene un problema gordo (lo veremos después).

```
#include <stdio.h>
#include <iostream>

using namespace std;

int main (int argc, char *argv[] ) {

    int numeroLeido;
    cin >> numeroLeido;

    if (numeroLeido < 10 )
       cout << "El número leído es " << numeroLeido << endl;
       cout << "El número leído es menor que 10 " << endl;
    else
      cout << "El número leído es " << numeroLeido << endl;
      cout << "El número Leído es mayor que 10"<< endl;
```

```
    cout << "He terminado  " << endl;
}
```

La línea en negrita y subrayada es la forma en la que en **C++** se puede leer un dato desde el teclado. En C la forma de hacerlo es también arcaica y no vamos a tratarla.

```
cin >> numeroLeido;
```

El operador ">>" da la impresión de ser una flecha o una cañería que transporta datos desde "cin" (entrada) hasta la variable "numeroLeido". Observa que esta variable es entera (`int`) y este hecho es lo que determina que leamos un número entero del teclado.

En el código anterior hay un error ¿¡ Adivinas qué está mal?!

Recuerda: Sé disciplinado con la posición de las instrucciones y fíjate en las llaves. En el anterior ejemplo, al no poner llaves, el segundo `cout` posterior al `if` está fuera

La solución correcta es la siguiente, observa las llaves puestas :

```
#include <stdio.h>
#include <iostream>

using namespace std;

int main (int argc, char *argv[] ) {

    int numeroLeido;
    cin >> numeroLeido;

    if (numeroLeido < 10 ){
       cout << "El número leído es " << numeroLeido << endl;
       cout << "El número leído es menor que 10 " << endl;
    }
    else {
      cout << "El número leído es " << numeroLeido << endl;
      cout << "El número Leído es mayor que 10"<< endl;
    }
    cout << "He terminado  " << endl;
}
```

Ejercicio para hacer 2021-2022:

Haz un programa que lea tres números y te indique si el primero es mayor que los dos restantes.

Bucles

Después de esta prueba mínima en la que hemos aprendido cómo hacer un fichero, compilarlo y generar el ejecutable; vamos a avanzar a programas algo más complejos que nos muestren cómo se hacen las cosas en C.

Enunciado de nuestro problema: El programa "*mostrará 8 veces un mensaje de saludo por la pantalla*".

Evidentemente, repetir varias veces la línea siguiente...

```
printf(" Hola mundo\n"); // o cout << "Hola Mundo" << endl;
```

Consigue lo que pide el enunciado.

```
printf(" Hola mundo\n");
printf(" Hola mundo\n");
printf(" Hola mundo\n");
...
printf(" Hola mundo\n");
```

Pero no vamos a caer en esta simpleza. Sino que vamos a hacer un **bucle** que repita la ejecución de dicha línea y vamos a controlar y determinar **cuántas veces se repite** dicho bucle. Es decir: sólo habrá una línea "printf", pero se ejecutará varias veces.

Para repetir la ejecución de una línea, podemos usar principalmente dos estructuras de control de bucles: un `while` o un `for`. Podemos usar `for` porque sabemos cuántas veces vamos a repetir el bucle, pero para aprender bien, vamos a usar while preferentemente al inicio de curso.

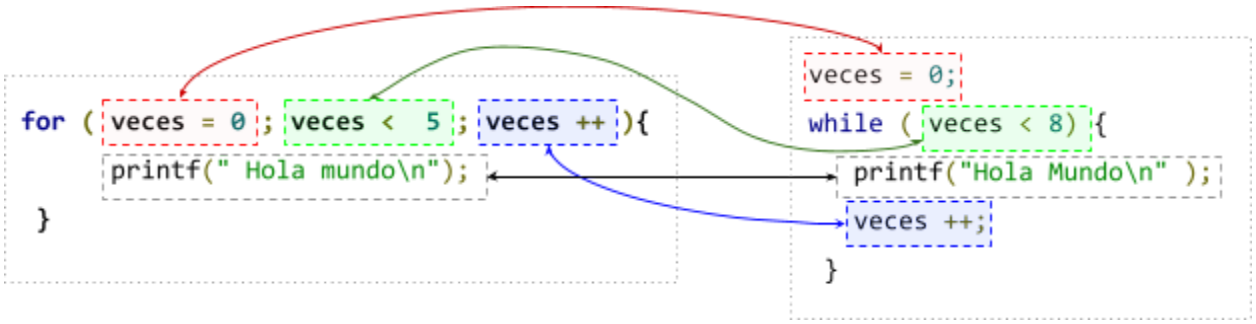
```
 #include <stdio.h>

int main (int argc, char *argv[] ) {

   int veces;
   veces = 0;
   while ( veces < 8)
            printf("Hola Mundo\n" );
```

```
            veces ++;
    }
}
```

El anterior bucle equivale al siguiente

```
1 #include <stdio.h>
2
3 int main (int argc, char *argv[] ) {
4
5   int veces;
6   for (veces = 0 ; veces <  8; veces ++ ) {
7       printf(" Hola mundo\n");
8   }
}
```

Existe una equivalencia entre un bucle while y uno for (se pueden traducir la mayoría de veces de uno a otro)



Ejercicio para hacer 2021-2022:

Hacer un programa que pida números repetidamente hasta que el número introducido sea mayor que 10, o la suma de los números introducidos mayor que 20

Vectores!!!

Los vectores en C son muy primitivos y su concepción es muy diferente a Java, aunque al usarlos no aparenten diferencias

En este documento vamos a crear y manipular un vector sin complicarnos mucho la vida, pero asumiendo que ya sabes algo de vectores. Si necesitas más detalle y explicación, visita este otro documento donde se tratan los vectores con profundidad y desde el principio

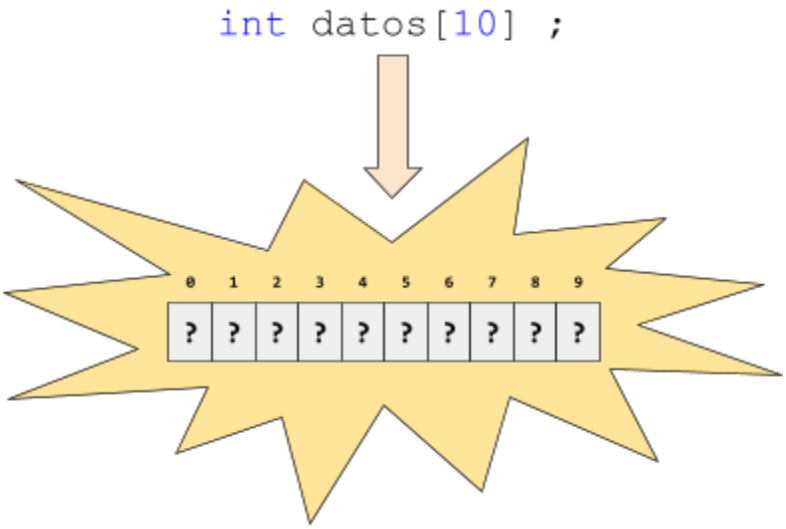
Al igual que la variable, un vector, hay que declararlo, indicando no sólo qué nombre tiene, sino cuántos elementos y de qué tipo son todos y cada uno de sus elementos. Ésto se realiza de la manera que se ve en el siguiente ejemplo:

```
#include <stdio.h>
2
3 int main (int argc, char *argv[] ) {
4
5   int numeros[10];
6   numeros[0]=1;
7   numeros[9]=12;
    cout << " el valor del elemento último es " << numeros[9] << endl;
}
```

En la línea 5 es donde se declara y crea el vector. Lo que allí aparece se explica a continuación:

- "**numeros**" es el nombre del vector.
- **[10]** : El tamaño, aquí se indica que es un vector de 10 elementos
- **int** : indica que cada elemento almacenará un número entero

Crear un vector parece similar a Java, pero es muy diferente... pero ¡Mucho! En la línea 5 anterior se crea el vector y sus 10 elementos. la siguiente imagen trata de expresar lo que ocurre:

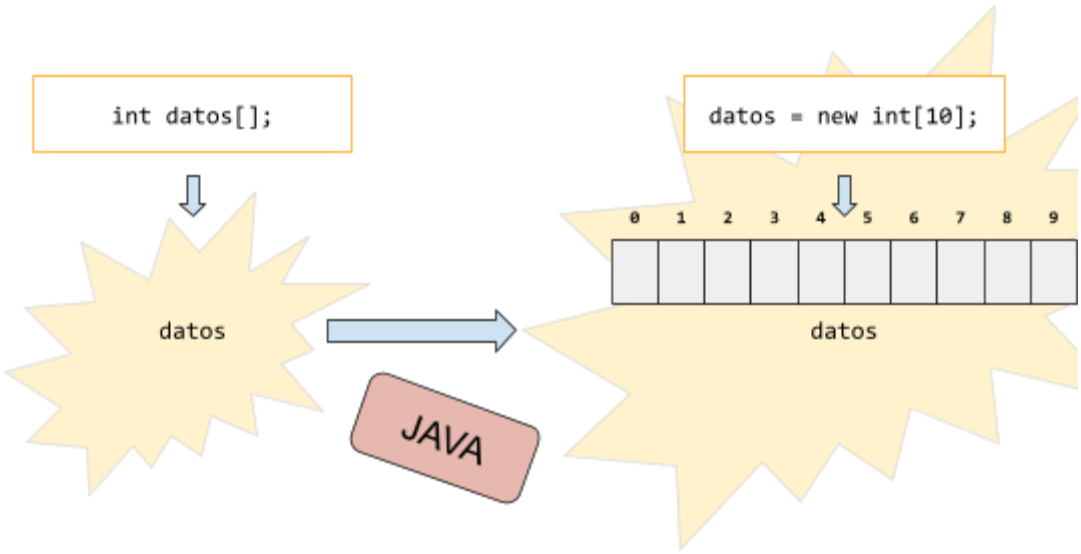


Con java, la declaración y el efecto sería distinto. La línea siguiente,

```
int datos[];
```

En java o crea un vector, sino que simplemente reserva una palabra "datos" para designar a algo que será un vector. Pero no se crea nada.

En java hay que crear después el vector



Por ello en C se puede hacer lo siguiente sin problemas

```
int numeros[10];
numeros[0]=1;
```

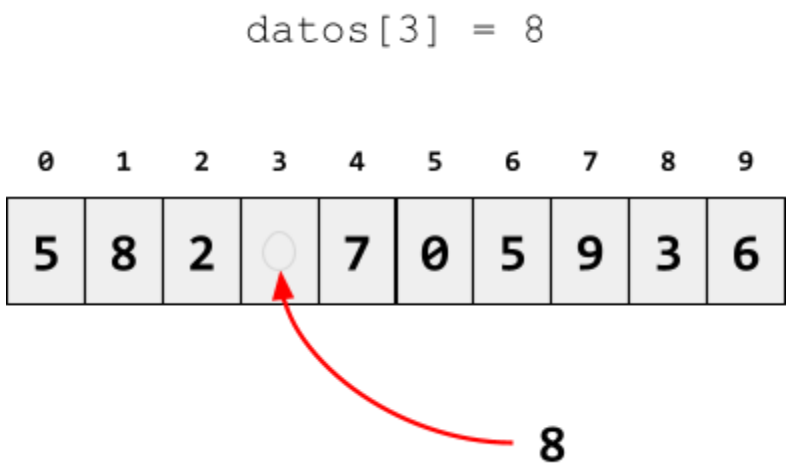
Porque cuando se ejecuta la segunda línea, el vector ya existe (todos sus elementos ya existen), mientras que en Java no. Antes de acceder al elemento, habría que hacer:

```
numeros = new int[10];
```

para poder usar el vector, quedando las anteriores líneas así:

```
int numeros[]; //esto es java
numeros = new int[10];
numeros[0]=1;
```

Por lo demás, el acceso a los elementos de un vector es igual que en Java.



Recorrido de un vector.

Recuerda:

> (en general) No se puede acceder a todos los elementos de un vector **de golpe**, hay que ir elemento por elemento

Para ir elemento a elemento, usamos un número que se incrementa, y ese número lo usamos como **índice** que selecciona a cada elemento

Por ejemplo, el siguiente código inicializa todos los elementos de un vector:

```
#include <stdio.h>
int main (int argc, char *argv[] ) {

    int numeros[10];

    numeros[0]=5;
    numeros[1]=6;
    numeros[2]=3;
    numeros[3]=7;
```

```
    numeros[4]=9;
    numeros[5]=6;
    numeros[6]=3;
    numeros[7]=2;
    numeros[8]=8;
    numeros[9]=1;
}
```

El anterior ejemplo, realmente no es un recorrido, sino una asignación "manual" uno por uno a todos los elementos. Vamos a hacer el bucle que recorre el vector. Para ello usaremos la estructura "for" o while. Ambas tienen tres elementos de control:

- Una "**instrucción de inicialización**". Se ejecuta **antes** de empezar nada y sólo una vez.
- Una "**condición de continuación**". Se comprueba **cada vez**, antes de empezar el cuerpo del bucle (las instrucciones que hay "dentro" del for)
- Una **instrucción de interacción**. Se ejecuta **cada vez** después de ejecutar el cuerpo del bucle.

```
for (inicialización ; condición continuación; instr. iteración ) {
            cuerpo
}
```

La forma general de recorrer un vector es:

```
int i=0;

while (i < tamanyoVector) {

    //acceder a numeros[i]

    i = i + 1;

}
```

`tamanyoVector` es el número de elementos del vector. En C, esto debe saberlo el programador (en los ejemplos estamos usando vectores de 10 elementos, por lo que tamanyoVector sería 10). Con C++ podremos usar otro tipo de vectores donde el tamaño puede ser consultado y cambiado a cada momento

Con while podemos igualmente recorrer un vector elemento a elemento.

```
for (int i=0 ; i < tamnayoVector; i++ ) {
           //acceder a numeros[i]
}
```

Por tanto la estrategia para recorrer un vector es tener una variable numérica entera adicional. Esta variable será un índice que usaremos para elegir cada vez un elemento diferente del vector.

En el siguiente bucle se recorre el vector para inicializar todos sus elementos a 8;

```
#include <stdio.h>

int main (int argc, char *argv[] ) {
    int numeros[10];
    int indice = 0;
    while (indice < 10 ) {
       numeros[indice] = 8;
       indice = indice + 1 ;
    }
}
```

En el siguiente programa se ha añadido otro recorrido del vector, esta vez para mostrar el contenido del mismo

```
#include <stdio.h>

int main (int argc, char *argv[] ) {

    int numeros[10];
    int indice;

    for ( indice=0; indice < 10; indice++)
       numeros[indice] = 8;


    printf("Voy a mostrar el vector por pantalla\n");
    for ( indice=0; indice < 10; indice++)
       cout << numeros[indice] << " " ;
}
```

Ejercicio para hacer 2021-2022:

Haz un programa que muestre los elementos en posiciones impares o aquellos cuyo valor es menor que 5. (Obviamente inicializa los elementos a unos valores adecuados para obtener una buena respuesta, no hagas todos los elementos mayores que 5, por ejemplo)

Inicialización ràpida de un vector

Un vector puede inicializarse en el momento de ser declarado de la siguiente forma:

```
int numeros[10] = {5,6,3,7,9,6,3,2,8,1}

// inicialización menos compacta equivalente
numeros[0]=5;
numeros[1]=6;
numeros[2]=3;
numeros[3]=7;
numeros[4]=9;
numeros[5]=6;
numeros[6]=3;
numeros[7]=2;
numeros[8]=8;
numeros[9]=1;
```

Esta inicialización permite ahorrar líneas y es equivalente a la hecha anteriormente

Inicializar un vector con números aleatorios

En ocasiones necesitaremos que los datos de los vectores sean aleatorios o que existen muchos datos (miles) haciendo imposible inicializarlos de la forma anterior. En estos casos vamos a aprovechar una función existente que genera números pseudoAleatorios (cada vez que se ejecuta el programa se repite la secuencia, pero es aleatoria). La función se llama `random()`. Busca ayuda en la consola de línux escribiendo "man rand" y verás una página que muestra la información necesaria para usar esa y otras funciones similares. Básicamente para usar rand hay que solicitar el uso de un fichero de cabecera externo "`#include <stdlib.h>`". `random()` genera números muy grandes cuando normalmente necesitaremos números pequeños. La forma de obtener un número pequeño a partir de uno grande es

mediante el resto de la división. Por ejemplo, para obtener números aleatorios entre 0 y 99 se puede hacer la siguiente operación.

```
int numero;
numero = random() % 100;
```

Con un vector, hay que hacer eso mismo pero con todos los elementos del vector

```
for ( indice=0; indice < 10; indice++) {
        numeros[indice] = random() % 100;
}
```

a partir de ahora, los ejercicios con vectores de números en C tendrán el bloque anterior de código para tener desde un primer momento datos aleatorios en el vector (pseudoaleatorios)

Buscar el máximo de un vector

Como primer ejercicio de análisis de un vector, vamos a programar un algoritmo que encuentre el número más alto del vector.

Cuando tengas delante un problema como éste, intenta imaginar un ejemplo de la vida real en el que debas obtener algo similar. Por ejemplo: Imagina que tienes varias entrevistas de trabajo en las que te informan del sueldo que ofrecen. Evidentemente, tú quieres el salario mejor y es lo único que va a determinar tu elección: vas a quedarte con el trabajo mejor pagado. Fíjate en las siguientes evidencias:

- Hasta que no termines la última entrevista SEGURO que no puedes decidirte.
- Si en la primera entrevista te ofrecen 1000 €/mes, antes de empezar la segunda entrevista, 1000 €/mes es lo máximo que has conseguido… hasta ese momento
- Si en la tercera entrevista te ofrecen 2000 €/mes, seguro que olvidas la primera entrevista. Lo que no sabemos es si también olvidarás la segunda, pero de momento debes proseguir acordándote de esta segunda entrevista.
- Seguro que cuando vayas a una entrevista cualquiera no tendrás que acordarte de todas las anteriores tan sólo necesitarás acordarte de la oferta más alta recibida hasta ese momento.

Por todo ello podemos coincidir en la siguiente idea:

Básicamente, cuando vayamos a una entrevista, nuestra misión será actualizar el recuerdo de cuál es el salario mejor que hay:

- Si en la entrevista nos dan un salario más alto que el recordado, entonces actualizamos nuestro "salario más alto"
- Si en la entrevista no nos ofrecen algo más elevado, no hacemos nada

Para hacer el programa, partimos de una variable que llamaremos "numeroMaximo", y a cada iteración del bucle, la actualizaremos o no, según el valor del elemento del vector.

```
int numeroMaximo;
for ( indice=0; indice < 10; indice++) {
       if (numeros[indice] > numeroMaximo)
            numeroMaximo = numeros[indice]
}
```

Hay un grave error en el código anterior. La variable `numeroMaximo` se utiliza sin inicializar. Eso es pecado Mortal y si lo haces tendrás que confesarte o irás al infierno.

¿Cómo inicializamos esa variable? ¿ Le ponemos un 0? ¿Y si el vector tiene todos sus elementos con valores negativos?

Recuerda los razonamientos anteriores "Si en el primer trabajo te ofrecen 1000 euros, tu salario más alto en ese momento son esos 1000 euros al mes". Por tanto la mejor inicialización consiste en tomar el primer elemento como el máximo, incluso aunque en el bucle se vuelva a examinar ese primer elemento otra vez

```
int numeroMaximo = numeros[0];
for ( indice=0; indice < 10; indice++) {
       if (numeros[indice] > numeroMaximo)
            numeroMaximo = numeros[indice]
 }
```

Al salir del bucle for, la variable numeroMaximo tiene el valor del elemento más alto del vector. Sin embargo, no sabemos la posición de éste elemento.

Ejercicio: Usando una sola variable llamada posMax (además de "indice") modifica el programa para que se calcule el máximo y la posición que ocupa.

Ejercicios para hacer 2021-2022:

Todos los ejercicios de a continuación están basados en usar vectores de enteros en los que se almacenan notas obtenidas en exámenes. En algún ejercicio puedes necesitar

dos vectores diferentes. Todos los vectores son de 10 elementos y puedes aprovechar esta circunstancia para simplificar las comprobaciones de tamaños.

- Encuentra el número de aprobados
- Encuentra la media de las notas
- Encuentra la mayor diferencia en el mismo examen entre las notas obtenidas entre dos alumnos. (Hay dos vectores, uno por cada alumno, ambos con el mísmo número de notas pero distinto valor)

Ejercicios para hacer 2021-2022:

Observa el siguiente vídeo en el que se plantea y resuelve un problema de programación sobre series de números

https://youtu.be/JMCZMdiW-98

La explicación está basada en el lenguaje de programación bash y los datos numéricos son pasados por argumento.

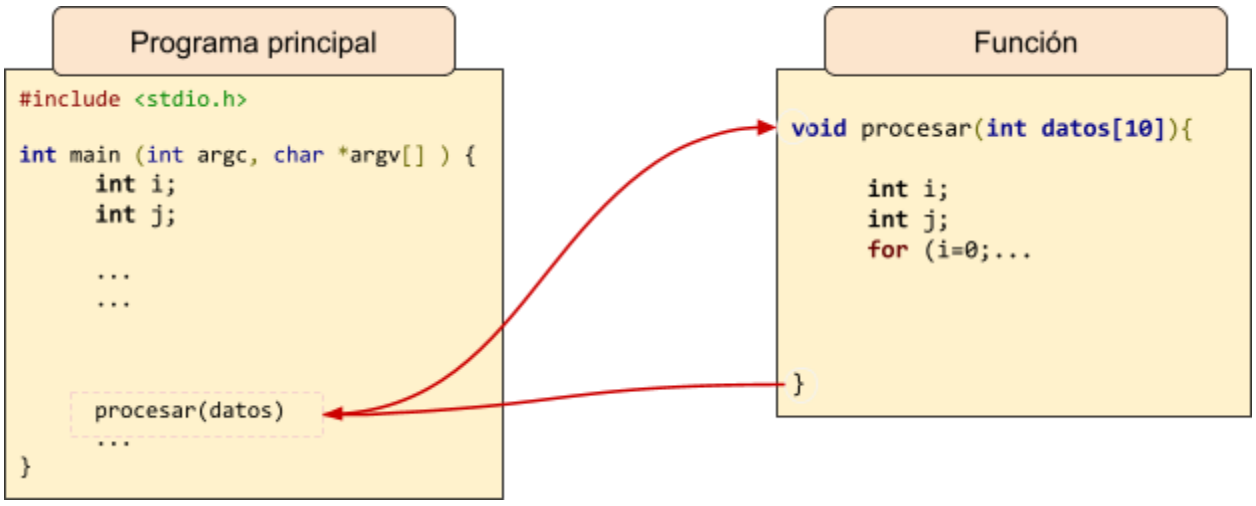
Se pide:

Resuelve el problema usando el lenguaje C y teniendo los datos en un vector en vez de leerlos por la entrada.

Funciones

Una función es un bloque de código (varias líneas) que se puede ejecutar mediante una llamada simple. Pero tiene algunas particularidades:

- Las funciones existen independientemente del programa principal
- Las funciones no pueden acceder a las variables del programa principal
- Las funciones son :
    - Programadas, creadas, implementadas, etc. por una persona
    - Usadas, llamadas, invocadas por otra parte desde un programa potencialmente hecho por otra persona.



La función está programada a la derecha, y se llama o invoca desde el programa principal hecho a la izquierda

Una función define siempre las siguientes 4 cosas:

- Nombre que recibe la función
- Valor de retorno
- Datos que recoge (parámetros)
- Acción que realiza

La declaración e implementación de la función incluyen todos estos elementos. Observa los colores para encajar cada parte del ejemplo siguiente con el elemento de la lista anterior. La siguiente función está pensada para mostrar por pantalla los valores de los elementos de un vector

```
void mostrar(int vectorcillo[10]){

    instrucción 1;

    instrucción 2;

    instrucción 3;

}
```

La función no conoce el vector que estamos usando (`"numeros"`), pero está preparada para, cuando sea llamada, recoger y operar con cualquier vector de diez posiciones enteras. En ese momento, sea cual sea el vector pasado, se conocerá como `verctorcillo` y se trabajara con él

```
void mostrar(int vectorcillo[10]){
    cout << "Voy a mostrar el vector por pantalla"<< endÍ;
    for ( indice=0; indice < 10; indice++)
      cout << vectorcillo[indice]<<" " ;

    cout << endl;
}
```

Falla la compilación ¿Por qué?

Solución definitiva (copiable y pegable para probar). El programa principal realiza una función muy simple. Adivínala

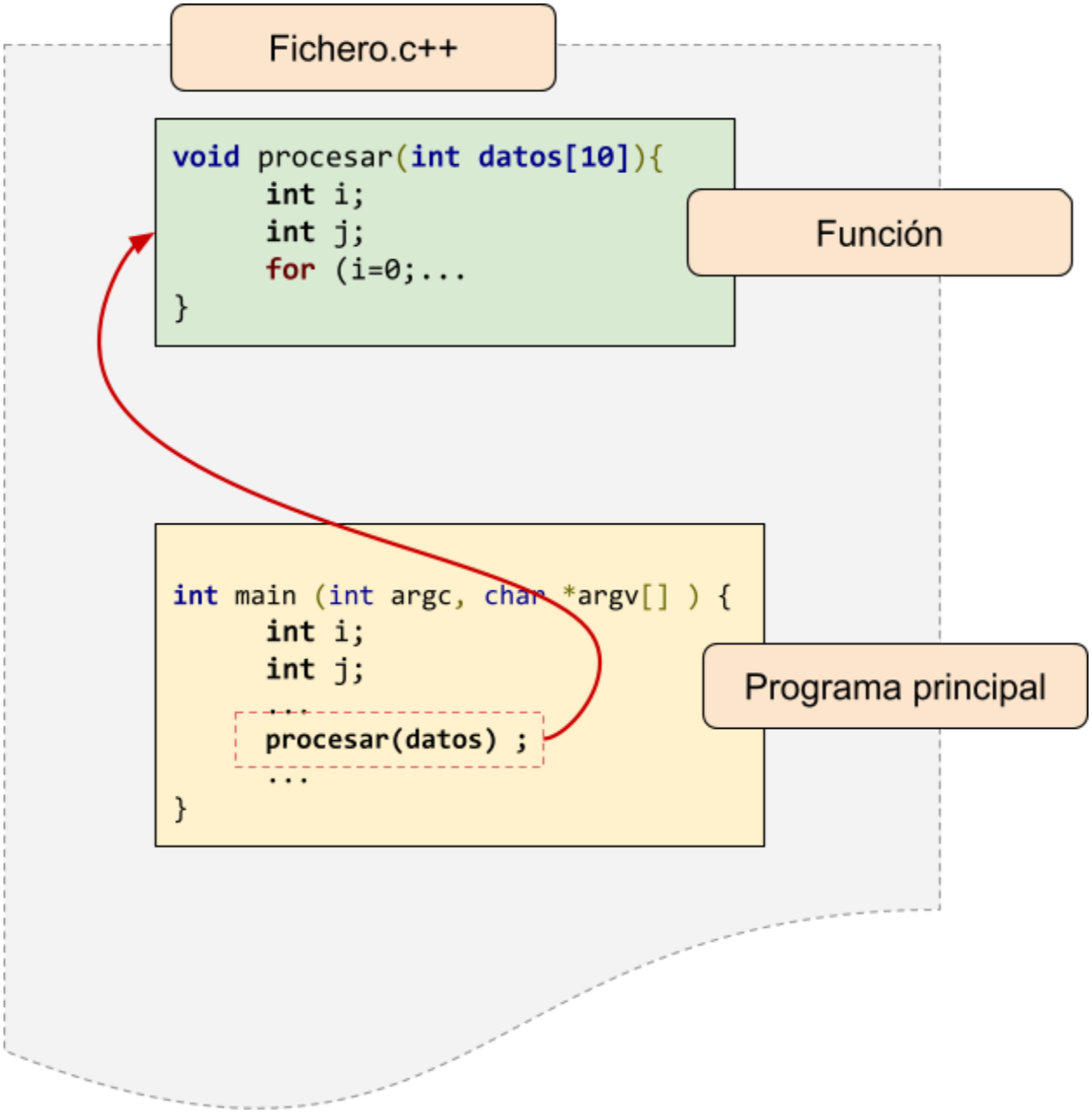
```
#include <stdio.h>

void mostrar(int vectorcillo[10]){
    int indice; // solución al fallo de compilación
    cout << "Voy a mostrar el vector por pantalla"<< endÍ;
    for ( indice=0; indice < 10; indice++)
      cout << vectorcillo[indice]<<" " ;

    cout << endl;
}

int main (int argc, char *argv[] ) {
    int vicent[10];
    int indice;

    for ( indice=0; indice < 10; indice++) {
      vicent[indice] = 10-indice;
    }

    mostrar (vicent);
    printf("\n ahora voy a quitar las posciones pares\n");

    for ( indice=0; indice < 10; indice++) {
```

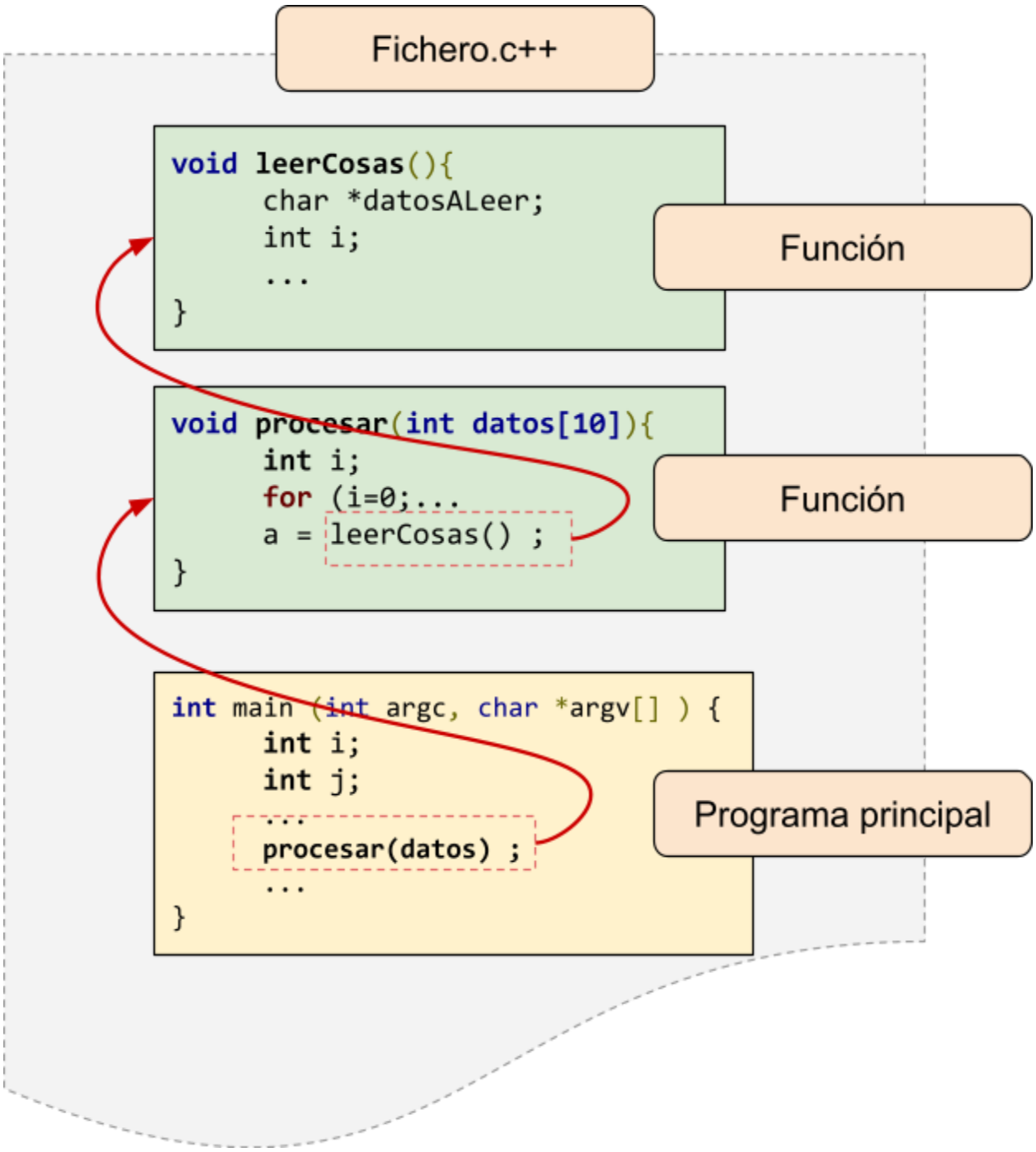
```
        if (indice%2 != 0) vicent[indice] = 0;
    }
    mostrar (vicent);
}
```

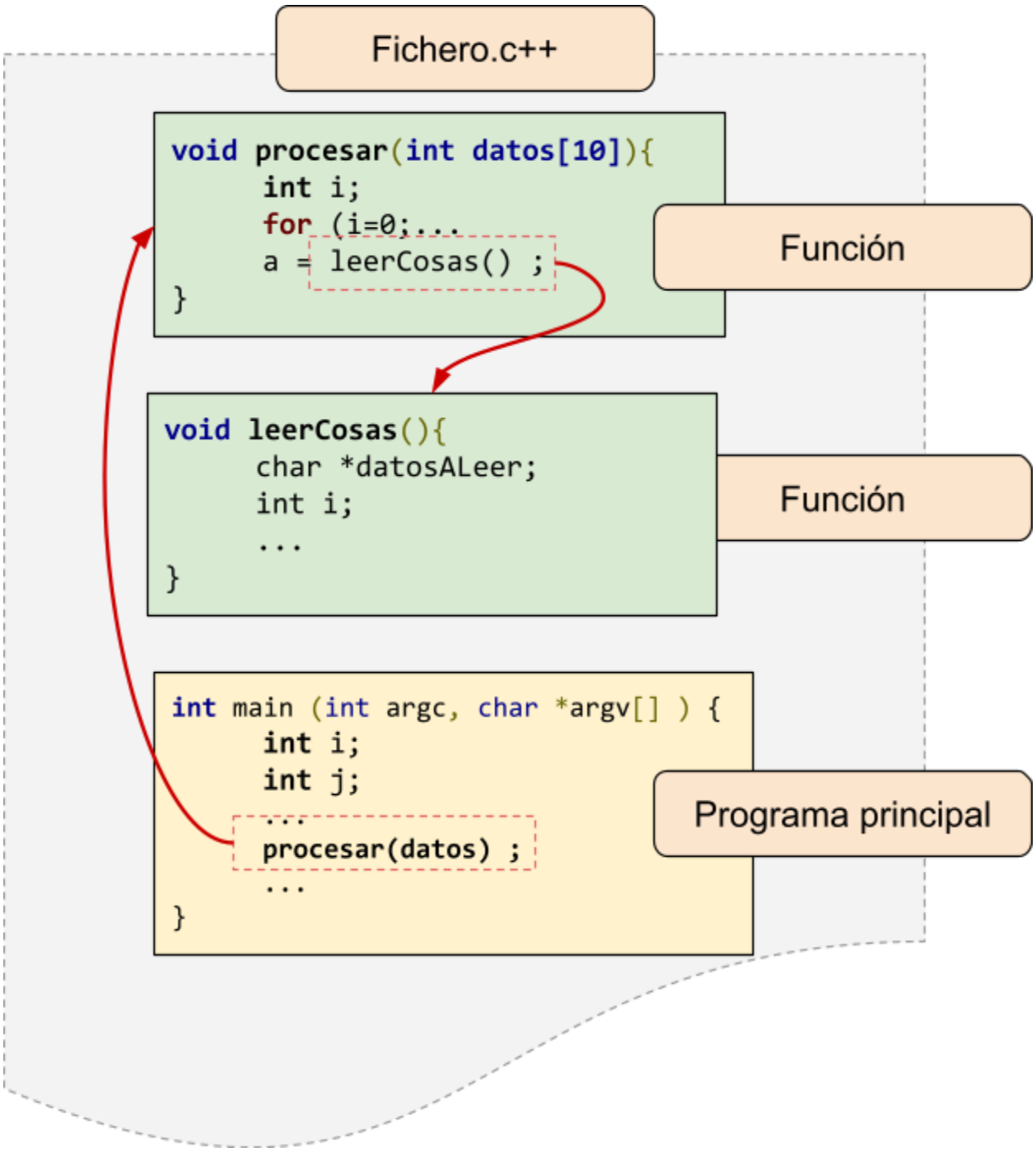
Ejercicios para hacer 2021-2022:

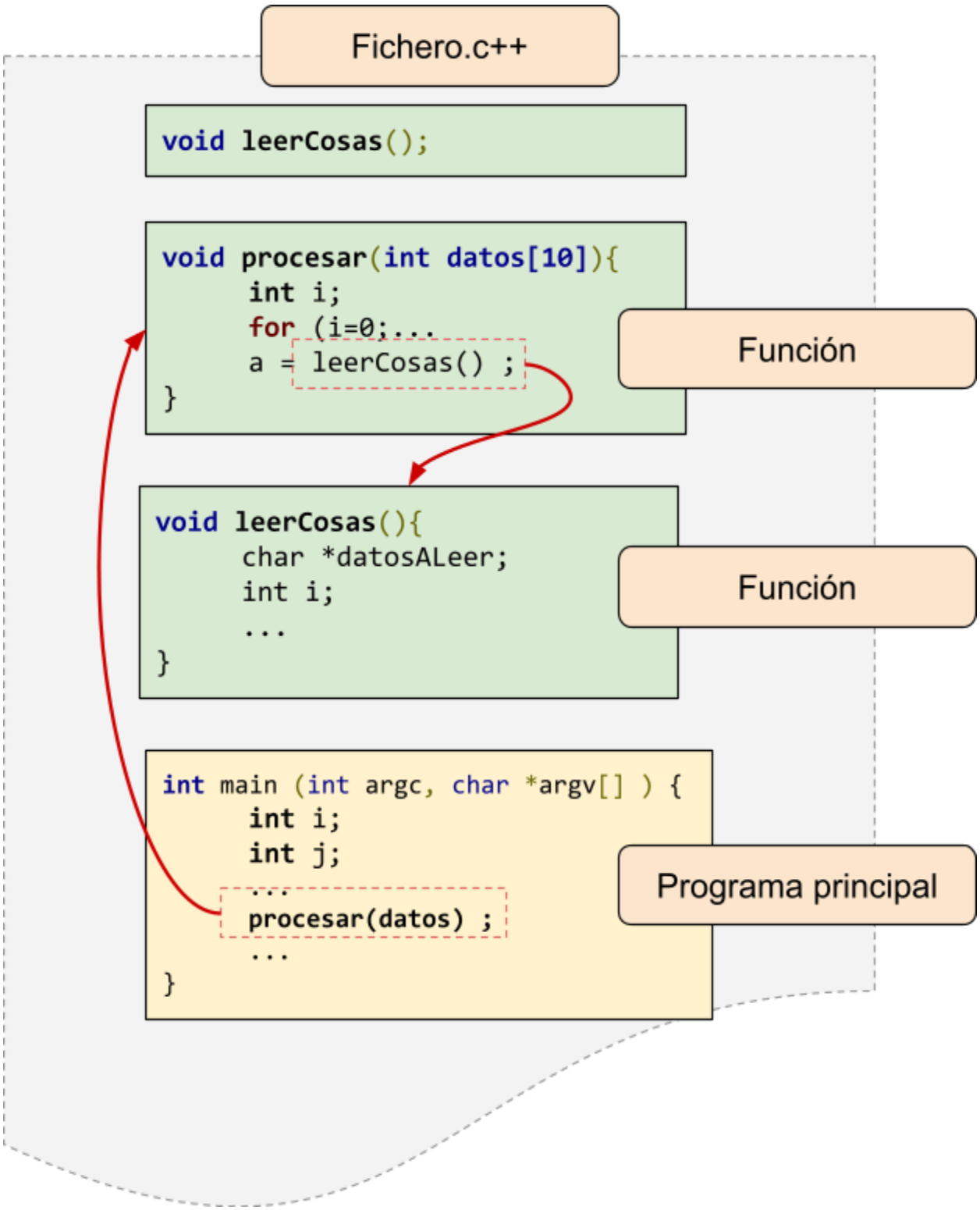
- Haz una función que obtenga la media en coma flotante de los elementos del vector comprendidos entre dos posiciones
- Obtén la declaración de una función que obtenga un vector con los valores de la tabla de multiplicar de dos números menores que 10.
- Haz una función que indique si dos elementos del vector están repetidos.

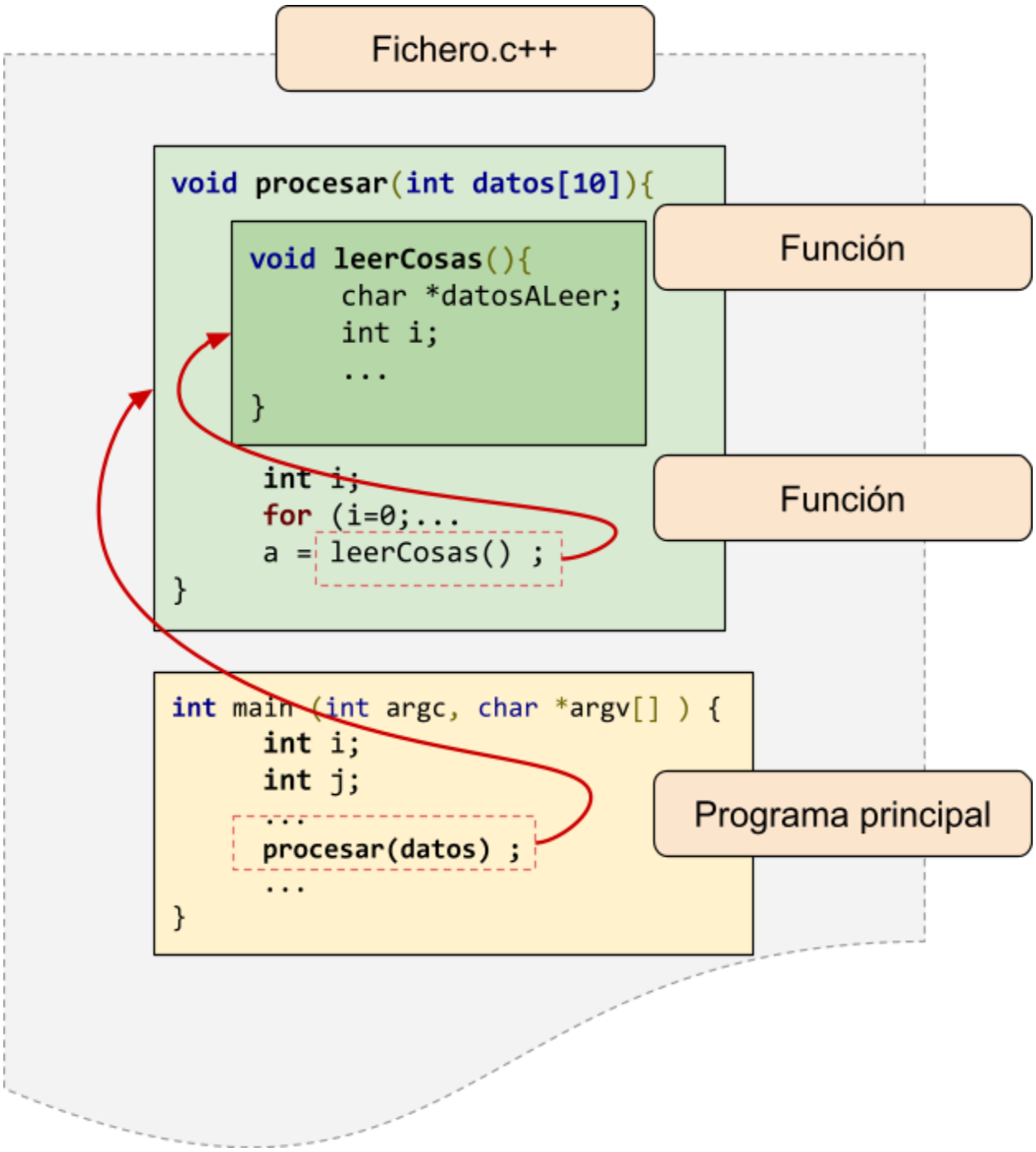
Declaración, implementación y llamadas. Orden











FIN PROVISIONAL VERSIÓN SEMIPRESENCIAL

Indicar si un vector es capicúa

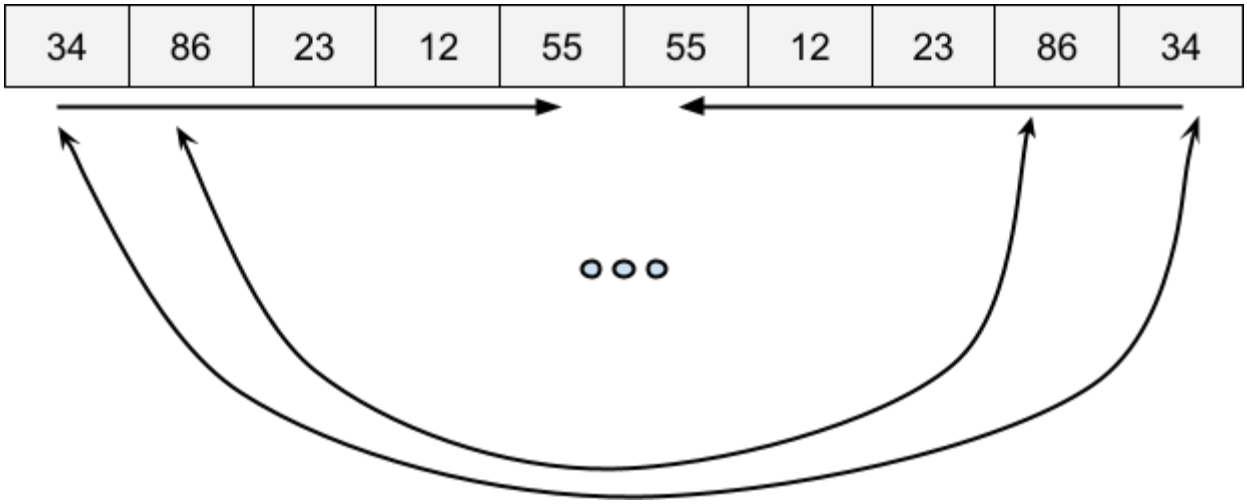
Un vector capicúa es el que muestra los mismos valores en sus elementos si es recorrido desde el primero al último o desde el último al primero.

34	86	23	12	55	55	12	23	86	34

Como siempre, no podemos tratar todos los elementos a la vez, hay que ir uno a uno, **o dos a dos en este caso**. Procedimiento:

- Hay que mirar el primero y el último, ver si son iguales
- Mirar el segundo y el penúltimo y ver si son iguales
- El tercero y el antepenúltimo

El recorrido de los elementos del vector es el que muestra la siguiente imagen



Para recorrer el vector con un bucle for, hay que darse cuenta de una circunstancia: antes sólo había un índice, una variable entera que se iba incrementando. El tipo de recorrido que necesitamos aquí en principio, es diferente, necesitamos coger elementos desde el principio hasta el final, necesitamos hacer las siguientes comparaciones:

i	Elementos que se comparan

0	numeros[0]	numeros[9]
1	numeros[1]	numeros[8]
2	numeros[2]	numeros[7]
3	numeros[3]	numeros[6]

Hace falta un poco de "idea feliz" para ver que se está comparando

```
numeros[i] == numeros[9-i]
```

El bucle por tanto es

```
tamanyo=10
for ( indice=0; indice < tamanyo/2; indice++) {
      if (numeros[indice] == numeros[tamanyo -1 -indice])

 }
```

Cuestiones a reflexionar:

- Hay que responder Sí o No. Ese es el único objetivo del trozo de programa que vamos a realizar ahora
- En principio recorreremos todo el vector.
- Al empezar a recorrer tenemos tres opciones:
    - No pensar si el vector es capicúa o no.
    - Pensar que no lo es e intentar descubrir si acaso el vector es capicúa.
    - Pensar que el vector sí es capicúa e intentar descubrir si alguna pareja falla
- Cada vez compararemos dos elementos del vector
    - Si una pareja de elementos de las que se examina son iguales, no se toma ninguna decisión.
    - Si la pareja es diferente podemos tomar la decisión de apuntar en algún lugar que el vector no es capicúa

Con ello podemos tomar una primera aproximación (con error) a la solución.

```
int tamanyo=10
for ( indice=0; indice < tamanyo/2; indice++) {
```

```
            if (numeros[indice] == numeros[tamanyo -1 -indice])
                  capicua = true;
            else
                  capicua = false;
      }
```

Como hemos dicho, si una pareja no es capicúa, entonces YA podemos decir que el vector entero no será capicúa. Pero tal y como está la solución anterior, lo único que hacemos es decir si la última pareja comparada fue igual o no. Ojo! Esto no es lo mismo que el vector y por tanto nuestra solución hasta el momento, está mal. No podemos decir capicua = true cuando una pareja es igual, porque una pareja no garantiza que el vector sea capicúa. Sin embargo, al revés sí que podemos tomar una decisión tajante: Si los elementos de una pareja no son iguales, entonces el vector no es capicúa.

```
      int tamanyo=10;
      bool capicua = true;
      for ( indice=0; indice < tamanyo/2; indice++) {
            if (numeros[indice] != numeros[tamanyo -1 -indice])
                  capicua = false;
      }
```

Esta solución ya funciona bien. Primero pensamos que el vector será capicúa. Si todas las parejas son iguales, no se cambia nada y esa idea inicial es la que termina valiendo (capicua=true). Pero si alguna pareja "falla", entonces capicúa será false aunque el resto de las parejas sean iguales.

Aunque la anterior solución funciona, no es óptima, puesto que se recorre todo el vector aunque se descubra pronto que no es capicúa. Hay que optimizar atajando el bucle cuando una pareja es diferente.

```
      int tamanyo=10;
      bool capicua = true;
      for ( indice=0; indice < tamanyo/2; indice++) {
            if (numeros[indice] != numeros[tamanyo -1 -indice])
                  capicua = false;
                  break;
      }
```

Una solución más compacta:

```
int tamanyo=10;
```

```
for ( i=0;
      ( i < tamanyo/2) && (numeros[i] != numeros[tamanyo -1 -i]) ;
      i++)
bool capicua = i == (tamanyo/2 );
```

Ordenación de vectores

Recordemos el ejercicio hecho en shellscript que ordenaba un vector

```
#!/bin/bash

datos=( 10 2 32 14 5 )


tamanyo=5

function  mostrar() {
indice=0;
 while [ $indice -lt $tamanyo ]  ; do
    echo -n "${datos[${indice}]} "
    let indice++
 done
 echo
}


#aux=${datos[0]}
#datos[0]=${datos[1]}
#datos[1]=$aux

mostrar
echo

contador=0;
cambios=SI
while [ $contador -lt $tamanyo ] && [ $cambio == SI ] ;  do
    posicion=0;
    final=$[ $tamanyo - 1 ]
    while [  $posicion -lt $final ] ; do
       siguiente=$[ $posicion + 1 ];

       if [ ${datos[$posicion]} -gt ${datos[$siguiente]} ] ; then
```

```
            aux=${datos[$posicion]}
            datos[$posicion]=${datos[$siguiente]}
            datos[$siguiente]=$aux
            cambios=SI
        fi
        let posicion++
    done
    mostrar
    contador=$(( contador + 1 ))
done
```

Ejercicio para hacer 2021-2022:

Inicializar un vector rápidamente en C++. Dos formas

```
    /* primera forma de inicializar un vector */
    int numeros[10]={8,2, 5,14,8,9,1,6,0,11};
    /* segunda forma de inicializar un vector */
    int indice;

    numeros[0]=8;
    numeros[1]=2;
    numeros[2]=5;
    numeros[3]=14;
    numeros[4]=8;
    numeros[5]=9;
    numeros[6]=1;
    numeros[7]=6;
    numeros[8]=0;
    numeros[9]=11;
```

Preparemos el trabajo. Primero mostraremos el vector por la pantalla, después ordenaremos (o haremos lo que podamos), y después volveremos a mostrar el vector por la pantalla para que el programador vea el resultado.

```
mostrar (numeros);
// vamos a ordenar el vector aquí:


// una vez ordenado, mostramos el vector
mostrar (numeros);
```

Primero implementaremos "una pasada al vector", no la ordenación entera y aquí ya aparecen

errores típicos:

1. Usar otros contadores adicionales al del for
2. Comparar elementos con índices
3. Usar una variable auxiliar siguiente = índice + 1
4. Comparar con -gt
5. No saber intercambiar dos elementos de un vector
6. Salirse del vector

Error: usar otros contadores adicionales al natural del for:

```
    for ( indice = 0 ; indice < 10; indice ++ )
           siguiente = indice + 1;
          if ( numeros[indice] > numeros[siguiente]


    for ( indice = 0 ; indice < 10; indice ++ )
     if ( numeros[indice] > numeros[indice + 1]
```

No abrir llaves y dejar *fuera del for* instrucciones que deberían estar dentro

```
   for ( indice = 0 ; indice < 10; indice ++ )
     siguiente = indice + 1 ;
     if ( numeros[indice] > numeros[siguiente] )
```

Comparar elementos con índices

```
     for ( indice = 0 ; indice < 10; indice ++ ) {
      siguiente = indice + 1 ;
      if ( numeros[indice] > siguiente )

    }


    for ( indice = 0 ; indice < 10; indice ++ ) {
       siguiente = numeros [indice + 1 ];
       if ( numeros[indice] > siguiente )

    }
```

intercambiar mal el vector

```
     if ( numeros[indice] > numeros[ indice +1 ] ){
           numeros[indice] = numeros[ indice +1 ]
      }


     if ( numeros[indice] > numeros[ indice +1 ] ){
           int aux = numeros[indice];
           numeros[indice] = numeros[ indice +1 ];
           numerso[indice  + 1 ] = aux;
      }
```

Salirse del vector

```
// ¿?Qué pasa cuando índice vale 9
  for ( indice = 0 ; indice < 10 ; indice ++ ) {
```

```
            // "indice+1" VALE 10!!! esto cae fuera del vector
        if ( numeros[indice] > numeros[ indice +1 ] ){
              int aux = numeros[indice];
              numeros[indice] = numeros[ indice +1 ];
              numeros[indice  + 1 ] = aux;
        }
    }



// solución, no hay que llegar al final del vector, sino al elemento
penúltimo
// el penúltimo se compara con el último
  for ( indice = 0 ; indice < (10 - 1) ; indice ++ ) {
        if ( numeros[indice] > numeros[ indice +1 ] ){
              int aux = numeros[indice];
              numeros[indice] = numeros[ indice +1 ];
              numeros[indice  + 1 ] = aux;
        }
    }
```

Solución parcial a dar una pasada **copiable y pegable**

```
#include <stdio.h>

void mostrar(int vectorcillo[10]){
    int indice;
    printf("Voy a mostrar el vector por pantalla\n");
    for ( indice=0; indice < 10; indice++)
      printf("%d ", vectorcillo[indice]);
    printf("\n"); //esto está fuera del for

}

int main (int argc, char *argv[] ) {
    int numeros[10]={6,2,7,14,8,2,1,9,5,1};
    int indice;


    mostrar (numeros);
// vamos a ordenar el vector aquí:
```

```
    for ( indice = 0 ; indice < (10 - 1) ; indice ++ ) {
      if ( numeros[indice] > numeros[ indice +1 ] ){
            int aux = numeros[indice];
            numeros[indice] = numeros[ indice +1 ];
            numeros[indice  + 1 ] = aux;
      }
    }
// una vez ordenado, mostramos el vector
    mostrar (numeros);
}
```

La solución definitiva se deja como ejercicio al alumno.

Ejercicio Resuelto para hacer 2021-2022:

A partir del programa anterior y modificando sólo un bloque de caracteres contiguos del programa, obtener un vector que tenga los valores del 10 al 1 (es decir, no modifiques dos líneas del programa, ni dos partes diferentes de una línea, sólo un trozo de una línea

Aproximación que no hace caso del enunciado (añade varias líneas) pero funciona

```
#include <stdio.h>

int main (int argc, char *argv[] ) {

/* java
   int numeros[];
   numeros = new int[10];
*/
    int numeros[10];

/* ¿qué pasa en java?
   numeros[10] = 232;
*/
    int indice;

    int decremento=10;

    for ( indice=0; indice < 10; indice++) {
      numeros[indice] = decremento;
      decremento = decremento -1 ;
    }

    printf("Voy a mostrar el vector por pantalla\n");
    for ( indice=0; indice < 10; indice++)
      printf("%d\n", numeros[indice]);
}
```

Solución

```
#include <stdio.h>

int main (int argc, char *argv[] ) {
```

```
    int vicent[10];
    int indice;

    for ( indice=0; indice < 10; indice++) {
       numeros[indice] = 10 - indice;

    }

    printf("Voy a mostrar el vector por pantalla\n");
    for ( indice=0; indice < 10; indice++)
       printf("%d\n", numeros[indice]);
}
```

Ampliar el ejercicio anterior para que en una segunda pasada (no hay que cambiar lo que ya hace el programa), se ponga un cero en los elementos del vector cuya posición sea par

Hay que añadir **después** de lo que tenemos ahora más acciones:

1. Modificar el vector
2. Volverlo a mostrar

Se muestra sólo la ampliación, no el programa entero, ya que sólo hay que añadir a continuación

Vamos a volver a recorrer todo el vector, elemento por elemento y la acción clave es la siguiente:

```
numeros[indice] = 0;
```

Sin embargo, la anterior instrucción no hay que hacerla en todos los elementos. Sólo en algunos que cumplan una condición.

```
if (indice%2 != 0) numeros[indice] = 0;
```

```
printf("\n ahora voy a quitar las posciones pares\n");
```

```
    for ( indice=0; indice < 10; indice++) {
       if (indice%2 != 0) vicent[indice] = 0;

    }

    printf("Voy a mostrar el vector por pantalla\n");
    for ( indice=0; indice < 10; indice++)
       printf("%d\n", vicent[indice]);

}
```

importante hay que copiar y pegar el bloque de código qu emuestra el vector la segunda vez.

Ejercicio para hacer 2021-2022: