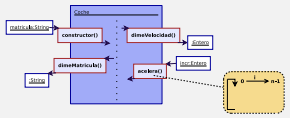
Grado Tecnologías Interactivas

Programación 1



Práctica 3

UNIVERSIDAD POLITECNICA DE VALENCIA

Escola Politècnica Superior de Gandia

DSIC

Departament de Sistemes Informàtics i Computació

Objetivos

- General: diseño de funciones que reciban o devuelvan listas.
- General: Implementación de algoritmos sencillos sobre listas.
- C/C++: Uso de arrays de tipos básicos desde funciones (necesariamente mediante punteros).

¡ Atención !

▷ Se recuerda que las prácticas deben prepararse antes de acudir al aula informática, anotando en el enunciado las dudas que se tengan.

▷ Los diseños y algortimos que se piden en esta práctica deben escribirse en la libreta de apuntes para poder ser revisados.

▷ La realización de las prácticas es un trabajo individual y original. En caso de plagio se excluirá al alumno de la asignatura. Por tanto, es preferible presentar el trabajo realizado por uno mismo aunque éste tenga errores.

1

Introducción a arrays básicos de C

Un array es una estructura que permite guardar valores del mismo tipo, cada uno en una casilla. Las casillas están numeradas consecutivamente, comenzando por 0, lo cuál permite acceder separadamente al contenido de la cada una para consulta o modificación.

El array es el tipo más simple en C que permite representar el concepto abstracto de "lista".

1. Declaración de un array de 10 números reales.

```
double numeros[10];
```

2. Asignación de un valor a una casilla.

```
numeros[5] = 7.2;
```

3. Asignación del valor 5.5 a todas las casillas (el array tiene 10 casillas, de 0 a 9, ambas inclusive):

```
for( int i=0; i<=9; i++ ) {
    numeros[ i ] = 5.5;
}
```

Como vemos, para utilizar arrays básicos es necesario conocer su nombre, el tipo de valor que guarda, y cuántas casillas hemos utilizado del array (porque no siempre utilizaremos todas las casillas disponibles que tenga éste).

Uso de arrays desde funciones

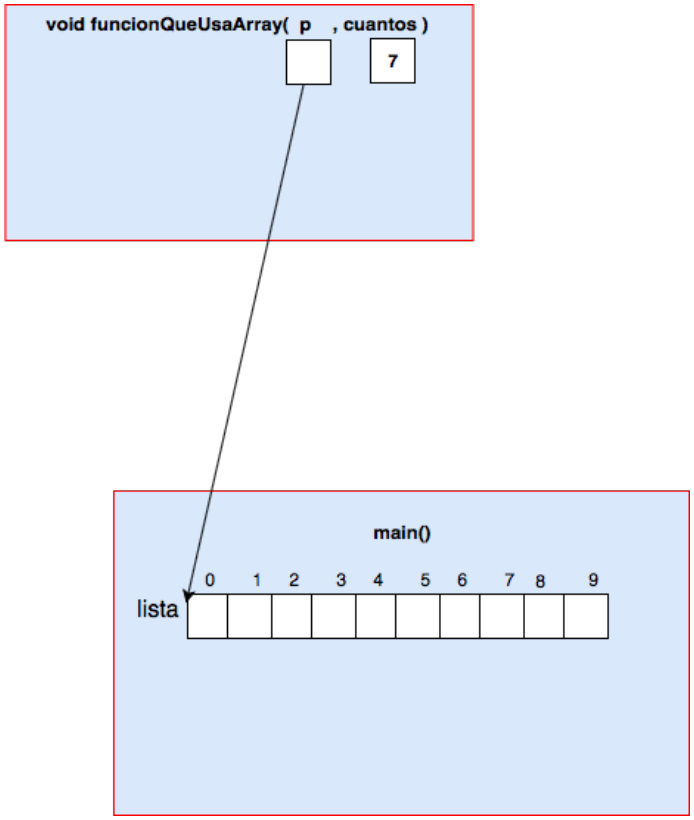
En C, los arrays básicos no se pueden copiar desde una función a otra para que ésta última trabaje con ellos. Necesariamente hay que pasar un puntero. Además, la función necesita saber cuántas casillas del array se han utilizado (porque es habitual no utilizarlas todas y porque no todos los arrays tienen porqué tener la misma cantidad de casillas).

Por ello, el esquema de llamada habitual para que una función pueda manipular un array mediante puntero es el siguiente:

```
// ------------------------
// ------------------------
void funcionQueUsaArray( double * p, unsigned int cuantos ) {

    // usar
 p[ i ]
    // estando i entre  0 y cuantos-1

} // ()

// ------------------------
// ------------------------
int main( ) {
  double lista[10];

  mostrarArray( & lista[0], 7 );
  // supuestamente, hemos usado 7 casillas de las 10
} // ()
```

Puede verse que en el anterior código se cumplen los requisitos para la utilización correcta de un puntero: declaración del puntero (`p`), existencia una variable a la que apuntar (`lista`), e inicialización del puntero para conseguir que apunte a la variable (`p = & lista[0]`).

Esta situación puede representarse gráficamente de esta forma:



Diseño de funciones y Arrays

Desde el punto de vista del diseño de la arquitectura de un programa y de sus algoritmos, preferimos hablar de lista indexada de elementos porque éste es un concepto general único.

Este concepto puede materializarse en un lenguaje de programación de varias formas. Por ejemplo, en C/C++ encontramos arrays (que estamos estudiando), pero también hay la clase vector u otras similares de bibllioteca.

En contreto, una lista indexada cuyos elementos sean del tipo T la representaremos formalmente como

$$Lista < T >$$

Por ejemplo, una lista de números reales es:

$$Lista < \mathbb{R} >$$

En C/C++, el uso de punteros (por ejemplo para manipular arrays desde una función) plantea muchas dudas en lo referente a los diseños de funciones.

En concreto, un puntero es simplemente un mecanismo y, dependiendo de cómo se use, una función podrá recibir información o devolverla o ambas cosas. Por ello:

▷ Atención: los punteros nunca aparecen en los diseños.

Para ilustrar esta importante idea veamos el código de estas dos funciones.

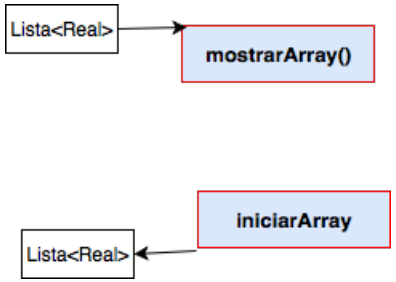
```
void mostrarArray( const double * p, const unsigned int cuantos ) {

  for( int i=0; i<= cuantos-1; i++ ) {
   cout << p[ i ] << " ";
  } // for

  cout << "\n";
} // ()
```

```
void iniciarArray( double * p, const unsigned int cuantos ) {

 for( int i=0; i<= cuantos-1; i++ ) {
   p[i] = i*10;
 } // for

} // ()
```

Y veamos cómo se llama a estas funciones.

```
int main( ) {

  double lista[10];

  iniciarArray( & lista[0], 3 );

  mostrarArray( & lista[0], 3 );

} // ()
```

El perfil de ambas funciones es prácticamente el mismo y las llamadas son iguales. Sin embargo, hacen cosas muy diferentes y eso se refleja necesariamente en sus diseños conceptuales:



2

Ejercicios

Atención: los programas deben estar correctamente comentados. Antes de cada función debe haber una comentario que resuma el diseño de la misma. En código del cuerpo de la función hay que intercalar los pasos del algoritmo.

Es obligatorio utilizar GIT.

1. Diseña[1] una función sumarLista() que reciba una lista de números enteros y devuelva su suma.

   Escribe el algoritmo de la anterior función.

   Implementa la anterior función en un programa llamado SumarLista.cpp.

[1] Escribe la respuesta a continuación.

2. Diseña, escribe su algoritmo e implementa una función llamada dondeEstaElMayor() (nombre del programa: DondeMayor.cpp) que, dada una lista de números reales, devuelva la posición (número de casilla) donde está guardado el mayor valor.

3. Diseña, escribe su algoritmo e implementa una función llamada contiene() (nombre del programa: Contiene.cpp) que, dada una lista de números enteros y un entero n, devuelva verdadero si la lista contiene a n.

4. Diseña, escribe su algoritmo e implementa una función llamada filtrar() (nombre del programa: Filtrar.cpp) que, dada una lista de números reales, devuelva otra lista sólo con los valores positivos de la primera lista.

28 septiembre 2018