Arrays como Parámetros en C

Vicente Benjumea, Manuel Roldán

Resumen

Este documento presenta una comparativa entre los diferentes métodos posibles para definir y utilizar arrays como parámetros y argumentos en el lenguaje de programación C.

En el paso de arrays como argumentos a subprogramas, el **array pierde su tamaño** y automáticamente el parámetro **se convierte en un puntero** al primer elemento del array. Este hecho da lugar a numerosas confusiones y situaciones propensas a errores, especialmente relacionadas con el desbordamiento de la memoria (*buffer overflow*).

A continuación se presentan varios métodos para realizar el paso de arrays como parámetros y argumentos a los subprogramas, y se analizan las ventajas e inconvenientes que puedan presentar.

Paso de Arrays como Parámetros VLA (RECOMENDADA)

VLA es la abreviatura de *"Variable Length Array"*, que en el contexto de paso de parámetros a subprogramas, significa que, además del parámetro array, el subprograma también debe recibir como parámetro la cantidad de elementos del parámetro array, con anterioridad al propio parámetro array, y que además, dicha cantidad de elementos debe ser utilizada en la especificación de la cantidad de elementos de dicho parámetro array.

```
void mostrar_vector(int nelms, const int v[nelms]) ;
void leer_vector(int nelms, int v[nelms]) ;
```

En este método, los subprogramas pueden recibir como argumentos arrays con diferente cantidad de elementos. Además, permite que el compilador pueda realizar varias comprobaciones respecto al tamaño de los arrays pasados como argumentos a los subprogramas. Esto hace que este método sea más robusto que otros en la detección de errores durante la compilación.

Este método es el **recomendado** en nuestro curso, ya que presenta una adecuada flexibilidad de uso, al permitir argumentos arrays con diferente cantidad de elementos, combinada con una adecuada robustez al facilitar la comprobación de errores.

Es importante remarcar que el análisis que realiza el compilador, respecto a la cantidad de elementos del array como parámetro y argumento, no es completo ni exhaustivo, ni detecta todos los posibles casos de error. Concretamente, el compilador no considera error cuando se pasa como argumento un array con mayor cantidad de elementos que la especificada en el parámetro.

```c
enum {
    NELMS_01 = 5,
    NELMS_02 = 9,
} ;
// MÉTODO RECOMENDADO (VLA como parámetro)
void mostrar_vector(int nelms, const int v[nelms])
{
    for (int i = 0; i < nelms; ++i) {
      printf("%d ", v[i]);
    }
    printf("\n");
}
// MÉTODO RECOMENDADO (VLA como parámetro)
void leer_vector(int nelms, int v[nelms])
{
    for (int i = 0; i < nelms; ++i) {
      scanf(" %d", &v[i]);
    }
}
```

```c
void prueba_01()
{
    // Codigo correcto
    int v1[NELMS_01];
    leer_vector(NELMS_01, v1);
    mostrar_vector(NELMS_01, v1);
}
void prueba_02()
{
    // Codigo correcto
    int v2[NELMS_02];
    leer_vector(NELMS_02, v2);
    mostrar_vector(NELMS_02, v2);
}
void prueba_03()
{
    // Codigo erróneo SÍ-DETECTADO por el compilador
    // Detecta el error de pasar como argumento arrays con cantidad
    // de elementos menor que la especificada en el parámetro
    // correspondiente
    int v3[NELMS_01];
    leer_vector(NELMS_02, v3);
      // error: 'leer_vector' accessing 36 bytes in a region of size 20
    mostrar_vector(NELMS_02, v3);
      // error: 'mostrar_vector' reading 36 bytes from a region of size 20
}
void prueba_04()
{
    // El compilador NO considera este caso erróneo
    // El argumento array tiene una cantidad de elementos mayor
    // que la especificada en el parámetro correspondiente
    int v4[NELMS_02];
    leer_vector(NELMS_01, v4);
    mostrar_vector(NELMS_01, v4);
}
```

Paso de Arrays como Parámetros con Cantidad de Elementos Preestablecida Constante

En este caso, la cantidad de elementos del array recibido como parámetro se especifica como una constante definida en una enumeración (*enum*).

```c
enum {
    NELMS_02 = 9,
} ;
void mostrar_vector(const int v[NELMS_02]) ;
void leer_vector(int v[NELMS_02]) ;
```

En este método, los subprogramas **sólo** pueden recibir como argumentos arrays con una cantidad de elementos preestablecida constante. Además, permite que el compilador pueda realizar varias comprobaciones respecto al tamaño de los arrays pasados como argumentos a los subprogramas.

Este método **no** es **recomendado** en nuestro curso, ya que, aunque tiene una adecuada robustez al facilitar la comprobación de errores, presenta una baja flexibilidad de uso, porque solo permite argumentos arrays con la cantidad de elementos preestablecida constante.

Es importante remarcar que el análisis que realiza el compilador, respecto a la cantidad de elementos del array como parámetro y argumento, no es completo ni exhaustivo, ni detecta todos los posibles casos de error. Concretamente, el compilador no considera error cuando se pasa como argumento un array con mayor cantidad de elementos que la preestablecida constante.

```
enum {
    NELMS_01 = 5,
    NELMS_02 = 9,
    NELMS_04 = 15,
} ;
// MÉTODO NO RECOMENDADO
void mostrar_vector(const int v[NELMS_02])
{
    for (int i = 0; i < NELMS_02; ++i) {
      printf("%d ", v[i]);
    }
    printf("\n");
}
// MÉTODO NO RECOMENDADO
void leer_vector(int v[NELMS_02])
{
    for (int i = 0; i < NELMS_02; ++i) {
      scanf(" %d", &v[i]);
    }
}
```

```
void prueba_01()
{
    // Codigo erróneo SÍ-DETECTADO por el compilador
    // Detecta el error de pasar como argumento arrays con cantidad
    // de elementos menor que la preestablecida
    int v1[NELMS_01];
    leer_vector(v1);
      // error: 'leer_vector' accessing 36 bytes in a region of size 20
    mostrar_vector(v1);
      // error: 'mostrar_vector' reading 36 bytes from a region of size 20
}
void prueba_02()
{
    // Codigo correcto
    int v2[NELMS_02];
    leer_vector(v2);
    mostrar_vector(v2);
}
void prueba_03()
{
    // Codigo erróneo SÍ-DETECTADO por el compilador
    // Detecta el error de pasar como argumento arrays con cantidad
    // de elementos menor que la preestablecida
    int v3[NELMS_01];
    leer_vector(v3);
      // error: 'leer_vector' accessing 36 bytes in a region of size 20
    mostrar_vector(v3);
      // error: 'mostrar_vector' reading 36 bytes from a region of size 20
}
void prueba_04()
{
    // El compilador NO considera este caso erróneo
    // El argumento array tiene una cantidad de elementos mayor
    // que la preestablecida
    int v4[NELMS_04];
    leer_vector(v4);
    mostrar_vector(v4);
}
```

Paso de Arrays como Parámetros sin Especificar la Cantidad de Elementos

En este caso, la cantidad de elementos del array recibido como parámetro queda sin especificar, por lo que es necesario que el subprograma reciba la cantidad de elementos del parámetro array como otro parámetro adicional.

```
void mostrar_vector(int nelms, const int v[]) ;
void leer_vector(int nelms, int v[]) ;
```

En este método, los subprogramas pueden recibir como argumentos arrays con diferente cantidad de elementos. Sin embargo, **no permite** que el compilador realice comprobaciones respecto a la cantidad de elementos de los arrays pasados como argumentos a los subprogramas, debido a que la definición de los parámetros no contiene información que permita establecer un vínculo entre el valor recibido como parámetro, y la cantidad de elementos del array recibido. Dicho vínculo está en la mente del programador, pero no aparece en la definición de los parámetros.

Este método **no** es **recomendado** en nuestro curso, ya que, aunque presenta una adecuada flexibilidad de uso, al permitir argumentos arrays con diferente cantidad de elementos, tiene una baja robustez, ya que el compilador **no realiza ninguna comprobación** respecto a la cantidad de elementos de los arrays pasados como argumentos a los subprogramas. Además, las relaciones entre los parámetros no aparecen claramente especificadas en su definición.

```c
enum {
    NELMS_01 = 5,
    NELMS_02 = 9,
} ;
// MÉTODO NO RECOMENDADO
void mostrar_vector(int nelms, const int v[])
{
    for (int i = 0; i < nelms; ++i) {
      printf("%d ", v[i]);
    }
    printf("\n");
}
// MÉTODO NO RECOMENDADO
void leer_vector(int nelms, int v[])
{
    for (int i = 0; i < nelms; ++i) {
      scanf(" %d", &v[i]);
    }
}
```

```c
void prueba_01()
{
    // Codigo correcto
    int v1[NELMS_01];
    leer_vector(NELMS_01, v1);
    mostrar_vector(NELMS_01, v1);
}
void prueba_02()
{
    // Codigo correcto
    int v2[NELMS_02];
    leer_vector(NELMS_02, v2);
    mostrar_vector(NELMS_02, v2);
}
void prueba_03()
{
    // Codigo erróneo NO-DETECTADO por el compilador
    // El argumento array tiene una cantidad  de elementos
    // menor que la especificada en el parámetro correspondiente
    int v3[NELMS_01];
    leer_vector(NELMS_02, v3);
    mostrar_vector(NELMS_02, v3);
}

void prueba_04()
{
    // El compilador NO considera este caso erróneo
    // El argumento array tiene una cantidad de elementos mayor
    // que la especificada en el parámetro correspondiente
    int v4[NELMS_02];
    leer_vector(NELMS_01, v4);
    mostrar_vector(NELMS_01, v4);
}
```

Paso de Arrays como Parámetros de Tipo Puntero

En este caso, el subprograma recibe un puntero al primer elemento del array recibido como parámetro, por lo tanto, la cantidad de elementos del array recibido como parámetro queda sin especificar, por lo que es necesario que el subprograma reciba la cantidad de elementos del array, correspondiente al parámetro puntero, como otro parámetro adicional.

```c
void mostrar_vector(int nelms, const int *v) ;
void leer_vector(int nelms, int *v) ;
```

En este método, los subprogramas pueden recibir como argumentos arrays con diferente cantidad de elementos. Sin embargo, **no permite** que el compilador realice comprobaciones respecto a la cantidad de elementos de los arrays pasados como argumentos a los subprogramas, debido a que la definición de los parámetros no contiene información que permita establecer un vínculo entre el valor recibido como parámetro, y la cantidad de elementos del array correspondiente al puntero recibido. Dicho vínculo está en la mente del programador, pero no aparece en la definición de los parámetros.

Además, la definición de los parámetros tampoco contiene ninguna información que permita diferenciar si el puntero apunta a un único elemento, o al primer elemento de un array.

Este método **no** es **recomendado** en nuestro curso, ya que, aunque presenta una adecuada flexibilidad de uso, al permitir argumentos arrays con diferente cantidad de elementos, tiene una baja robustez, ya que el compilador **no realiza ninguna comprobación** respecto a la cantidad de elementos de los arrays pasados como argumentos a los subprogramas. Además, las relaciones entre los parámetros no aparecen claramente especificadas en su definición.

```c
enum {
    NELMS_01 = 5,
    NELMS_02 = 9,
} ;
// MÉTODO NO RECOMENDADO
void mostrar_vector(int nelms, const int *v)
{
    for (int i = 0; i < nelms; ++i) {
      printf("%d ", v[i]);
    }
    printf("\n");
}
// MÉTODO NO RECOMENDADO
void leer_vector(int nelms, int *v)
{
    for (int i = 0; i < nelms; ++i) {
      scanf(" %d", &v[i]);
    }
}
```

```c
void prueba_01()
{
    // Codigo correcto
    int v1[NELMS_01];
    leer_vector(NELMS_01, v1);
    mostrar_vector(NELMS_01, v1);
}
void prueba_02()
{
    // Codigo correcto
    int v2[NELMS_02];
    leer_vector(NELMS_02, v2);
    mostrar_vector(NELMS_02, v2);
}
void prueba_03()
{
    // Codigo erróneo NO-DETECTADO por el compilador
    // El argumento array tiene una cantidad de elementos
    // menor que la especificada en el parámetro correspondiente
    int v3[NELMS_01];
    leer_vector(NELMS_02, v3);
    mostrar_vector(NELMS_02, v3);
}
void prueba_04()
{
    // El compilador NO considera este caso erróneo
    // El argumento array tiene una cantidad de elementos mayor
    // que la especificada en el parámetro correspondiente
    int v4[NELMS_02];
    leer_vector(NELMS_01, v4);
    mostrar_vector(NELMS_01, v4);
}
```