

PRACTICA INICIALIZACIÓN DE VARIABLES

Rebeca Radío Armindo

November 2023

1 EJERCICIOS

1.1 BLOQUE DLC:REGLAS

Repaso de la REGLA 2: Declarations and Initialization (DCL) en el segun el SEI CERT C Coding Standard. • DCL30-C. Declare objects with appropriate storage durations • DCL31-C. Declare identifiers before using them

- DCL36-C. Do not declare an identifier with conflicting linkage classifications
- DCL37-C. Do not declare or define a reserved identifier
- DCL38-C. Use the correct syntax when declaring a flexible array member
- DCL39-C. Avoid information leakage when passing a structure across a trust boundary
- DCL40-C. Do not create incompatible declarations of the same function or object
- DCL41-C. Do not declare variables inside a switch statement before the first case label

1.1. Ejemplo 1 Revisa y evalua la siguiente traza de código. 1. Define la regla que se incumple y pro´on una alternativa m´as adecuada seg´un el SEI CERT C.

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

const char *p;

char *funcion1(void) {
    char array[10] = "Mi Cadena";
    /* Initialize array */
    return array;
}

void funcion2(void) {
    const char c_str[] = "Todo va bien";
    p = c_str;
}

void funcion3(void) {
    printf("%s\n", p);
}
```

```
}

int main(void) {
    p = funcion1();
    printf("%s\n", p);
    funcion2();
    funcion3();
    printf("%s\n", p);

    return 0;
}
```

DCL30-C: Declare objects with appropriate storage durations

En la función llamada funcion1 se declara un array local y se devuelve un puntero a el. El array local tendrá una duración limitada a la vida util de la funcion y devolver un putero puede ocasionar comportamientos extraños despues de que la función haya terminado.

Es una regla sobre el tiempo que dura el almacenamiento disponible.

1.2. Ejemplo 2 1. ¿Qué hace el siguiente segmento de código?

Este trozo de código contiene una función llamada `func` que tiene como argumento el tamaño de un array y realiza diferentes operaciones:

Primero declara una estructura llamada `flexArrayStruct`, que contiene un número de tipo entero `num` y un array de enteros de tamaño 1.

Dentro de esta función también se utiliza una llamada a `malloc` para asignar dinámicamente para una instancia de `flexArrayStruct`. La memoria asignada se calcula para almacenar la estructura de enteros del array y la dirección de memoria asignada se almacena con un puntero `structP`.

Se verifica si la asignación de memoria ocurrió correctamente comprobando si el `malloc` devuelve `null`. Si falla se maneja el error ((comentado pero no especificado en el código)).

Luego se asigna el valor del tamaño del array al valor `num` apuntado por `structP` y se hace un `for` para inicializar los elementos del array.

En resumen, `func` asigna memoria dinámica a un array que varía su tamaño, inicializa sus componentes y luego inicia los elementos del array a 1.

2. De haber algún problema ¿Podrías decir la línea en la que se encuentra?

En mi opinión en la línea donde se verifica si la asignación de memoria funciona, se verifica que `structP` es `null` lo cual significaría que la asignación de memoria ha fallado pero no hay código dentro de este condicional y por lo tanto si la asignación falla, no ocurrirá nada diferente ya que no hay indicado un tratamiento de error en caso de que falle.

3. Define la regla que se incumple y propon una alternativa correcta siguiendo el SEI CERT C.

```
#include <stdlib.h>

struct flexArrayStruct {
    int num;
    int data[1];
};

void func(size_t array_size) {
    /* Space is allocated for the struct */
    struct flexArrayStruct *structP
        = (struct flexArrayStruct *)
            malloc(sizeof(struct flexArrayStruct)
                + sizeof(int) * (array_size - 1));
    if (structP == NULL) {
        /* Handle malloc failure */
    }

    structP->num = array_size;

    /*
     * Access data[] as if it had been allocated
     * as data[array_size].
     */
    for (size_t i = 0; i < array_size; ++i) {
        structP->data[i] = 1;
    }
}
```

La regla que se incumple en el código proporcionado es la regla DCL30-C del SEI CERT C Coding Standard: "Declare objects with appropriate storage durations."

El problema específico está en la función `func` en las líneas 10-13, donde se utiliza `malloc` para asignar memoria para la estructura `flexArrayStruct` y el array `data`. La memoria asignada con `malloc` tiene una duración de almacenamiento (lifetime) que persiste hasta que se libera explícitamente mediante `free`. Sin embargo, en el código proporcionado, no hay una llamada a `free` para liberar la memoria asignada, lo que podría resultar en una fuga de memoria.

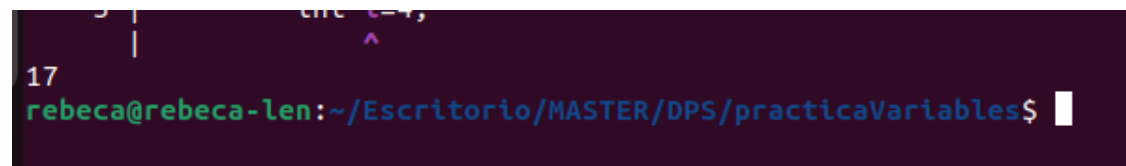
1.3. Ejemplo 2 1. ¿Qué hace el siguiente segmento de código si invocamos la función `funcion` un 0?

Entonces, si invocas la función `func` con un argumento de 0, imprimirá el valor 17 en la salida estándar ya que asigna el valor 17 a la variable `i` en `func` y se imprime el valor de `i` actual ya que no hay `break` después de `case 0`. Por último termina la función.

2. De haber algún problema ¿Podrías decir la línea en la que se encuentra?

En mi opinión se debe poner el `break` después de `case 0` y en la línea `int i=4` se puede producir un problema. La declaración de la variable está dentro del `switch` y se debe poner al principio de la función.

3. Crea un fichero con un `main` y ejecuta el segmento de código.



```
17
rebeca@rebeca-len:~/Escritorio/MASTER/DPS/practicaVariables$
```

4. Propón una solución al ejemplo que cumpla con las normas del CMU

La declaración de la variable está dentro del `switch` y se debe poner al principio de la función ya que sino se incumple la norma del estándar C11

5. Realiza un análisis estático del código erróneo y copia en tu solución el resultado.

Utiliza las herramientas: (a) `rats` (b) `cppchecker` (c) `splint` (d) `vera++` (e) `valgrind`

```
#include <stdio.h>

extern void f(int i);

void func(int expr) {
    switch (expr) {
        int i = 4;
        f(i);
        case 0:
            i = 17;
        default:
            printf("%d\n", i);
    }
}
```

1.2 BLOQUE DLC: Recomendaciones

Repaso de las Recomendaciones 2: Declarations and Initialization (DCL) en el segundo el SEI CERT C Coding Standard

- DCL00-C. Const-qualify immutable objects
- DCL01-C. Do not reuse variable names in subscopes

- DCL02-C. Use visually distinct identifiers
- DCL03-C. Use a static assertion to test the value of a constant expression
- DCL04-C. Do not declare more than one variable per declaration
- DCL05-C. Use typedefs of non-pointer types only
- DCL06-C. Use meaningful symbolic constants to represent literal values
- DCL07-C. Include the appropriate type information in function declarators
- DCL08-C. Properly encode relationships in constant definitions
- DCL09-C. Declare functions that return errno with a return type of errno_t
- DCL10-C. Maintain the contract between the writer and caller of variadic functions
- DCL11-C. Understand the type issues associated with variadic functions
- DCL12-C. Implement abstract data types using opaque types
- DCL13-C. Declare function parameters that are pointers to values not changed by the function as const
- DCL15-C. Declare file-scope objects or functions that do not need external linkage as static
- DCL16-C. Use "L," not "l," to indicate a long value
- DCL17-C. Beware of miscompiled volatile-qualified variables
- DCL18-C. Do not begin integer constants with 0 when specifying a decimal value
- DCL19-C. Minimize the scope of variables and functions
- DCL20-C. Explicitly specify void when a function accepts no arguments
- DCL21-C. Understand the storage of compound literals
- DCL22-C. Use volatile for data that cannot be cached
- DCL23-C. Guarantee that mutually visible identifiers are unique

```
-Werror -Wall -Wextra -pedantic -Wcast-align -Wcast-qual -Wctor-  
dtor-privacy -Wdisabled-optimization -Wformat=2 -Winit-self -Wlogical-  
op -Wmissing-include-dirs -Wnoexcept -Wold-style-cast -Woverloaded-  
virtual -Wredundant-decls -Wshadow -Wsign-promo -Wstrict-null-sentinel  
-Wstrict-overflow=5 -Wundef -Wno-unused -Wno-variadic-macros -Wno-  
parentheses -fdiagnostics-show-option
```

2.1. Ejercicio 1 • ¿Qué hace el siguiente segmento de código? Este código implementa una función llamada average que calcula el promedio de una lista variable de argumentos.

¿Para que se utiliza la variable va_eol?

El propósito es servir como un marcador o en la lista de argumentos de la función average. En este caso, -1 se ha asignado al enumerador va_eol, y se utiliza para indicar el final de la lista de argumentos.

2.2. Ejercicio 2 • Comenta qué reglas/recomendaciones se están rompiendo aquí. También entran reglas pasadas.

DCL01-C. Do not reuse variable names in subscopes

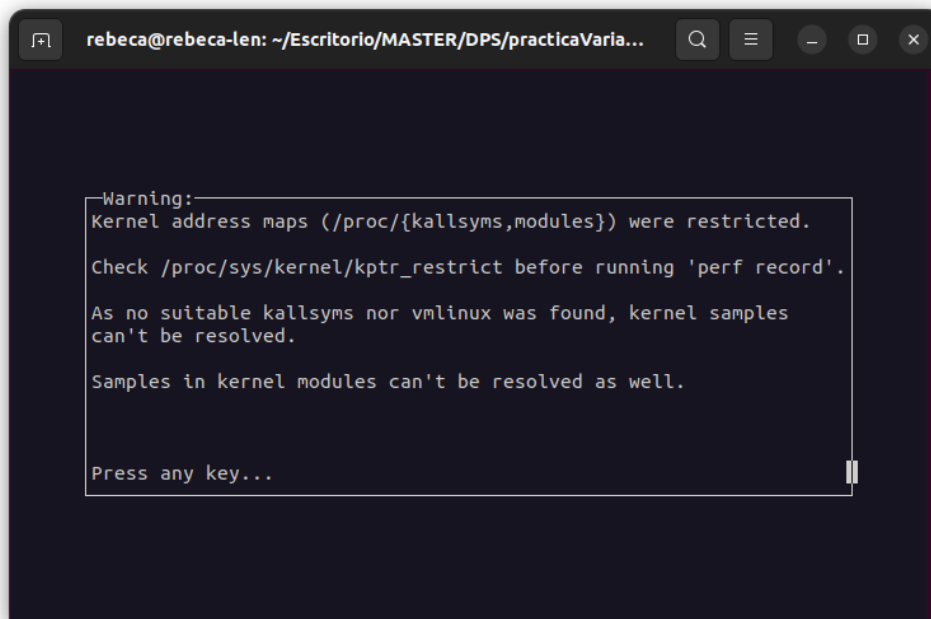
DCL04-C. Do not declare more than one variable per declaration

DCL13-C. Declare function parameters that are pointers to values not changed by the function as const

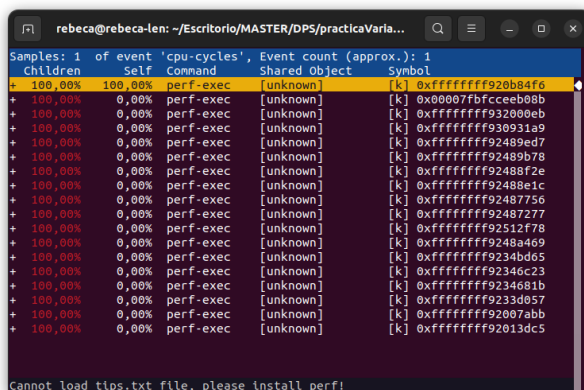
DCL16-C. Use "L," not "l," to indicate a long value

DCL19-C. Minimize the scope of variables and functions

- Instala la herramienta perf para realizar el profiling de la aplicación. Se puede instalar con apt.
- El programa permite mostrar el código desensamblado de la aplicación, adjunta alguna captura.



```
rebeca@rebeca-len: ~/Escritorio/MASTER/DPS/practicaVaria...  
Warning:  
Kernel address maps (/proc/{kallsyms,modules}) were restricted.  
Check /proc/sys/kernel/kptr_restrict before running 'perf record'.  
  
As no suitable kallsyms nor vmlinux was found, kernel samples  
can't be resolved.  
  
Samples in kernel modules can't be resolved as well.  
  
Press any key...
```



```
rebeca@rebeca-len: ~/Escritorio/MASTER/DPS/practicaVaria...  
Samples: 1 of event 'cpu-cycles', Event count (approx.): 1  
Children  Self Command Shared Object Symbol  
+ 100,00% 100,00% perf-exec [unknown] [k] 0xffffffff920b84f6  
+ 100,00% 0,00% perf-exec [unknown] [k] 0x00007fbfcceb88b  
+ 100,00% 0,00% perf-exec [unknown] [k] 0xffffffff932000eb  
+ 100,00% 0,00% perf-exec [unknown] [k] 0xffffffff930931a9  
+ 100,00% 0,00% perf-exec [unknown] [k] 0xffffffff92489ed7  
+ 100,00% 0,00% perf-exec [unknown] [k] 0xffffffff92489b78  
+ 100,00% 0,00% perf-exec [unknown] [k] 0xffffffff92488f2e  
+ 100,00% 0,00% perf-exec [unknown] [k] 0xffffffff92488e1c  
+ 100,00% 0,00% perf-exec [unknown] [k] 0xffffffff92487756  
+ 100,00% 0,00% perf-exec [unknown] [k] 0xffffffff92487277  
+ 100,00% 0,00% perf-exec [unknown] [k] 0xffffffff92512f78  
+ 100,00% 0,00% perf-exec [unknown] [k] 0xffffffff9248a469  
+ 100,00% 0,00% perf-exec [unknown] [k] 0xffffffff9234bd05  
+ 100,00% 0,00% perf-exec [unknown] [k] 0xffffffff92346c23  
+ 100,00% 0,00% perf-exec [unknown] [k] 0xffffffff9234681b  
+ 100,00% 0,00% perf-exec [unknown] [k] 0xffffffff9233d057  
+ 100,00% 0,00% perf-exec [unknown] [k] 0xffffffff92087abb  
+ 100,00% 0,00% perf-exec [unknown] [k] 0xffffffff92013dc5  
Cannot load tips.txt file, please install perf!
```

- ¿Podrías decir cual es la instrucción que mas tiempo de CPU requiere? Adjunta una captura y describe la razón.

```

#include <stdio.h>

unsigned long long int factorial(unsigned int i) {

    if(i <= 1) {
        return 1;
    }
    return i * factorial(i - 1);
}

int main(int argc, char *argv[]) {
    int i = 12, j=3, f=0;

    if (argc==1){

        printf("Factorial of %d is %lld\n", i, factorial(i)
        );
    }
    else{
        j=atoi(argv[1]);
        for (f=0;f<j;f++){
            printf("Factorial of %d is %lld\n", f,
            factorial(f));
        }
    }

    return 0;
}

```

¿Podrias decir cual es la instrucción que más tiempo de CPU requiere la siguiente traza de código? Adjunta una captura y describe la razón

```

// fib.c
#include <stdio.h>
#include <stdlib.h>

int fib(int x) {
    if (x == 0) return 0;
    else if (x == 1) return 1;
    return fib(x - 1) + fib(x - 2);
}

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

    for (size_t i = 0; i < 45; ++i) {
        printf("%d\n", fib(i));
    }
    return 0;
}

```