Evaluación de Código C

Francisco J. Rodríguez Lera

[1]MUIC - Master Universitario en Investigación en Ciberseguridad

1. Repositorio

https://classroom.github.com/a/2TxvKdKR

2. BLOQUE DLC: C

Repaso de las reglas asociadas al tema Declarations and Initialization (DCL) del 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

2.1. Ejemplo 1

Revisa y evalua la siguiente traza de código.

1. Define la regla que se incumple y proón una alternativa más adecuada según el SEI CERT C.

```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;
}
```

```
16
17 void funcion3(void) {
18   printf("%s\n", p);
19 }
20
21 int main(void) {
22
23   p = funcion1();
24   printf("%s\n",p)
25   funcion2();
26   funcion3();
27   printf("%s\n", p);
28
29   return 0;
30 }
```

2.2. Ejemplo 2

1. ¿Qué hace el siguiente segmento de código?
2. De haber algún problema ¿Podrías decir la línea en la que se encuentra?
3. Define la regla que se incumple y proón una alternativa correcta siguiendo el SEI CERT C.

.

```
1  #include <stdlib.h>
2
3  struct flexArrayStruct {
4    int num;
5    int data[1];
6  };
7
8  void func(size_t array_size) {
9    /* Space is allocated for the struct */
10   struct flexArrayStruct *structP
11     = (struct flexArrayStruct *)
12       malloc(sizeof(struct flexArrayStruct)
13            + sizeof(int) * (array_size - 1));
14   if (structP == NULL) {
15     /* Handle malloc failure */
16   }
17
18   structP->num = array_size;
19
20   /*
21    * Access data[] as if it had been allocated
22    * as data[array_size].
23    */
```

```
24    for (size_t i = 0; i < array_size; ++i) {
25      structP->data[i] = 1;
26    }
27  }
```

2.3. Ejemplo 2

1. ¿Qué hace el siguiente segmento de código si invocamos la función **func** con un **0**?
2. De haber algún problema ¿Podrías decir la línea en la que se encuentra?
3. Crea un fichero con un main y ejecuta el segmento de código.
4. Propón una solución al ejemplo que cumpla con las normal del CMU
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

```
1  #include <stdio.h>
2
3  extern void f(int i);
4
5  void func(int expr) {
6    switch (expr) {
7      int i = 4;
8      f(i);
9    case 0:
10     i = 17;
11   default:
12     printf("%d\n", i);
13   }
14 }
```

3. BLOQUE DLC: Java

Repaso de las reglas asociadas al tema Declarations and Initialization (DCL) del SEI CERT Java Coding Standard.

- DCL00-J. Prevent class initialization cycles
- DCL01-J. Do not reuse public identifiers from the Java Standard Library
- DCL02-J. Do not modify the collection's elements during an enhanced for statement

3.1. Ejemplo

1. ¿Qué hace el siguiente segmento de código?
2. De haber algún problema ¿Podrías decir la línea en la que se encuentra?
3. Crea un fichero .java con un main, adapta y ejecuta el segmento de código.
4. Propón una solución al ejemplo que cumpla con las normal del CMU
5. Indica la versión de Java que estás utilizando y cómo has hecho para compilar y ejecutar.

```
1  List<Integer> list = Arrays.asList(new Integer[] {13, 14,
       15});
2  boolean first = true;
3
4  System.out.println("Processing list...");
5  for (Integer i: list) {
6    if (first) {
7      first = false;
8      i = new Integer(99);
9    }
10   System.out.println(" New item: " + i);
11   // Process i
12 }
13
14 System.out.println("Modified list?");
15 for (Integer i: list) {
16   System.out.println("List item: " + i);
17 }
```

4. BLOQUE Inicialización y Anotación en Python

1. ¿Qué hace el siguiente segmento de código?

```
1            def greeting(name: str) -> str:
2            return 'Hello ' + name
```

2. ¿Cual es la diferencia entre estas dos trazas de código

```
1  # ========  Traza 1 ============
2  from typing import TypeVar, Iterable, Tuple
3
4  T = TypeVar('T', int, float, complex)
5  Vector = Iterable[Tuple[T, T]]
6
7  def inproduct(v: Vector[T]) -> T:
8      return sum(x*y for x, y in v)
9  def dilate(v: Vector[T], scale: T) -> Vector[T]:
10     return ((x * scale, y * scale) for x, y in v)
11 vec = []
12
13 # ========  Traza 2 ============
14
15 from typing import TypeVar, Iterable, Tuple
16
17 T = TypeVar('T', int, float, complex)
18
19 def inproduct(v: Iterable[Tuple[T, T]]) -> T:
20     return sum(x*y for x, y in v)
21 def dilate(v: Iterable[Tuple[T, T]], scale: T) ->
     Iterable[Tuple[T, T]]:
22     return ((x * scale, y * scale) for x, y in v)
23 vec = []
```

3. ¿Qué hace el siguiente segmento de código? ¿Se debería declarar de alguna otra forma?

```
1          f = lambda x: 2*x
```

4. Qué son los Python Enhancement Proposal y para que se utilizan. Si lo consideras necesario une los ejemplos anteriores a los Python Enhancement Proposal adecuados.

5. BLOQUE DLC: Recomendaciones

Repaso de las Recomendaciones 2: Declarations and Initialization (DCL) en el segun 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

5.1. Ejercicio Extra

- ¿Qué hace el siguiente segmento de código?
- Comenta qué reglas/recomendaciones se están rompiendo aquí. Tambien entran reglas pasadas.
- 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.
- ¿Podrías decir cual es la instrucción que más tiempo de CPU requiere? Adjunta uns captura y describe la razón.

```
1 #include <stdio.h>
2
3 unsigned long long int factorial(unsigned int i) {
4
```

```
5     if(i <= 1) {
6        return 1;
7     }
8     return i * factorial(i - 1);
9  }
10
11 int  main(int argc, char *argv[]) {
12     int i = 12, j=3, f=0;
13
14     if (argc==1){
15          printf("Factorial of %d is %lld\n", i, factorial(i)
               );
16     }
17     else{
18          j=atoi(argv[1]);
19          for (f=0;f<j;f++){
20                   printf("Factorial of %d is %lld\n", f,
                        factorial(f));
21          }
22     }
23
24    return 0;
25 }
```