Resultados de la evaluación. Errores más comunes

1. Letra ilegible
   - Identificación en formato diferente al indicado (Apellidos, Nombre)
2. Estructura de código deficiente
   - Variables innecesarias y código repetido o inútil
   - Declaración de variables al vuelo, incluso dentro de bucles
   - Bucles infinitos o que no ejecutan nunca
3. Sintaxis ***include*** **en C**: ver https://gcc.gnu.org/onlinedocs/cpp/Include-Syntax.html
4. Utilizar llamadas al sistema diferentes a las indicadas al final del enunciado (especificación)
   - Por ejemplo: **exit()**, ***strdup()***, ***fopen()***, ***fgets()***, ***strcmp()***, ***strncmp()*** no se admiten
5. Utilizar incorrectamente llamadas al sistema cuya sintaxis se incluye al final del enunciado
6. Control del número de argumentos en línea de orden incorrecto
7. La línea de orden es una cosa y *stdin* (o |) otra muy diferente
8. Confusión uso de ***sizeof()*** y ***strlen()***. Ver código ejemplo adjunto
9. Declaración de **array estático** para acomodar el path y añadir el carácter '/'. El enunciado especifica claramente que el programa solicitará memoria dinámica y en la cantidad estrictamente necesaria
   - Se requieren **exactamente strlen(path)+2** bytes (caracteres '/' y '\0')
10. Confusión y/o maluso de las declaraciones *char * vs char[] vs char * []*
11. Confusión "/" y '/'. Lo primero es un *string* (vector terminado en 0), lo segundo un *char*
12. En C, de momento, el **operador +** solo sirve para sumar, no para concatenar *strings*
13. En C = es una cosa y == otra (se suponía básico y superado)
14. Lo mismo (se suponía básico y superado) hacer *include* de fuente .c en lugar de cabecera .h
15. Confusión 0 vs '\0' vs "" (recurrente)
16. Confusión llamada al sistema *strcpy()* y asignación de punteros
    - *strcpy(str1, str2)* es diferente a *str1 = str2*
17. Si str1 es *NULL* o no hay memoria asignada *strcpy(str1, str2)* provoca "problemas"
18. Invocar a *pps_scandir()* sin memoria asignada correctamente para sus dos argumentos
19. Invocar a *a_lsdir()* sin atender a la precondición i1ϵ[1, n); i2ϵ(1, n]; i1<i2; n nº entradas leídas
20. Código en *a_lsdir()* que comprueba o testea algún aspecto de la precondición
21. Tamaño insuficiente del vector de permisos
22. Indexar y acceder incorrectamente el array de elementos del **tipo *scandir_t*** (*struct scandir*)
    - Es análogo al segundo ejercicio práctico
    - En *scptr→aadr* se tiene la dirección de un array de *scptr→nent* elementos *scandir_t*. Luego, para acceder al i-ésimo elemento del array la expresión es *scptr→aadr[i]*
    - *scptr→aadr[i]* es un *struct*, no un puntero. Para acceder a sus campos se usa '.' y no '→'
23. *DT_DIR*, al igual que *MAX_PILA*, es una constante simbólica. No es ni un string ni un carácter

Además de los descritos en la lista anterior, en la solución comentada que se muestra a continuación se muestran más errores y su correspondiente explicación/justificación
Al final del documento se incluye el código fuente de un ejemplo que ahonda en la explicación de la solución del examen. Se recomienda compilar y ejecutar el ejemplo planteando las dudas que proceda enviando previo a la revisión un correo a la dirección pps@fi.upm.es con el asunto "Dudas sizelen.c"

Importante:

En la revisión no se atenderán dudas teóricas ni prácticas relativas a la presente solución. Como se ha señalado, se deberán remitir al correo anterior

Hoja #1: función main() [6 ptos]

```c
/*
 * Includes sistema/usuario
 * Ver sintaxis en https://gcc.gnu.org/onlinedocs/cpp/Include-Syntax.html
 */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "scandir.h"

/*
 * Programa principal
 */
int main(int argc, char * argv[])
{
	scanarr_t scanarr = {NULL, 0};
	char *dpath = NULL;
	int rcod;

	if (argc == 2) {
		rcod = strlen(argv[1]);
		if (argv[1][rcod-1] != '/') {
			dpath = (char *) calloc(rcod+2, sizeof(unsigned char));
			if (dpath != NULL) {
				strcpy(dpath, argv[1]);
				dpath[rcod] = '/';
			}
			else {
				printf("Fallo de memoria. Fin del programa\n");
				return -1;
			}
		}
		else
			dpath = argv[1];
	}
	else {
		printf("Parámetros incorrectos. No hace nada\n");
		return -1;
	}
	rcod = pps_scandir(&scanarr, dpath);
	if (dpath != argv[1])
		free(dpath);
	if(rcod>0) {
		printf("a_lsdir:\n");
		printf("=======\n");
		a_lsdir(&scanarr, 1, scanarr.nent);
		free(scanarr.aadr);
	}
	return rcod;
}
```

Declaración de variables
La declaración de scanarr es estática. No hace falta hacerla dinámica
No vale declarar scanarr_t *scanarr; (un puntero) ya que entonces pps_scandir() no dispone de memoria y causa *core*

Solo un (**1**) argumento

Nunca ***sizeof(argv[1])*** (es un **puntero**). Ver ejemplo ***sizelen.c***

strcpy(), nunca asignación *dpath = argv[1]*
Provoca que el *calloc()* sea inútil ya que la memoria obtenida queda inaccesible (dereferenciada)

A diferencia de lo anterior, esto no supone ningún problema. Son dos punteros que señalan a la misma memoria
Se hace para tener un único punto de invocación de *pps_scandir()* con argumento *dpath*

Liberar sólo si procede (si hubo asignación dinámica)

Liberar únicamente lo que procede. En este caso, ***scanarr*** es una estructura declarada de forma estática. Por tanto, sólo hay que liberar la memoria referenciada desde ***aadr*** (*array address*)

Hoja #2: a_lsdir() [4 ptos]

```c
/*
 * Funcion que muestra el nombre del fichero, el tamaño en bytes y los
 * tres grupos de permisos de las entradas en el rango (i1,i2) de filas
 * del array (tipo scanarr_t)
 * Precondición: i1 ∈ [1, n);  i2 ∈ (1, n]; i1<i2; n, nº entradas leídas
 * Un ejemplo de salida (2 líneas) es el siguiente:
 * test           (type DT_DIR)     Permisos: rwxrwxr-x
 * Ficheros.pdf   (55347 bytes)     Permisos: rw-rw-r--
 * El string "rw-rw-r--" se obtiene invocando smode
 */
void a_lsdir (scanarr_t *scptr, int i1, int i2)
{
      char perms[10];
      int idx;

      memset(perms,0,10);

      for (idx = i1 - 1; idx < i2; idx++) {

         if(scptr->aadr[idx].entry.d_type == DT_DIR)

               printf("%-10s\t(type DT_DIR)\tPermisos: ",
                           scptr->aadr[idx].entry.d_name);
         else
               printf("%-10s\t(%.5ld bytes)\tPermisos: ",
                           scptr->aadr[idx].entry.d_name,
                           scptr->aadr[idx].info.st_size);
         smode(perms, scptr->aadr[idx].info);
         printf("%s\n", perms);
      }
}
```

Vector estático para acomodar los permisos. Se declara fuera del bucle
Tamaño: 3 caracteres (de los 3 grupos de permisos) más el fin de cadena

Inicializar siempre. Nunca se sabe...

Para indexar se usa una variable local, no el argumento de entrada
Los argumentos toman valores en los intervalos que determina la precondición. Esta marca que el valor mínimo del índice inferior i1 es 1 y el valor máximo del superior i2 es el tamaño del array. Pero los arrays en C indexan desde 0. Por eso el índice idx se inicializa a i1-1 y la condición de salida es idx<i2

Constante simbólica. No sirve "DT_DIR"

En *scptr→aadr* se tiene la dirección de un array de *scptr→nent* elementos del **tipo *scandir_t***
Luego, el i-ésimo elemento del array es *scptr→aadr[i]*
Además, *scptr→aadr[i]* es un struct, no un puntero. Para acceder a sus campos se usa '.' y no '→'

Ejemplo sizelen.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {

        char *str1 = "szEj1";
        char str2[] = "SizeLenEj2";
        char *sizelen = NULL;

        printf("sizeof(sizelen): %02d bytes\n", (int) sizeof(sizelen));
        printf("strlen(sizelen): Lindo core\n");
        printf("str1; strlen: %02d bytes, sizeof: %02d bytes\n", (int) strlen(str1), (int) sizeof(str1));
        printf("str2; strlen: %02d bytes, sizeof: %02d bytes\n", (int) strlen(str2), (int) sizeof(str2));

        printf("-----\n");

        sizelen = (char *) calloc(strlen(str1)+2, sizeof(unsigned char));
        if (sizelen == NULL)
                return -1;
        printf("sizelen; strlen: %02d bytes, sizeof: %02d (puntero) y no %02d\n",
                        (int) strlen(sizelen), (int) sizeof(sizelen), (int) strlen(str1)+2);

        printf("-----\n");

        strcpy(sizelen, str1);
        printf("sizelen; strlen: %02d bytes, sizeof: %02d bytes\n", (int) strlen(sizelen), (int) sizeof(sizelen));
        printf("OJO: En sizelen hay %02d bytes disponibles y a 0\n", (int) (strlen(str1)+2-strlen(sizelen)));

        printf("-----\n");

        printf("sizelen; dirección antes de asignar: %p, ", sizelen);
        sizelen = str1;
        printf("después de asignar: %p\n", sizelen);
        printf("sizelen; strlen: %02d bytes, sizeof: %02d bytes\n", (int) strlen(sizelen), (int) sizeof(sizelen));
        printf("OJO: Hay %02d bytes perdidos (mem leak) en la memoria profunda\n", (int) strlen(str1)+2);

        return 0;
}
```

Ejecución

```
sizeof(sizelen): 08 bytes
strlen(sizelen): Lindo core
str1; strlen: 05 bytes, sizeof: 08 bytes
str2; strlen: 10 bytes, sizeof: 11 bytes
-----
sizelen; strlen: 00 bytes, sizeof: 08 (puntero) y no 07
-----
sizelen; strlen: 05 bytes, sizeof: 08 bytes
OJO: En sizelen hay 02 bytes disponibles y a 0
-----
sizelen; dirección antes de asignar: 0x5569205c8670, después de asignar: 0x55691ea74aa8
sizelen; strlen: 05 bytes, sizeof: 08 bytes
OJO: Hay 07 bytes perdidos (mem leak) en la memoria profunda
```