8. Memoria Dinámica

1) Escribir una función que permita leer de un stream una cadena de caracteres de longitud indefinida y la devuelva por el nombre. La función debe responder al prototipo:

```
char *read_line(FILE *);
```

2) Idem ejercicio anterior, pero devolviendo la cadena de caracteres por la interfaz de la función.

3) Escribir una función que duplique una cadena de caracteres (clonación) recibida como argumento.

4) Escribir un programa que reciba una cadena de caracteres como parámetro que comienza con espacios en blanco, y retorne por el nombre una nueva cadena limpia, desplazando los caracteres útiles hacia la izquierda (operación *left-trim*). El prototipo de la función pedida es: `char *left_trim(const char *);`

5) Ídem para función *right-trim*. Prototipo: `char *right_trim(const char *);`

6) Ídem para función *full-trim*. Prototipo: `char *full_trim(const char *);`

7) Escriba una función que permita convertir un número entero en su representación decimal como cadena de caracteres. Prototipo: `char *itoa(int n);` Sugerencia: Utilizar la función `sprintf()` de la biblioteca estándar `<string.h>`.

8) Idem para su representación octal. Sugerencia: Utilizar la función `sprintf()` de la biblioteca estándar `<string.h>`.

9) Idem para su representación hexadecimal. Sugerencia: Utilizar la función `sprintf()` de la biblioteca estándar `<string.h>`.

10) Idem para su representación binaria. Sugerencia: Utilizar la función `sprintf()` de la biblioteca estándar `<string.h>`.

11) Proponga un fragmento de código defectuoso que produzca un *memory leak*.

12) Escriba una función que genere dinámicamente una matriz identidad de dimensión n, a partir del valor de su dimensión recibido como parámetro. Realizar las validaciones que condidere necesarias.

13) Escriba una función que reciba una cadena de caracteres CSV (Comma-separated values) de longitud desconocida, y el carácter delimitador, y que devuelva un arreglo de cadenas –y su longitud– con los valores extraídos como copia de la cadena original. Prototipo:

```
char **split(const char *s, char delimitter, size_t *fields);
```

14) Modifique el ejercicio del punto anterior, para retornar el arreglo de campos CSV también por la interfaz: Prototipo:

```
status_t split(const char *s, char delimitter, size_t *fields, char ***values);
```

en donde `status_t` es un tipo enumerativo con los símbolos ERROR y OK.

15) Escriba una función que reciba una cadena de caracteres en formato ASCII y devuelva una nueva cadena codificada en formato ISO-8859-1. Para ello, construya un diccionario con los símbolos de ambos alfabetos. Prototipo de la función pedida: `char *ascii_to_iso88591(const char *s);`

16) Idem en sentido inverso: `char *iso88591_to_ascii(const char *s);`

17) Escribir un subprograma que reciba un número entero sin signo, y que retorne una cadena dinámica de caracteres con su representación binaria.

8.1. Memoria Dinámica (adicionales)

Indicar si los siguientes fragmentos de código son correctos o no, justificando exhaustivamente su respuesta, línea por línea. De contener errores, proponer una posible solución para corregirlos.

†De tener alguna eventual fuga de memoria en tiempo de ejecución, estimar cuantitativamente su magnitud en bytes (una vez corregidos todos los posibles errores sintácticos).

1)

a)

```
1  void destruir_cadenas (char *** a,
       size_t *L) {
2      while ((*L)>=0)
3          free(*a[(*L)--]);
4      free(*a);
5      *L = 0;
6  }
7  struct tm * creacion_fecha (char *str)
       /* me sirve para la fecha 13 12
       2008*/ {
8      static struct tm * fecha;
9      char * t = malloc(sizeof(str)+1);
10     memcpy(t,str,2);
11     fecha->tm_mday = atoi(t);
12     memcpy(t,str+2,2);
13     fecha->tm_mon = atoi(t) - 1;
14     memcpy(t,str+4,4);
15     fecha->tm_year = atoi(t);
16     return fecha;
17 }
```

b)

```
1  int [] cargar_registros (FILE * f, int
       cantidad)
2  {
3      int * registros = NULL;
4      char linea[ANCHO_LINEA];
5      int i = 0;
6      /* leo una cadena del archivo
           binario "f", de ANCHO_LINEA
           caracteres */
7      while (fgets(linea,ANCHO_LINEA,f)
           != NULL)
8      {
9          if ((registros = realloc(
               registros,i++)) == NULL)
               return NULL;
10         registros[i] = atoi(linea);
11     }
12     *cantidad = i;
13     return registros;
14 }
```

2)

a)

```
1  typedef char *string;
2  string s;
3  gets(s);
4  if (s == "QUIT") return exit(0);
```

b)

```
1  char * get_eq_level (int level) {
2      static char s[10];
3      if(level>0) strcpy("+");
4      else strcpy(s,"-|0");
5      return s; }
```

3)

a)

```
typedef struct _struct { float real,
    float imag} complex_t;

complex_t 2_complex(float real) {
    complex_t * p;
    p = malloc(sizeof(complex_t));
    p.real = real;
    *p.imag = 0;
    return *p;
}
char * get_error_msg(int id){
    char s[100];
    sprintf(s,"%s%i\n","ERROR:",id);
    return s;
}
```

b)

```
char * get_port_number (char const * s)
    {
    static char id;
    id = *(++s);
    return &id;
}
int main (void){
    char * str = "DSLAM-HUAWEI-5100BM";
    complex_t c;
    c = 2_complex(2,5); // creo
        complejo desde real
    fwrite(&c,sizeof(complex_t),1,
        stdout);
        // imprimo asi porque porque es
            mas veloz
    free(&c); //lo libero porque no lo
        necesito mas
    free(&2_complex(NULL)); // a ver si
        anda bien?
    puts(get_error_msg(1));
    for (i = 0; i < strlen(str); i++)
        putchar(*get_port_number(str));
    return 0;
}
```

4)

```
typedef struct _struct { float real, float imag} complex_t;
complex_t 2_complex(float real) {
    complex_t * p;
    p = malloc(sizeof(complex_t));
    p.real = real;
    *p.imag = 0;
    return *p;
}
char * get_error_msg(int id){
    char s[100];
    sprintf(s,"%s%i\n","ERROR:",id);
    return s;
}
```

5)

a)

```
int i = 0;
char *p;
#define K 1024
...
for(i = 0; i < 10000; i++){
    p = malloc(i * K);
    sprintf(p,"%i\n",i);
}
```

b)

```
char *p = "prueba.txt";
if (archivo == "c:\temp\carta.txt"){
    fprintf(stderror, "%s\n", "Error en
        factura "DGI".");
    return FAILURE_EXIT_FAILURE;
}
else exit;
```

6)

a)

```
#DEFINE MAX "1024" /* Defino una
    constante simbolica */
int i;
char original[MAX];

for ( i = 0; i < 1000; i++) {
    gets(original);
    printf("%s%s\n","original:",
        original);
    printf("%s%s\n",,"minusculas:",
        strlower(original));
}
```

b)

```
con:

char *strlower (char * cadena){
    char *s = strduplicate(*cadena);
    while (*s != '\0') { *s = tolower(*
        s); s++; }
    free(cadena);
    free(s);
    return s;
}

char *strduplicate (char *cadena){
    char *p = malloc(strlen(cadena) );
    strcpy(p, cadena);
    return p;
}
```