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: struct tm * creacion_fecha (char *
str) /* me sirve para la fecha 13 12 2008
*/ {
7:      static struct tm * fecha;
8:      char * t = malloc(sizeof(str)+1);

9:      memcpy(t,str,2);
10:     fecha->tm_mday = atoi(t);
11:     memcpy(t,str+2,2);
12:     fecha->tm_mon = atoi(t) - 1;
13:     memcpy(t,str+4,4);
14:     fecha->tm_year = atoi(t);
15:     return fecha;
}
```

b)

```
16: int [ ] cargar_registros (FILE * f, int cantidad)
17: {
18:     int * registros = NULL;
19:     char  linea[ANCHO_LINEA];
20:     int   i = 0;

21:     /* leo una cadena del archivo binario "f", de
           ANCHO_LINEA caracteres */
22:     while (fgets(linea,ANCHO_LINEA,f) != NULL)
23:     {
24:         if ((registros = realloc(registros,i++))
                == NULL) return NULL;
25:         registros[i] = atoi(linea);
26:     }
27:     *cantidad = i;
28:     return registros;
29: }
```

2)

a)

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

b)

```
5:  char * get_eq_level (int level)  {
6:     static char s[10];
7:     if(level>0) strcpy("+");
8:     else strcpy(s,"-|0");
9:     return s; }
```

3)

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

2:complex_t 2_complex(float real) {
3:    complex_t * p;

4:    p = malloc(sizeof(complex_t));
5:    p.real = real;
6:    *p.imag = 0;
7:    return *p;
}

8:char * get_error_msg(int id){
9:    char s[100];
10:    sprintf(s,"%s%i\n","ERROR:",id);
11:    return s;
}
```

```
12:char * get_port_number (char const * s) {
13:    static char id;
14:    id = *(++s);
15:    return &id;
}

16:int main (void){
17:    char * str = "DSLAM-HUAWEI-5100BM";
18:    complex_t c;
19:    c = 2_complex(2,5); // creo complejo desde real
20:    fwrite(&c,sizeof(complex_t),1,stdout);
21:      // imprimo así porque porque es más veloz
22:    free(&c); //lo libero porque no lo necesito más
23:    free(&2_complex(NULL)); // a ver si anda bien?
24:    puts(get_error_msg(1));
25:    for (i = 0; i < strlen(str); i++)
26:        putchar(*get_port_number(str));
27:    return 0;
}
```

4)

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

2:complex_t 2_complex(float real) {
3:    complex_t * p;

4:    p = malloc(sizeof(complex_t));
5:    p.real = real;
6:    *p.imag = 0;
7:    return *p;
}

8:char * get_error_msg(int id){
9:    char s[100];
10:    sprintf(s,"%s%i\n","ERROR:",id);
11:    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)

```
#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));
}
```

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;
}
```