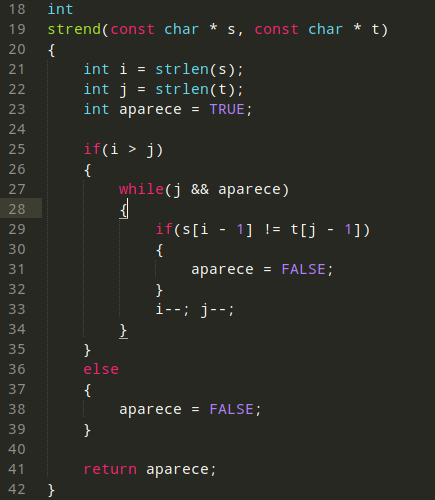
### Ejercicio 1

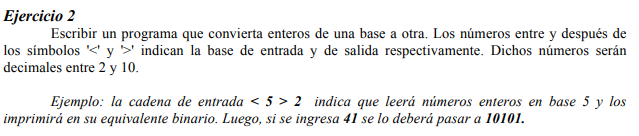
### 

### 

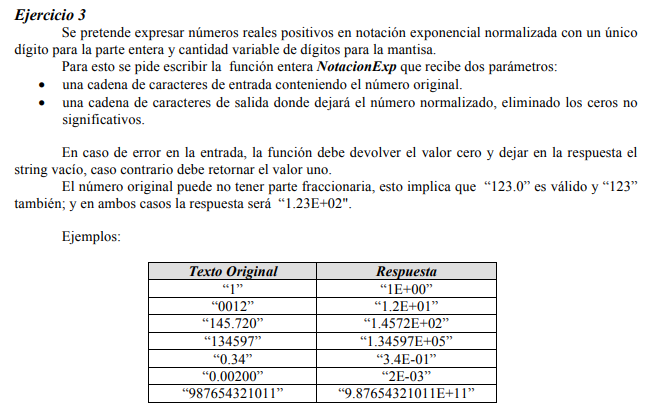


### 

### Ejercicio 2

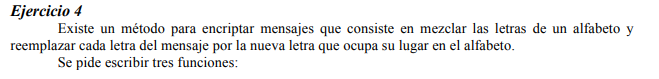


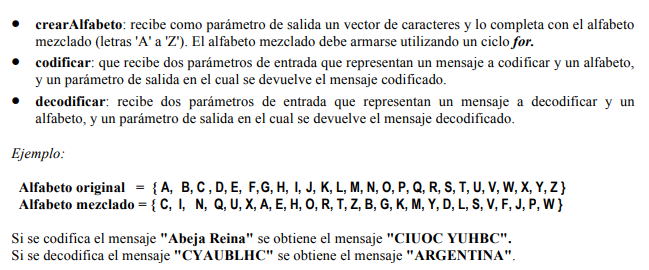
### Ejercicio 3



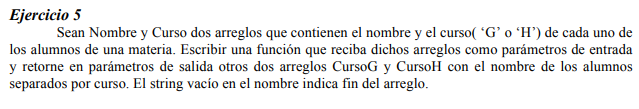
### 

### Ejercicio 4



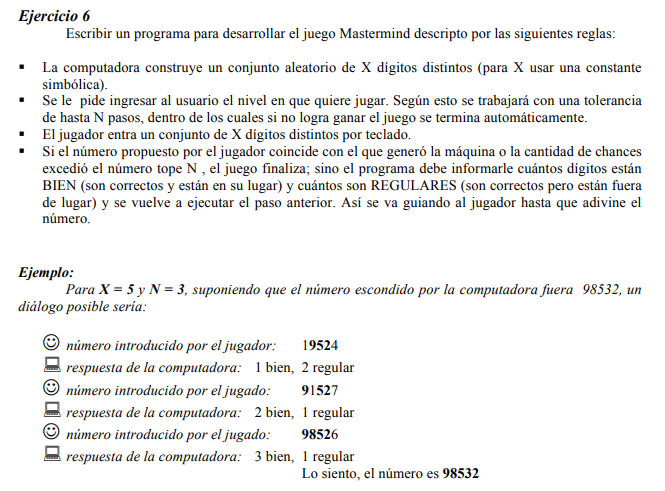


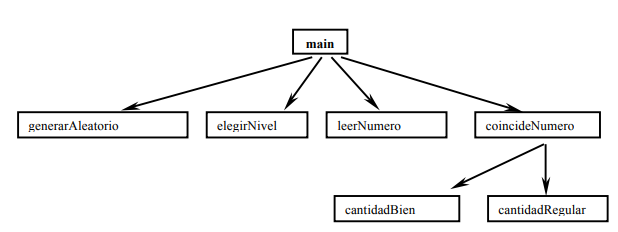
### Ejercicio 5

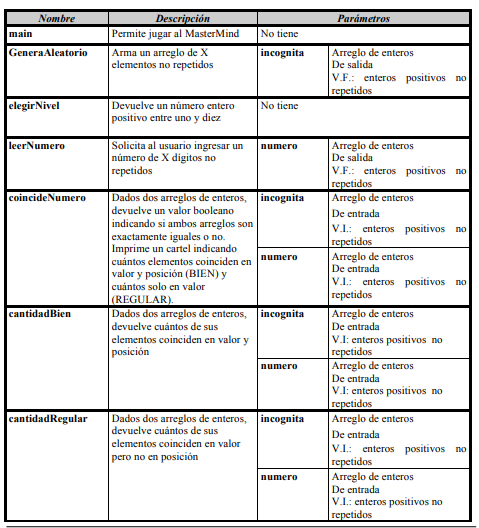


### 

### Ejercicio 6





****

### 

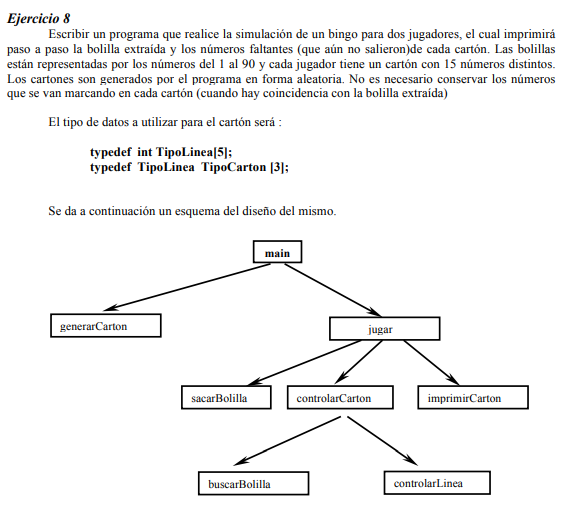
### Ejercicio 7

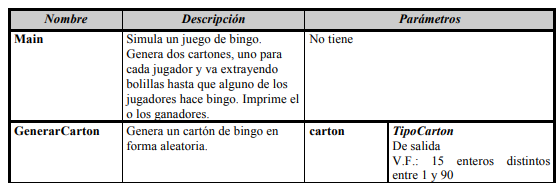


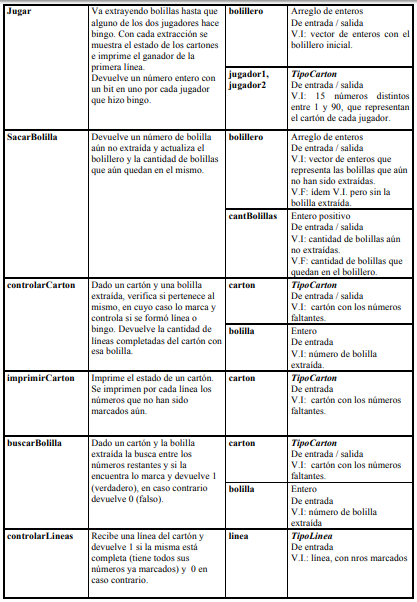
El árbol de módulos no cambia. La tabla de interfaces cambia ya que en cada función se debe añadir como parámetro la cantidad de dígitos.

### 

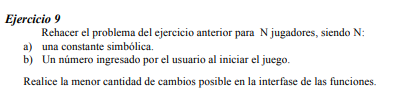
### Ejercicio 8



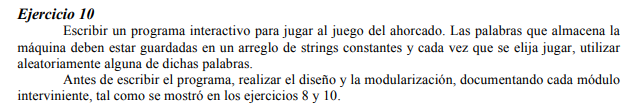




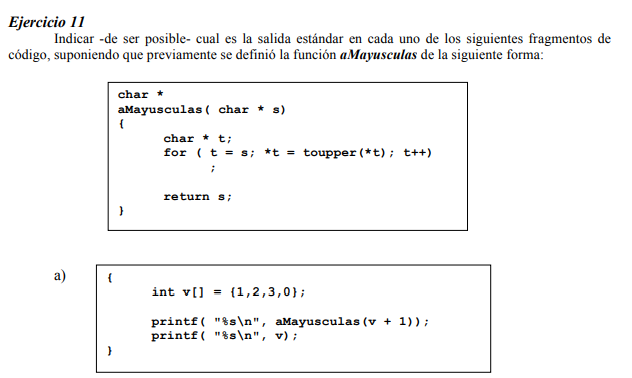
### Ejercicio 9



### Ejercicio 10



### Ejercicio 11



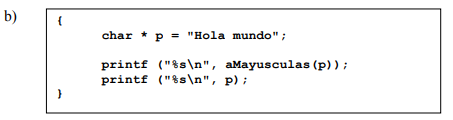
Espera un puntero a char y recibe un puntero a int.

Recibe 02 00 00 00 (little endian). El ciclo se aplica al 02 que no lo modifica por no ser una letra minúscula.

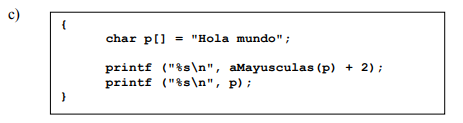
Retorna la dirección del valor 02.

La primer invocación a printf toma dicha dirección como string y como el ‘02’ no es un ASCII imprimible; imprime basura y termina.

El segundo printf imprime basura y termina.



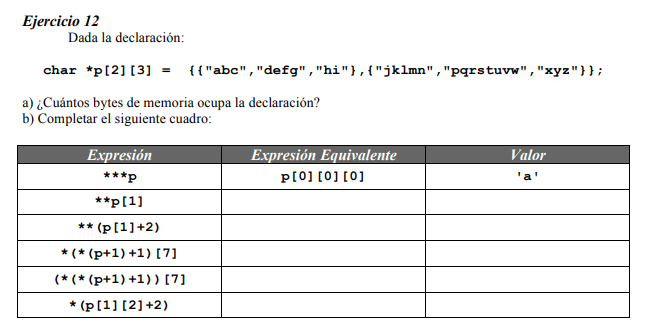
El puntero p apunta a una constante de cadena, la cual se encuentra en una zona de memoria inalterable; lo cual produce que la función “aMayusculas” aborte en ejecución.



La función aMayusuculas recibe el string p (el +2 no interviene) por lo que convierte todo el string a mayúsculas.

Imprime “LA MUNDO” y luego “HOLA MUNDO”.

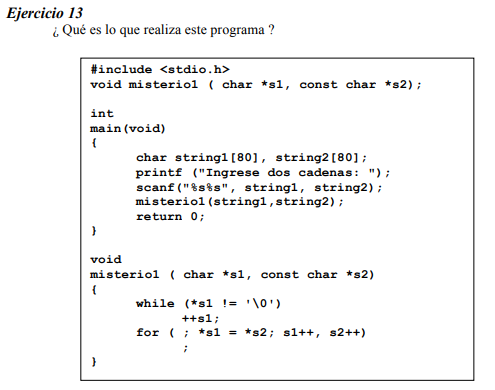
### Ejercicio 12



La declaración ocupa 24 bytes

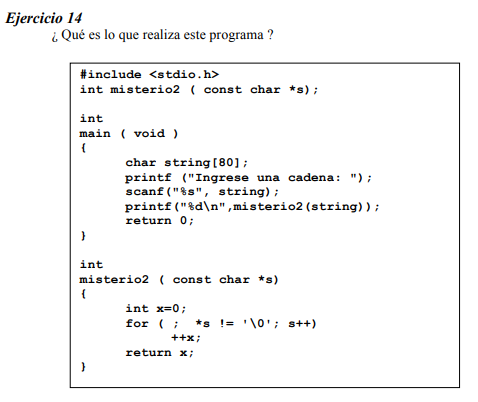
|  |  |  |
| --- | --- | --- |
| Expresion | Expresión equivalente | Valor |
| \*\*\*p | p[0][0][0] | ‘a’ |
| \*\*p[1] | p[1][0][0] | ‘j’ |
| \*\*(p[1]+2) | p[1][2][0] | ‘x’ |
| \*(\*(p+1)+1)[7] | p[1][8][0] | Error |
| (\*(\*(p+1)+1))[7] | p[1][1][7] | ‘w’ |
| \*(p[1][2]+2) | p[1][2][2] | ‘z’ |

### Ejercicio 13



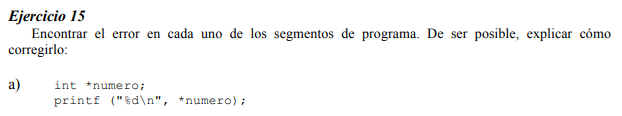
El programa concatena la cadena 2 (char \*s2) al final de la cadena 1 (char \* s1). Equivale a función de librería estándar ‘strcat’.

### Ejercicio 14

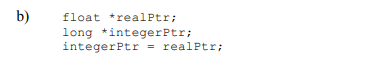


El programa cuenta la cantidad de caracteres del string. Equivale a la función de librería estándar ‘strlen’.

### Ejercicio 15



El puntero no está inicializado. Se puede corregir con funciones de asignación de memoria, o asignando a numero una dirección de memoria válida.



Se intenta asignar a un puntero, otro de distinto tipo.



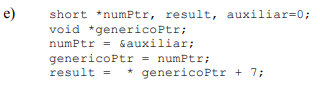
y no es un puntero. Se puede realizar lo siguiente p = &y; pero como y no está inicializada, el puntero apuntará a basura.



No se puede incrementar s, debido a que corresponde a una valor constante que indica la dirección del primer elemento del arreglo.

for(int i = 0; s[i] != ‘\0’; i++)

printf(“%c”, s[i]);



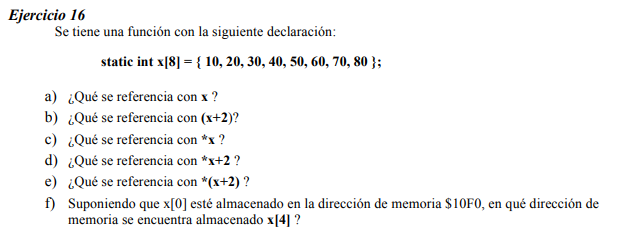
El puntero genericoPtr es de tipo void, por lo que éste no se puede desreferenciar. Primero se debe castear a tipo short \* (En este caso).

result = \* (short \*)genericoPtr + 7;



s no está incializado.

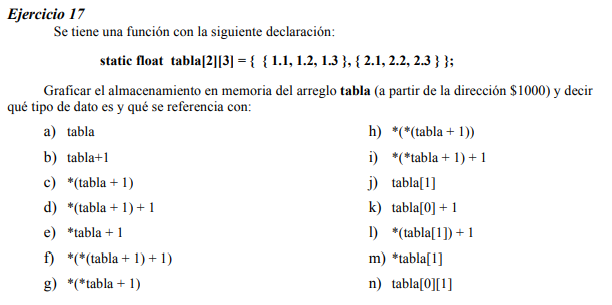
### Ejercicio 16



1. Con x se referencia la dirección de memoria del primer elemento del arreglo. x[0].
2. Con (x+2) se referencia la dirección de memoria del tercer elemento. x[2].
3. Con \*x se accede al primer elemento del arreglo.
4. Primero desreferencia x, y luego le suma 2. Se obtiene 12.
5. Devuelve 30.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 10F |  | | | |  | | | |  | | | |  | | | |
| 110 | x[4] | | | |  | | | |  | | | |  | | | |

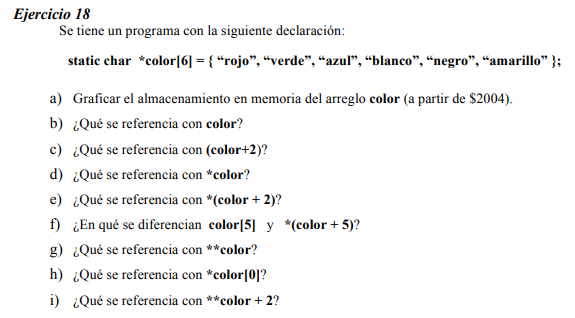
### Ejercicio 17



Considerar float de 4 bytes.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 100 | 1.1 | | | | 1.2 | | | | 1.3 | | | | 2.1 | | | |
| 101 | 2.2 | | | | 2.3 | | | |  | | | |  | | | |
| 102 |  | | | |  | | | |  | | | |  | | | |

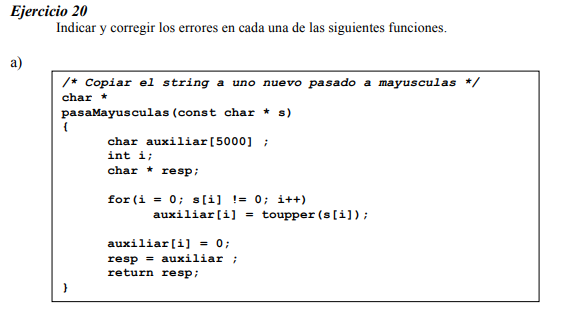
### Ejercicio 18

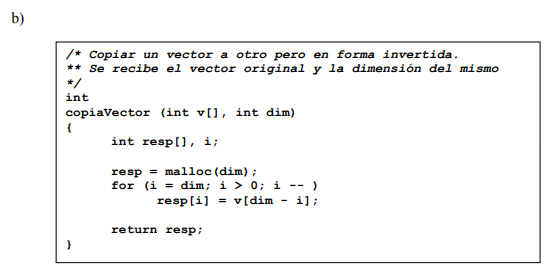


### Ejercicio 19

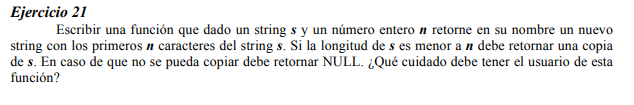


### Ejercicio 20



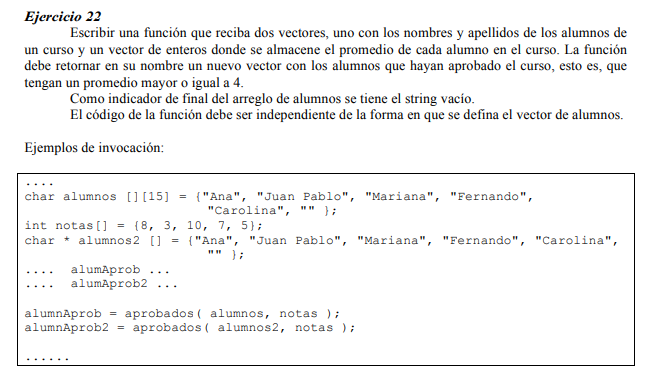


### Ejercicio 21



El usuario luego debe encargarse de liberar la memoria utilizada.

### Ejercicio 22



### Ejercicio 23

