4. Punteros

1) Escribir la definición de variable puntero.

2) Dadas las siguientes declaraciones:

```
1  int x, int_array[MAX];
2  x = int_array[4];
```

explicar qué traduce el compilador para acceder al quinto elemento del arreglo y asignárselo a `x`.

3) Dadas las siguientes declaraciones:

```
1  int *ip, **ipp, (*ip4)[4], i, j;
2  int ventas[3][4];
```

explicar las siguientes expresiones:

a) `ip4 = ventas;`

b) `ip = (int *)ventas;`

c) `ipp = (int **)ventas;`

d) `*(*(ip4 + i)+ j);`

e) `*(*(ventas + i)+ j);`

4) ¿Qué diferencia hay entre el nombre de un arreglo y un puntero?

5) ¿Cómo sabe el compilador el tamaño de un objeto al que apunta un puntero?

6) Escribir un programa que imprima cada uno de los elementos de un arreglo dos dimensional utilizando un puntero para acceder a los mismos, en lugar de utilizar subíndices. Utilizar el siguiente arreglo y los punteros indicados abajo:

```
1  int dos_vector[3][4] = {{1,2,3,4}, { 5,6,7,8}, {9,10,11,12}};
2  int *dos_ptr;
3  int (*ptr2vector)[4];
4  int fila, col;
```

7) a) Explicar las diferencias entre estas declaraciones:

```
1  char cadena1[] = "Hola";
2  char *cadena2 = "Hola";
```

```
1  char meses1[12][] = {"Enero", "Febrero", ..., "Diciembre"};
2  char *meses2[12] = {"Enero", "Febrero", ..., "Diciembre"};
```

b) Con las definiciones anteriores, analizar si los siguientes fragmentos son correctos y qué hacen:

```
1  cadena1 = "Chau";
2  cadena2 = "Chau";
```

```
1  strcpy(cadena1, "Chau");
2  strcpy(cadena2, "Chau");
```

```
1  strcpy(cadena1, "Hola y chau");
```

8) Explicar qué pasa si se olvida el carácter nulo como último carácter de una cadena (string).

9) Escribir un programa en el que se defina un arreglo de 10 punteros a `float`, se lean diez números en las ubicaciones en las que hacen referencia los punteros. Se sumen todos los números y se almacene el resultado en una dirección a la que haga referencia un puntero. El programa deberá mostrar el contenido de todas las variables, tanto los punteros como los números de tipo `float`.

10) Explicar qué es una variable tal como la que se define en la siguiente declaración; dé un ejemplo de valor posible para la misma: `int ***miVariable` y trate de representarla gráficamente.

11) Explicar el significado de las siguientes declaraciones:

a) `int (*uno)[12];`

b) `int *dos[12];`

c) `void *fu();`

d) `void (*fa)();`

Nota: Los siguientes ejercicios se refieren a programas ANSI-C modularizados.

4.1. Funciones con punteros

12) Escribir un procedimiento que reciba como argumento un arreglo de caracteres y lo devuelva invertido por la interfaz:

```
void string_reverse(char *);
```

13) Escribir un procedimiento que reciba una cadena de caracteres `s` y un arreglo de caracteres con espacio suficiente `t`, y copie la cadena en el arreglo, terminando la cadena con el caracter `'\0'` (función `strcpy()` de la biblioteca `<string.h>`):

```
void strcpy (char *t, const char *s);
```

14) Escribir un procedimiento que reciba como argumento dos cadenas de caracteres, y realice la concatenación de una sobre la otra, terminando la cadena con el caracter `'\0'` (función `strcat()` de la biblioteca `<string.h>`):

```
void strcat(char *t, const char *s);
```

15) Escribir un procedimiento que reciba una cadena de caracteres, un arreglo de caracteres con espacio suficiente, y una variable entera `n`, y copie los primeros `n` caracteres de la cadena sobre el arreglo, sin terminar la cadena con el caracter nulo (función `strncpy()` de la biblioteca `<string.h>`):

```
int strncpy(char *t, const char *s, int n);
```

16) Escribir una función que convierta a minúsculas una cadena de caracteres recibida como argumento:

```
void strlwr(char *);
```

17) Escribir una función que convierta a minúsculas una cadena de caracteres recibida como argumento:

```
void strupr(char *);
```

18) Escribir una función que reciba una cadena de caracteres como argumento, y la convierta a minúsculas o mayúsculas, de acuerdo a una opción ingresada por el usuario a través de un parámetro de tipo `case_t`, definido como: `typedef enum {UPPERCASE, LOWERCASE} case_t;`

El prototipo de la función pedida es:

```
void change_case(char *, case_t);
```

19) Escribir una función que responda al siguiente prototipo:

```
void replace(char *s, char nuevo, char viejo);
```

y reemplace en la cadena `s` todas las apariciones del caracter `viejo` por el carácter `nuevo`.

20) Escribir un procedimiento que reciba como parámetro una cadena de caracteres que comienza con espacios en blanco, y los elimine desplazando los caracteres útiles hacia la izquierda (operación *left-trim*):

```
void left_trim(char *);
```

21) Escribir un procedimiento que reciba como parámetro una cadena de caracteres que finaliza con espacios en blanco, y los elimine desplazando los caracteres útiles hacia la izquierda (operación *right-trim*):

```
void right_trim(char *);
```

22) Escribir una función que reciba una matriz cuadrada de enteros y su dimensión, y determine si es una matriz simétrica o no, retornando el resultado por el nombre.

23) Escribir un subprograma que reciba una matriz cuadrada de doubles y su dimensión, y retorne el valor de su traza.

24) Escribir un subprograma que reciba una matriz cuadrada de doubles y su dimensión, y retorne el valor de su determinante.

25) Escribir un subprograma que calcule la partes entera y decimal de cualquier número real recibido como argumento, y las retorne por su interfaz.

26) Escribir un subprograma que convierta un número que representa una cantidad de segundos, a su equivalente en horas, minutos y segundos, retornando las partes por la interfaz.

27)

a) Escribir una función que reciba las coordenadas rectangulares de dos puntos del plano y devuelva la distancia entre ellos.

b) Valiéndose de la función del punto anterior, escribir una función que calcule la longitud total de un segmento de `n` puntos bidimensionales, respondiendo al siguiente prototipo:

```
double longitud_segmento(double puntos[][2], size_t n);
```