PRÁCTICA 2

1. PUNTEROS (CONTINUACIÓN)

1) Se pide escribir un programa que mantenga la información de 10 números complejos. El siguiente código muestra una solución propuesta. Explique qué hace. Esquematice gráficamente las estructuras de datos usadas y las variables. Ejemplifique entradas y salidas. ¿Es correcta dicha propuesta para solucionar el problema? Implementar las funcines `leerComplejo()` y `mostrarComplejo()`.

Algorithm 1: Punteros

```
1  #define DIM 10
2  typedef struct {
3      int real, imag;
4  } Complex_T;
5  Complex_T leerComplejo(); // lee los campos de un complejo y lo retorna
6  int mostrarComplejo(Complex_T num_c); // muestra los campos del complejo num_c
7  int main()
8  {
9      Complex_T *vector[DIM];
10     Complex_T *p_num;
11     int i;
12     Complex_T num1;
13     // inicializacion del vector de complejos
14     for(i=0; i <DIM; i++) {
15         num1 = leerComplejo();
16         vectorC[i] = &num1;
17     }
18     // impresion del vector de complejos
19     for(i=0; i <DIM; i++) {
20         p_num = vectorC[i];
21         mostrarComplejo(*p_num);
22     }
23     return 0;
24 }
```

2) A continuación se describe el ejemplo del libro "*Introducción a la Programación en C*" de los autores Marco A. Peña Basurto y José M. Cela Espin (Aula Politécnica /ETSETB). Dicho ejemplo está en el capítulo 9, ejemplificando el uso de punteros y tablas. Analizar y describir qué es lo que hace este código.

Algorithm 2: Fuerza

```
1  #define DIM 10

2  int main()
3  {
4      int v1[DIM], v2[DIM];
5      int fuerza1, fuerza2, i;
6      int *fuerte;

7      /* lectura de los vectores */
8      for(i=0; i <DIM; i++) {
9          scanf("%d ", &v1[i]);
10     }
11     for(i=0; i <DIM; i++) {
12         scanf("%d ", &v2[i]);
13     }

14     /* cálculo de fuerza de los vectores */
15     fuerza1 = fuerza2 = 0;
16     for(i=0; i <DIM; i++) {
17         fuerza1 += v1[i];
18         fuerza2 += v2[i];
19     }
20     if (fuerza1 >fuerza2)
21         fuerte = v1;
22     else
23         fuerte = v2;

24     /* escritura del vector más fuerte */
25     for(i=0; i <DIM; i++) {
26         printf("%d ", fuerte[i]);
27     }
28     return 0;
29 }
```

2. MEMORIA DINÁMICA

Objetivos: aprender a utilizar memoria dinámica.
Nota: cada vez que se aloca memoria de forma dinámica, recordar liberarla. Uso de librerías `stdlib.h` **y** `string.h`**.**

3) Explique: ¿qué diferencia hay entre las operaciones `malloc()`, `calloc()` y `realloc()`?. Explique para cada una el tipo retornado y cada uno de los parámetros que recibe.

4) Realizar un programa que lea un valor del teclado. Alocar en memoria dinámica un vector de enteros de longitud igual al valor leído. Inicializar dicho vector con valores aleatorios entre 10 y 20. Imprimir el vector. Implementar usando funciones.

5) Explique qué realiza el siguiente programa. Agregue comentarios explicando cada parte del código.

Algorithm 3: Memoria dinámica

```
1  #include <stdio.h>
2  #include <stdlib.h>

3  int main()
4  {
5      float *pf = NULL;
6      int i, num;

7      do{
8          printf(”Ingrese la dimensión del vector");
9          scanf("%d", &num);
10     }while(num <1);

11     pf = (float*) calloc(num, sizeof(float));
12     if(pf == NULL) { printf(”Error en la asignacion de memoria"); exit(-1); }

13     printf(”Ingrese %d valores", num);
14     for(i=0; i<num; i++) {
15         scanf("%f", &pf[i]);
16     }

17     free(pf);
18     return 0;
19 }
```

6) Escribir un programa que lea del teclado 2 valores que indican filas y columnas de una matriz (se aloca en tiempo de ejecución). Inicializar dicha matriz, y para cada fila de la matriz, indicar el valor máximo y el valor mínimo. Implementar con funciones.

7) Se tiene un arreglo de n elementos enteros (n es ingresado por teclado y se aloca memoria de forma dinámica). Se inicializa el vector con valores aleatorios entre 0 y VALOR_MAX. Se quiere generar otro arreglo donde estén los elementos del vector original sin repetir. Escribir un programa que genere el nuevo vector, cuya dimensión sólo se conoce en tiempo de ejecución. Luego muestre en pantalla ambos vectores. Nota: no implementar con un arreglo de N datos. Ir realocando en memoria dinámica espacio a medida que el vector resultado va aumentando.

8) Escriba un programa para generar matrices cuadradas de enteros (el usuario introduce la dimensión por teclado y todas las matrices generadas tienen el mismo tamaño). El programa reserva memoria todas las matrices requeridas. Se pide implementar funciones para inicializar matrices: con números aleatorios entre 1 y 9, matriz simétrica con números aleatorios entre 1 y 9, matriz identidad. Luego, implemente funciones para imprimir una

matriz y funciones para resolver la suma, resta, multiplicación y transpuesta de matrices. En el código de la función `main()` se debe resolver A * I = A, donde I es la matriz identidad y A es una matriz cualquiera. Mostrar por pantalla las matrices para verificar esta igualdad.

9) Declare una estructura para representar un punto en el espacio tridimensional (con campos x,y,z). Declare un puntero a la estructura para que tenga la dirección de un arreglo dinámico de n estructuras punto. Utilizar la función `calloc()` para asignar memoria al arreglo y compruebe que se ha podido asignar la memoria requerida. Escribir una función que retorne un puntero a un array de estructuras punto. Los valores de los puntos se ingresan en esa función.

10) Vector de vectores: realizar un programa que lea un valor. Dicho valor especifica la cantidad de elementos de un arreglo de vectores de enteros. Para cada posición del vector, pedir la dimensión del subvector y alocar memoria para dicho vector. Inicializar toda la estructura y luego, imprimir los datos del vector. Desalocar memoria de forma correcta. Notar que es necesario mantener la información de la dimensión de cada uno de los subvectores. Una forma de realizarlo es que cada subvector sea una estructura de la forma:

```
typedef struct
{
   int n;
   int *vector;
}Vector_T;
```

donde el campo `n` de la estructura es la dimensión del vector. El puntero a enteros es el arreglo dinámico con `n` posiciones.

3. Punteros a funciones

11) Especificar, para las siguientes lineas de código qué significa cada una:

Algorithm 4: Puntero a funcion

```
1 int (*puntero_funcion)(int);
2 int *puntero_funcion (int);
```

Si el objetivo es declarar un puntero a función que retorna un int y recibe un int. ¿Son ambas declaraciones correctas?

12) Retomar los algoritmos de ordenación vistos anteriormente e implementar un programa donde se le pida al usuario qué algoritmo quiere utilizar: (1) Método de la burbuja, (2) Método de Inserción, (3) Método de Selección, (4) Shell Sort. Declarar un puntero a una función que ordene un vector de enteros de dimension DIM. Elegir en tiempo de ejecución el método de ordenación e invocarlo utilizando un puntero a función. Tomar tiempos utilizando las funciones `clock()` de la librería `time.h` o `gettimeofday()` de `sys/time.h` (investigar).

13) **Calculadora:** Implementar una calculadora que sepa sumar, restar, multiplicar y dividir 2 números flotantes. Mostrar un menú con las 4 opciones, y utilizando puntero a función, realizar la operación solicitada y mostrar el resultado.