Ejercitación № 3: Estructuras (structs)

Cátedra Programación II **Parte II**: *Lenguaje C* Junio 2019

Desarrollar cada uno de los siguientes ejercicios de forma completa, detallando cada una de las etapas involucradas en la construcción de un programa, tal cual las presentamos en La Receta. Tener en cuenta que deberá crear funciones que permitan el pasaje de argumentos para posteriormente testear el programa contra los ejemplos inicialmente escritos.

1. Ejercicios Simples con Estructuras

Ejercicio 1. ¿Cuál es la diferencia entre los siguientes tres programas?

```
1
2   (a)  #include <stdio.h>
3        struct point { double x; double y; };
4        int main(void) {
5           struct point test;
6           test.x = .25; test.y = .75;
7           printf("[%f %f]\n", test.x, test.y);
8           return 0;
9        }
10
11  (b)  #include <stdio.h>
12       typedef struct { double x; double y; } Point;
13       int main(void) {
14          Point test;
15          test.x = .25; test.y = .75;
16          printf("[%f %f]\n", test.x, test.y);
17          return 0;
18       }
19
20  (c)  #include <stdio.h>
21       typedef struct { double x; double y; } Point;
22       int main(void) {
23          Point test = {.25, .75};
24          printf("[%f %f]\n", test.x, test.y);
25          return 0;
26       }
```

Código 1: Buscando las diferencias

Ejercicio 2. ¿Qué problema existe en las siguientes declaraciones de estructuras?

```
1       A.  struct point ( double x, y )
2       B.  struct point { double x, double y };
3       C.  struct point { double x; double y }
4       D.  struct point { double x; double y; };
5       E.  struct point { double x; double y; }
```

Código 2: Declaraciones con Problemas

Ejercicio 3. Utilizando la definición de la estructura **Point** resuelva:

```
1  struct point {
2     double x;
3     double y;
4  };
5  typedef struct point Point;
```

Código 3: Estructura Point

1. Definir la función **POINTdist()** que calcula la distancia Euclidiana entre dos puntos dados.
2. Definir la función **POINTeq()** la cual retorna 1 si dos puntos son iguales; y 0 en caso contrario. Recordar que entre números flotantes (reales) no tienen mucho sentido chequear exactitud, pero si podemos verificar que la diferencia en valor absoluto sea menor a 0,000001.
3. Definir el tipo **Rectangulo** para aquellos rectángulos paralelos a los ejes en un sistema de coordenas cartesianas. Representar a una rectángulo por sus vértices izquierdo inferior y derecho superior. Usar para estos últimos puntos, la estructura **point** definida anteriormente.
4. Definir la función **RECTarea()** la cual calcula el área de un rectángulo.
5. Definir la función **RECTin()** la cual retorna 1 si un punto se encuentra dentro del rectangulo, y 0 si esta a fuera. Utilizar los tipos **Point** y **Rectangulo** definidos anteriormente.

2. Definiendo Nuevas Estructuras

Ejercicio 4. El **Tiempo**

1. Proponer una representación con estructuras para modelar el tiempo: `hhhh:mm:ss`. No impondremos ninguna restricción con respecto a la cantidad de horas almacenadas.
2. Escriba una función de creación de un elemento del tipo ***Tiempo***, el cual tome tres datos, la hora, los minutos, y los segundos, y retorne un elemento de tipo ***Tiempo*** válido.
3. Escribir una función **imprimir_Tiempo** la cual recibe un dato de tipo Tiempo e imprime este con el siguiente formato: **hh:mm:ss**.
4. Escribir una función **suma_Tiempo** que reciba dos tiempos dados y devuelva la suma de éstos.
5. Escribir una función **resta_Tiempo** que reciba dos tiempos dados y devuelva la resta de éstos.
6. Escribir una función **Tiempo_Dias** que reciba un dato de tipo Tiempo, y devuelve la cantidad de días que representa. En el caso que la cantidad de horas sea menor a 24 devolverá cero, y si por el contrario la cantidad de horas superará los 365 días (8760 horas corresponden a un año), por ejemplo, si el dato ingresado fuera 8800 horas 15 min 30 seg, la función deberá retornar, entonces devolverá 1 año y 40 días.

Ejercicio 5. Los **Números Complejos**

1. Proponer una representación con estructuras para modelar un número complejo: $\mathbf{2 + 5i}$.
2. Escriba una función de creación de un elemento del tipo ***Complejo***, el cual tome dos datos, la parte entera y la parte compleja de un número, y retorne un elemento de tipo ***Complejo*** válido.
3. Escribir una función **suma_complejo** que reciba dos números complejos y devuelva la suma de éstos.

4. Escribir una función **resta_complejos** que reciba dos números complejos y devuelva la resta de éstos.
5. Escribir una función **multiplica_complejos** que reciba dos números complejos y devuelva el producto de éstos.

Ejercicio 6. Una **Persona**

1. Proponer una representación con estructuras para modelar una persona.
2. Escriba una función de creación de un elemento del tipo ***Persona***, el cual toma cinco datos de una persona, como por ejemplo, nombre, apellido, dni, teléfono, y edad, y retorna una elemento de tipo ***Persona*** válido.
3. Escriba la función **imprimir_Persona**:**Persona** → **void**, la cual muestra en pantalla los datos de un elemento del tipo ***Persona***.
4. Escriba la función **igual_Identidad**:**Persona** → **Persona** →**Int**, la cual compara si dos datos de tipo persona son iguales, basándose en que coincidan su número de dni.
5. Escriba una función que indique si dos ***Personas*** son familiares. Una persona es familiar de otra, si tiene el mismo apellido, y dirección.
6. Diseñe las siguientes tres estructuras para que contengan al tipo ***Persona*** como dato: **Estudiante**, **Empleado**, **Socio**, y para ellas defina las siguientes funciones, reutilizando las funciones antes definidas para la estructura **Persona**:
   - Creación de elementos de la estructura.
   - Impresión de datos de la estructura.

3. Problemas mas Complejos

Ejercicio 7. Las **Cartas**

1. Proponga una representación con estructuras para las cartas de la baraja española.
2. Escriba una función de creación de un elemento del tipo ***Carta***, la cual tome dos datos, el valor y el palo de una carta, y retorne un elemento de tipo ***Carta*** válido.
3. Escriba una función **barajar** la cual no recibe argumentos, pero entrega una carta válida de la baraja española. Utilizar alguna función de aleatoriedad para asignar el número y el palo de la carta.
4. Escriba las siguientes funciones:
   - **es_igual**, : ***Carta*** → ***Carta*** → ***Int***, la cual determina si dos cartas son iguales o no.
   - **es_igualPalo**: ***Carta*** → ***Carta*** → ***Int***, la cual determina si dos cartas tienen igual palo.
   - **es_igualNumero**: ***Carta*** → ***Carta*** → ***Int***, la cual determina si dos cartas tienen el mismo número.
   - **es_full**: ***Carta*** → ***Carta*** → ***Cartas*** → ***Cartas*** → ***Int***, determina si las cartas pasadas forman un *full*, dos pares de cartas de igual número y distinto palo. Utilizar las funciones anteriormente definidas.
   - **es_poker**: ***Carta*** → ***Carta*** → ***Cartas*** → ***Cartas*** → ***Int***, determina si las cartas pasadas forman un *poker*, igual número distinto palo. Utilizar las funciones anteriormente definidas.
   - **es_escalera**: ***Carta*** → ***Carta*** → ***Cartas*** → ***Int***, la cual determina si las cartas pasadas como argumento forman una:

- *escalera verdadera*: números consecutivos e igual palo. El orden de las cartas, para un mazo de 48 cartas, es: 1, 2, 3, 4, 5, 6, 7, 8, 9, sota, caballo, rey.
- *escalera falsa*: números consecutivos pero distinto palo;
- *escalera real*: números consecutivos e igual palo, pero admite circularidad: luego del rey puede seguir el 1, y así sucesivamente, formando así una escalera, como por ejemplo, con la siguiente tirada de cartas: rey-1-2, o caballo, rey, 1, etc.

5. Escriba un programa que luego de barajar, y obtener tres cartas aleatoriamente, determine que forman las mismas, empleando las funciones anteriormente definidas. Luego muestre por pantalla lo obtenido y clasificado, y por último que ofrezca al usuario la posibilidad de volver a barajar, o terminar. No llevamos ningún tipo de registro de cartas que salieron, cuando decimos barajar, nos referimos a obtener una carta aleatoriamente. ¿Como implementaría además un sistema de puntaje para este juego?

Referencias

[1] Think Python: How to Think Like a Computer Scientist, Allen B. Downey, 2nd Edition, Version 2.2.18.

[2] Algoritmos y Programación I, Aprendiendo a programar usando Phyton como herramienta, Rosita Wachenchauzer et.al., 2016, (sin publicar).