T.P. N° 8: Estructuras

Ejercicio 1

Indique si cada uno de los siguientes enunciados es verdadero o falso. Si es falso, explicar por qué:

a) Las estructuras pueden contener únicamente un tipo de datos.

b) Dos uniones pueden ser comparadas entre sí para determinar si son iguales

c) La palabra reservada **typedef** se utiliza para definir nuevos tipos de datos.

d) Las estructuras no pueden ser comparadas.

e) Una estructura de *n* elementos siempre ocupa más memoria que *n* variables del mismo tipo que los campos de la estructura.

f) Con **typedef int Matriz[5][10]** se reservan 50 * sizeof(int) bytes de memoria.

Ejercicio 2

Encontrar el error en cada uno de los siguientes puntos:

a) Suponer que **struct carta** se ha definido conteniendo dos apuntadores al tipo **char**: **figura** y **palo**. También, la variable **c** ha sido declarada del tipo **struct carta** y la variable **cPtr** ha sido declarada como puntero a **struct carta**. La variable **cPtr** ha sido asignada a la dirección de **c**. Se imprime `printf("%s\n", *cPtr->figura);`

b) Suponer que el arreglo **corazones[13]** ha sido declarado del tipo **struct carta**.

Se intenta imprimir el miembro **figura** del elemento 10 del arreglo con `printf("%s\n", corazones.figura);`

c) Se intenta declarar e inicializar la variable **v**:

```
union valores
{
char w;
float x;
double y;
}  v = {1.27};
```

d) Se declara la estructura **persona**:

```
struct persona
{
      char apellido[15];
      char nombre[15];
      int edad;
}
```

e) Suponiendo que **struct persona** ha sido definida como en la parte (d), pero con la corrección apropiada, se declara la variable **m** a través de: `persona m;`

Ejercicio 3

a) Simular un juego de naipes de 52 cartas, en el cual se mezcle el mazo y se tomen las 5 primeras cartas, indicando si se obtuvo alguna de las siguientes configuraciones:

- PAR: hay sólo dos de las cinco cartas con igual valor
- PIERNA: hay sólo tres de las cinco cartas de igual valor
- POKER: hay cuatro cartas con igual valor

b) Extender el punto anterior de modo tal que jueguen la computadora contra el usuario hasta que se acabe el mazo o el usuario decida no seguir jugando. Acumular los puntajes obtenidos por ambos jugadores considerando que el ganador de cada mano obtiene 1 punto y solo se considera empate (un punto para cada uno) una mano donde ambos jugadores tienen la misma configuración y con la misma figura (de coincidir solo en configuración, gana aquella de figura mayor).

Ejercicio 4

Una partícula realiza un camino aleatorio dentro de un círculo de acuerdo a la siguientes reglas :

- En tiempo t = 0 la partícula está en el centro (x = 0, y = 0).
- La partícula hace un paso aleatorio en una de las cuatro direcciones dada por
  - x = x - 1
  - x = x + 1
  - y = y - 1
  - y = y + 1
- La caminata termina cuando se sale fuera del círculo ($x^2 + y^2 >= r^2$).

Considerando cada punto como una estructura de componentes cartesianas y teniendo en cuenta que el tiempo se mide con un contador que se incrementa en cada paso de la partícula, escribir un programa que para distintos tamaños de círculos determine experimentalmente la relación entre el tiempo requerido para terminar la caminata y el valor del radio.

Mostrar los valores hallados mediante una tabla cuyas columnas sean:

- ☞ Radio
- ☞ Tiempo
- ☞ Relación (radio / tiempo)

Ejercicio 5

Escribir un programa para hallar las raíces de una función matemática en un intervalo cerrado, recorriéndolo de forma tal que el intervalo quede dividido en 100000 (cien mil) particiones o subintervalos. Ejemplo: si el intervalo es [1, 50000] deberá evaluar la función en los puntos 1, 1.5, 2, etc. (también se tomarán como válidos los puntos 1, 1.49999, etc).

El programa deberá solicitar los extremos del intervalo, imprimiendo los resultados en la salida estándar. La función a evaluar recibe y devuelve un valor real y está dada por la macro FUNCION.

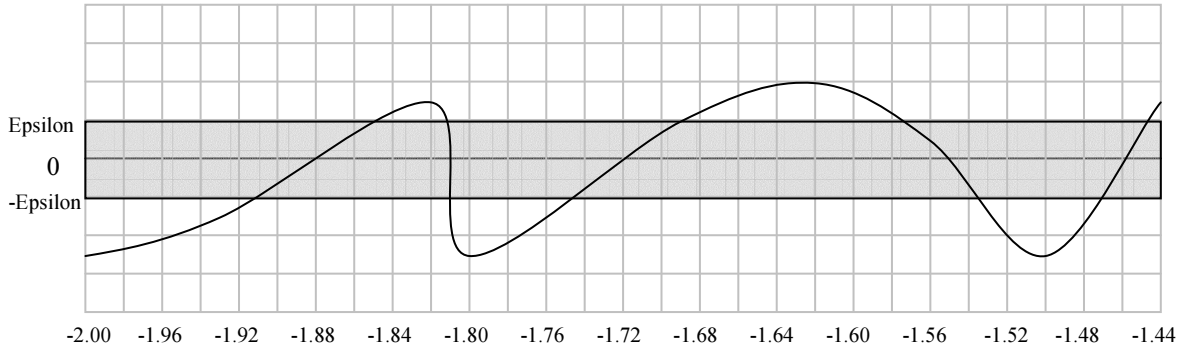
La función que realice la búsqueda de las raíces debe recibir como parámetros una estructura que represente al intervalo, regresando en su nombre una estructura que empaquete un arreglo con aquellas particiones donde haya raíces y la dimensión de dicho arreglo.

Para detectar una raíz se deben considerar dos casos:

- Que la función cambie de signo entre dos puntos: En ese caso se agrega al arreglo una partición con ambos puntos.
- Que la función se haga cero en un punto (considerando un error de EPSILON): En ese caso la partición que se agrega al arreglo de resultados está formada por el punto anterior al que se detectó como raíz y el próximo que no lo sea.

Ejemplo:

Considerando un intervalo cuyo paso es de 0.02, se grafica un fragmento del mismo indicando que raíces se encuentran:



Se detectarán raíces en :

- **-1.90**, porque el valor absoluto de la imagen en ese punto es menor que EPSILON. Pero los dos puntos siguientes (-1.88 y -1.86) también se consideran raíces por el mismo motivo. Por lo tanto en este caso la partición que se agrega con resultado toma los valores **-1.92** (por ser el anterior a la primera raíz) y **-1.84** (por ser el posterior a la última raíz).
- luego se detecta un cambio de signo de la función entre los puntos **-1.82** y **-1.80**, por lo que se agrega dicha partición.
- **-1.74** es raíz por comparación con EPSILON, al igual que los dos puntos siguientes (-1.72, -1.70) por lo que se agrega la partición **-1.76**, **-1.68**.
- **-1.56** es raíz por comparación con EPSILON, al igual que el punto siguiente -1.54, por lo que se agrega la partición **-1.58, -1.52**.
- **-1.46** es raíz por comparación con EPSILON, y el punto siguiente ya no lo es , por lo que se agrega la partición **-1.48, -1.44**.

Por lo tanto nuestra función debería retornar una estructura conteniendo los siguientes valores:

{ 5, { { -1.92, -1.84}, { -1.82, -1.80}, { -1.76, -1.68}, { -1.58, -1.52}, { -1.48, -1.44} } }