***TP N° 8: Estructuras***

La siguiente guía cubre los contenidos vistos en la clase teórica:

**15. Registros y uniones**

Antes de empezar con los ejercicios de esta guía se recomienda la resolución de la **Autoevaluación 8** disponible en Campus ITBA, para consolidar los contenidos vistos en la teórica.

***Ejercicio 1***

Corregir los siguientes fragmentos de código. El rojo no estaba, lo puse yo (javi).

| char c = 'a';  char \*p = malloc(sizeof(char));  \*p = &c; |
| --- |

| struct art {  int code;  double price;  }; no veo el error  struct art \* newArt(int code, double price) {  struct art \* p = calloc(1, sizeof(struct art));  p->code = code;  p->price = price;  return p;  } |
| --- |

| int \* p = malloc(n \* sizeof(int));  int \* q = p;  /\* llenamos y usamos el vector \*/  // ...  // ...    // Ya no los necesitamos, podemos liberar ambos vectores  free(p); uno de estos no hace falta porque ya estamos  free(q); liberando la memoria. |
| --- |

| int \* p = malloc(n \* sizeof(int));  int \* q = p;  /\* llenamos y usamos el vector \*/  // ...  // ...    // Ya no los necesitamos, podemos liberar ambos vectores  free(q); Exactamente el mismo error que antes solo que  free(p); cambiado de orden. |
| --- |

1. Fragmento de código para buscar secuencias de repetidos en un vector

| for (int i=0; i<dim-1 ; i++){  if(vec[i] == vec[i+1]) { |
| --- |

f)

| struct art {  int code;  double price;  };  struct art \* newArt(int code, double price) {  struct art p; Hay que asignarle memoria a ‘p’.  p.code = code;  p.price = price;  return &p; ← hay que devolver p no &p  } |
| --- |

g)

| int \* p;  p = realloc(p, cant \* sizeof(int)); |
| --- |

No tiene sentido el codigo de arriba porque estamos tratando de modificar la memoria de un puntero a int sin antes haberla definido.

Habria que agregar lo siguiente: int \* p = NULL;

***Ejercicio 2***

Indique si cada uno de los siguientes enunciados es verdadero o falso. Si es falso,

explicar por qué:

1. Las estructuras pueden contener únicamente un tipo de datos. F las estructuras pueden contener muchos tipos de datos.
2. Dos uniones pueden ser comparadas entre sí para determinar si son iguales F
3. La palabra reservada **typedef** se utiliza para definir nuevos tipos de datos. V
4. Las estructuras no pueden ser comparadas. V
5. Una estructura de *n* elementos siempre ocupa más memoria que *n* variables del mismo tipo que los campos de la estructura. F
6. Con **typedef int Matriz[5][10]** sereservan 50 \* sizeof(int) bytes de memoria. F typedef no reserva memoria

***Ejercicio 3***

Encontrar el error en cada uno de los siguientes puntos:

1. 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); <- cPtr->figura**
2. 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); <- corazones[9].figura**

1. Se declara la estructura **persona**:

**struct persona**

**{**

**char apellido[15];**

**char nombre[15];**

**int edad;**

**}**

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

**al no tener el typedef adelante, c/vez que se define hay poner struct persona m**

1. Indicar cuál de los siguientes fragmentos de código es correcto para recorrer un vector de estructuras definidas según el punto c

| **struct persona v[DIM];**  **// se llena el vector**  **...**  **for (int i=0; i<DIM; i++) {**  **printf("%s\n", (\*v).apellido);**  **v++;**  **}** | **struct persona v[DIM];**  **struct persona \*p = v;**  **// se llena el vector**  **...**  **for (int i=0; i<DIM; i++) {**  **printf("%s\n", (\*p).apellido);**  **p++;**  **}ESTE** |
| --- | --- |

***Ejercicio 4***

1. 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

1. 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 sólo en configuración, gana aquella de figura mayor).

***Ejercicio 5***

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 ( x2 + y2>= r2 ).

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 6***

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 qué 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} } }**

**Ejercicio de Parcial**

***Ejercicio 7***

Se desea escribir una función para resolver una sopa de letras. Una sopa de letras se representa como una matriz de char de FILS filas y COLS columnas, donde FILS y COLS son constantes previamente definidas.

Se cuenta además con un vector ordenado que contiene las palabras a buscar, donde cada elemento es un string constante, y como marca de final contiene el string vacío.

Se cuenta además con las siguientes declaraciones de tipos

typedef enum {DER=0, IZQ, ABA, ARR, I\_AR, I\_AB, D\_AR, D\_AB } Tdireccion;

struct posicion {

char \* palabra;

size\_t fila;

size\_t columna;

Tdireccion direccion;

};

Se pide escribir la función resolverSopa que reciba:

* una matriz de char de FILS x COLS que representa la sopa de letras
* un diccionario ordenado alfabéticamente (vector de strings constantes donde el último elemento es un string vacío)

y retorne un vector de tipo struct posicion que contenga las palabras del diccionario que se encontraron en la sopa de letras, considerando que las palabras pueden estar en forma horizontal (hacia la izquierda o la derecha), vertical (hacia abajo o hacia arriba) **o diagonal**; y donde el último elemento del vector está representado por el valor NULL en el campo palabra.

El vector de respuesta debe ocupar la menor cantidad de memoria posible.

**Si la palabra aparece más de una vez en la sopa, la respuesta debe contener todas las apariciones.**

En caso de error la función debe liberar la memoria reservada y retornar NULL.

Ejemplo:

Dada la siguiente matriz y el diccionario (**se** **asume que está todo en mayúscula**)

| X | X | X | **O** | X | X | X |
| --- | --- | --- | --- | --- | --- | --- |
| **A** | **P** | **A** | **C** | Y | Y | **O** |
| Z | **E** | Z | **R** | Z | **C** | X |
| **E** | **R** | **R** | **O** | **R** | X | X |
| X | **R** | **I** | **O** | **I** | **E** | **X** |
| X | **O** | X | X | **O** | X | X |

diccionario = {"ARRE", "CANCION", "CAPA", "ERROR", "ORCO", "PERRO", "PERTINAZ", "REA", "RIO", ""}

debe retonar

{ {"ARRE", 1, 2, D\_AB}, {"CAPA", 1, 3, IZQ}, {"ERROR", 3, 0, DER}, {"ORCO", 3,3,ARR}, {"ORCO", 4,3,D\_AR}, {"PERRO", 1,1,ABA}, {"REA", 3, 2, I\_AR}, {"RIO", 3,4,ABA}, {"RIO", 4,1,DER}, {NULL, ¿?, ¿?, ¿?}}

***Ejercicio 8***

Se desea implementar una estructura para guardar palabras, donde se sabe que cada palabra ocupa a lo sumo 20 caracteres. Indique para cada una de las opciones propuestas si sirve o no, y en caso de no servir cuál es el error cometido en rojo mal, en verde bien

1. struct dictionary {

char \* words[21]; // vector dinámico de strings de hasta 20 chars

size\_t dim; // cantidad de palabras si tenes menos letras es espacio al pedo

};

1. struct dictionary {

char \* words[]; // vector dinámico de strings

size\_t dim; // cantidad de palabras ahora ni hay espacio para una letra

};

1. struct dictionary {

char \*\* words; // vector dinámico de strings

size\_t dim; // cantidad de palabras bien

};

1. struct dictionary {

char \*\* words; // vector dinámico de strings

// el último elemento es NULL bien

};

1. struct dictionary {

char \*\* words; // vector dinámico de strings

// el último elemento es "" bien

};

1. struct dictionary {

char \* words; // vector dinámico de strings

size\_t dim; // cantidad de palabras mal porque no almacenas palabras

};

1. typedef char Tword[21];

struct dictionary {

Tword \* word; // vector dinámico de strings de hasta 20 chars

size\_t dim; // cantidad de palabras no se puede redimensionar

};

1. typedef char Tword[21];

struct dictionary {

Tword \*\* word; // vector dinámico de strings de hasta 20 chars

size\_t dim; // cantidad de palabras bien

};

| **Ejercicio desafío** |
| --- |

***Ejercicio 9***

Escribir una versión de la macro offsetof. La misma debe funcionar en pampero. No necesariamente deberá funcionar en otras plataformas.