| | |
|---|---|
| **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | Laboratorio de docencia |

# Laboratorios de computación salas A y B

| | |
|---|---|
| *Profesor:* | M.C. JOSE MAURICIO MATAMOROS DE MARIA Y CAMPOS |
| *Asignatura:* | ESTRUCTURA DE DATOS Y ALGORITMOS I |
| *Grupo:* | 12 |
| *No de Práctica(s):* | 04 |
| *Integrante(s):* | ARELLANES CONDE ESTEBAN |
| *No. de Equipo de cómputo empleado:* | 26 |
| *No. de Lista o Brigada:* | 01 |
| *Semestre:* | 2022-2 |
| *Fecha de entrega:* | 14 mar, 23:59 |
| *Observaciones:* | |

CALIFICACIÓN: _____

# Práctica 4.
# Tipos de datos abstractos

Estructuras de Datos y Algoritmos I

Autor: Arellanes Conde Esteban

# Índice



# 1. Objetivo

El alumno aprenderá a crear estructuras en lenguaje C para modelar tipos de datos abstractos y a utilizarlos junto con las estructuras de datos lineales.

# 2. Introducción

Una estructura es una lista de declaraciones entre llaves para cada uno de sus miembros. La palabra clave struct introduce una declaración de tipo estructura (puede ser anónima o no) con un nombre opcional llamado ‘‘`structure tag`’’, o etiqueta de la estructura que puede seguir a la palabra ‘‘`struct`’’. La etiqueta nombrada de este tipo de estructura se puede utilizar posteriormente como abreviatura de la parte de la declaración entre llaves.

# 3. Actividad 1: Estructuras vs tipos

Los programas proporcionados por el profesor llamados ‘‘`add_scomplex`’’ y ‘‘`add_tcomplex`’’ respectivamente suman números complejos y son, en principio, equivalentes. Compilando y ejecutando dichos programas con las líneas de comando:

```
./add_scomplex 1 -1+2i
./add_tcomplex 1 -1+2i
```

Comparamos los resultados obtenidos para los mismos conjuntos de datos y podemos confirmar que efectivamente son iguales aunque sus diferencias radican en el código ya que uno utiliza la línea ‘‘`typedef`’’ al final de la estructura para no repetir el tipo de variable perteneciente a esta, personalmente ambos son igualmente inteligibles en cuanto a formato. [2 puntos]

# 4. Actividad 2: Producto de números complejos

Con base en los programas anteriores desarrollamos un programa que toma como parámetros dos números complejos y devuelve como resultado el producto de los mismos. Compilando y probándolo el programa que se tomó como base podía ser cualquiera de los dos anteriores, pero con la diferencia de multiplicar complejos en lugar de sumarlos y sumar los productos como en una multiplicación normal cambiando: ‘‘`prod.re = a.re * b.re + a.im * b.im`’’ y ‘‘`prod.im = a.re * b.im + a.im * b.re`’’ pudiendo variar el orden ya que la suma es conmutativa. [1 punto]

# 5. Actividad 3: Suma de vectores en 2D

El programa ‘‘`add_v2`’’ calcula la suma de dos vectores de dos dimensiones. Compilando y ejecutando el programa con la línea: `./add_v2 1,0 -1,-1` pudimos analizar que el código en comparación con el de los dos programas anteriores comparten la similtud de usar estructuras para sumar miembros de esta, sólo que uno en una es para números complejos ocupando los miembros para la parte real e imaginaria y en el otro para las componentes suponiendo que todas son reales. [2 puntos]

# 6. Actividad 4: Suma de vectores en 3D

Modificando en una copia el programa del ‘‘`add_v2`’’ nombrandolo como: ‘‘`add_v3.c`’’ para calcular la suma de dos vectores, pero ahora de dimensión tres recibidos por línea de comandos. Compilando y ejecutándolo las modificaciones que se realizaron fueron simplemente agregar un miembro más a la estructura y modificar la función ‘‘`parse`’’ para incluir a la tercera componente. [2 puntos]

# 7. Actividad 5: Suma normalizada de vectores

Modificando una copia del programa anterior nombrado: ‘‘`src/addn_v3.c`’’ para calcular la suma normalizada de dos vectores de dimensión tres recibidos por la línea de comandos implementándose mediante una función con prototipo: ‘‘`void vector3_normalize(vector3 *v);`’’. Compilando, ejecutando y verificando las modificaciones que se realizaron fueron implementar con la biblioteca ‘‘`math.c`’’ la raíz cuadrada para calcular el módulo de manera que la función ‘‘`vector3_normalize`’’ sólo necesita calcular la razón de la suma de las componentes entre el módulo del vector y con eso ya tenemos el vector normalizado. [3 puntos]

# 8. Conclusión y Referencias

## 8.1. Conclusión:

Una declaración de estructura que no va seguida de una lista de variables no reserva memoria, simplemente describe una plantilla, la forma de la estructura. Sin embargo, si la declaración está etiquetada, la etiqueta se puede usar más adelante en las definiciones de instancias de la estructura. Ejemplo de lo anterior lo pudimos observar en los primeros dos programas, dada la declaración ‘‘typedef complex’’ evitando colocar caracteres adicionales.

## 8.2. Referencias:


- Thomas H. Cormen. (2009). Introduction to Algorithms 3e. United States of America: Massachusets Institute of Technology.
- Stephen R. Davis. (2015). Beginning Programming with C++ For Dummies. New Jersey, United States of America: John Wiley  Sons.


**Códigos utilizados en el laboratorio:**

- add_scomplex.c: https://onlinegdb.com/D9WyLbzhu
- add_tcomplex.c: https://onlinegdb.com/Ybb68ldrK
- mul_tcomplex.c: https://onlinegdb.com/yzhN5ui3IP
- add_v2.c: https://onlinegdb.com/kBc4kJB4i
- add_v3.c: https://onlinegdb.com/O8BasApEw
- add_nv3.c: https://onlinegdb.com/G7xFA10OF