
**Universidad de San Carlos de Guatemala**
**Facultad de ingeniería**
**Estructura de datos “C“**

# Proyecto, Fase 3
# USAC Games, Batalla naval
# Manual Tecnico

**Hector Josue Ponsoy Ayala**
**201807220**

# Introducción

Los lenguajes puramente funcionales operan solamente a través de funciones. Una función devuelve un solo valor, dada una lista de parámetros. No se permiten asignaciones globales, llamadas efectos colaterales. Un programa es una llamada de función con parámetros que posiblemente llaman a otras funciones para producir valores de parámetro real.

Las funciones mismas son valores de primera clase que pueden ser pasados a otras funciones, y devueltos como valores funcionales. Así, la programación funcional proporciona la capacidad para que un programa se modifique a sí mismo, es por eso que la parte funcional de este programa fue desarrollada en python, un lenguaje de programación con el cual se pueden crear funciones de manera facíl. Por lo que es posible darle un mejor enfoque a la forma de visualizar las estructuras realizadas

# REQUISITOS DEL PROGRAMA

El proyecto fue creado en lenguaje C++ y Python. Dichos lenguajes fáciles de ejecutar/instalar en casi cualquier sistema operativo.

**Requisitos Generales:** Contar con MAKE instalado en el equipo ya que esto permitirá la ejecución del sistema, también tener instalada cualquier versión de Python, en algunos casos será necesario configurar las variables de entorno según SO.

Requisitos por Sistema Operativo:

- Windows:
  -Windows 7 o superior.
  -RAM 1Gb mínimo
  -Arquitectura x32bits o x64bits

- Linux:
  -Cualquier Distro
  -Ram 1GB mínimo
  -Arquitectura 64 u 86 bits

# Clase ListaAdjacente

## Métodos:

**ingresarCoordenadas():** Metodo al cual se le ingresa el destino1 y el destino2.

**enlazar():** Método que enlaza el destino2 con su respectiva cabecera en la lista adyacente.

**graphvizAdyacente():** Método para generar el .dot de la lista adyacente.

**graphvizGrafo():** Método para generar el .dot del grafo de recorrido.

**sorte():** Método para imprimir los usuarios ordenados en consola.

# Clase HashingCerrado

## Métodos:

**insertar():** Método que recibe un índice y un objeto Producto.

**llenarTabla():** Método para inicializar la tabla con un valor de 0 a 13.

**funcionHash():** Método que retorna el índice en el cual insertar el producto.

**colisiones():** Método que retorna un índice vacío en el cual insertar cuando haya una colisión.

**rehashing():** Método el cual hace que el vector inicial cambie a un tamaño más grande según el 20% del tamaño actual.

**graphvizTablaHash():** Método que genera el .dot de la tabla hash.

# Clase Merkle

## Métodos:

**insertar():** Método que ingresa un dato en un nodo data.

**crearArbol():** Método que inicializa el top de hash con un cero, en lo que se generan los otros nodos.

**Autenticacion():** Método que verifica que el bloque de datos sea de orden 2, si no es de orden 2 lo completa.

**graphvizMerkle():** Método que genera el .dot del árbol de Merkle.

## Clase Bloque

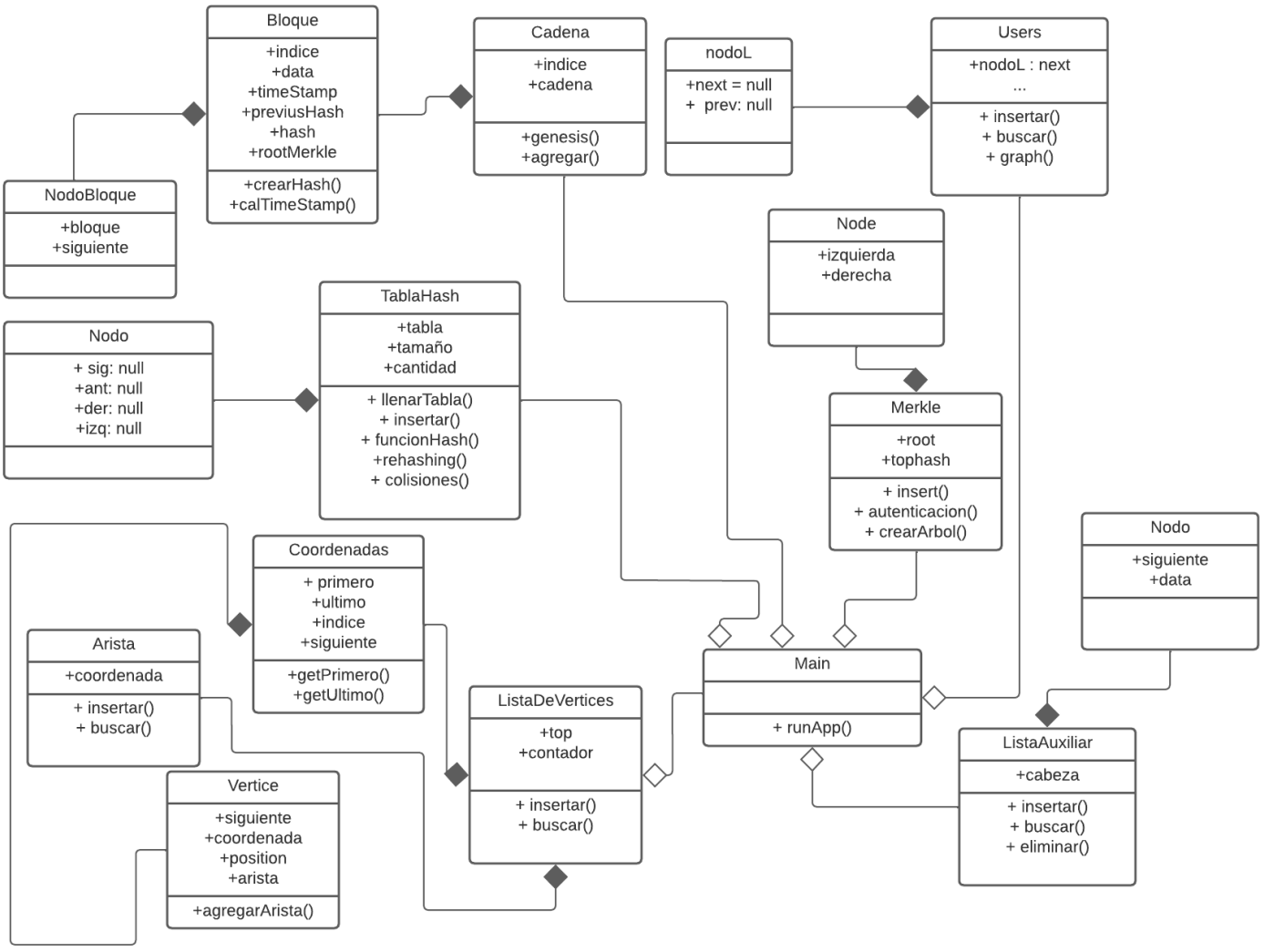
### Métodos:

**genesis():** Método que genera el primer bloque.

**insertar():** Método que crea un nuevo bloque y lo agrega a la data.

**escribirBloque():** Método que escribe un .json con los respectivos bloques.

# Diagrama de clases



## CONCLUSIÓN

Las estructuras de esta fase son catalogadas como “estructuras no-lineales” por el hecho de que la mayoría de estas cuentan con una altura, cuentan con una visualización en 2D como los sería una matriz, una visualización en 3D en todo caso en lo que sería un cubo para lo cual sería la lista de adyacencia en conjunto con el grafo de recorrido