Cierre del semestre

¿Qué fue lo que vimos?

Tipos de datos

En C++, los tipos de datos incluyen int, float, char, etc. Estos definen el tipo y tamaño de los valores que pueden ser almacenados y manipulados en el código.

Errores comunes de C++

Errores como fugas de memoria, desreferenciación de punteros nulos y accesos fuera de límites son comunes. La gestión incorrecta de la memoria. Todos problemas graves

Análisis de complejidad

El análisis de complejidad evalúa el tiempo y espacio que un algoritmo necesita para resolver un problema en función del tamaño de la entrada, vital para la eficiencia del programa.

Team Work

Colaborar en programación competitiva es clave. Comunicación efectiva, división de tareas y apoyo mutuo ayudan a enfrentar desafíos complejos con éxito.

Ordenamiento y clasificación

Técnicas como Quicksort, Mergesort son de gran ayuda para organizar datos en diferentes estructuras.

Bitmask

El uso eficiente de bits para representar conjuntos y manipular información. Útil para problemas que requieren manejo compacto y rápido de conjuntos de elementos.

Apuntadores e Iteradores

Los punteros y los iteradores son herramientas esenciales para acceder y manipular datos en memoria. Facilitan la gestión de estructuras dinámicas como listas y árboles.

Struct

Permite definir tipos de datos personalizados que agrupan diferentes variables bajo un mismo nombre, facilitando la organización y manipulación de datos relacionados.

Map y Hash map

Estructuras de datos que asignan claves a valores. Los mapas mantienen un orden, mientras que los hash maps ofrecen acceso rápido a los valores utilizando funciones hash.

Árboles

Estructuras jerárquicas cruciales en programación competitiva. Árboles binarios, completos entre otros, son necesarios para resolver diversos problemas.

Búsqueda

Técnicas como búsqueda lineal, binaria y búsqueda en árboles son esenciales para encontrar elementos en conjuntos de datos, optimizando el tiempo de ejecución.

Divide y vencerás

Un enfoque algorítmico que divide un problema en subproblemas más simples, resolviéndolos individualmente para luego combinar las soluciones.

Greedy

Un paradigma que elige la opción óptima en cada paso para resolver un problema. Aunque no siempre garantiza la solución óptima global, es útil en muchos casos.

Programación dinámica

Optimización de problemas dividiéndolos en subproblemas más pequeños. Almacenar y reutilizar resultados previos para mejorar la eficiencia en tiempo de ejecución.

¿Qué más puedo hacer?

Aplicaciones web

Lenguajes de programación

1. **JavaScript (JS)**:
    - Esencial para el desarrollo web. Permite la interactividad en el navegador.
    - Ventajas: Amplia comunidad, es el estándar para el desarrollo web.

2. **Python**:
    - Versátil y fácil de aprender. Ideal para la lógica del servidor y aplicaciones complejas.
    - Ventajas: Sintaxis clara, legible y gran cantidad de bibliotecas.

3. **Ruby**:
    - Famoso por Ruby on Rails, un framework MVC para desarrollo rápido..
    - Ventajas: Convenios sobre configuración, desarrollo ágil.

Frameworks para desarrollo web

1. **React**:
   - Biblioteca JavaScript para construir interfaces de usuario interactivas.
   - Ventajas: Virtual DOM, reusable components, rendimiento.

2. **Angular**:
    - Framework de Google para construir aplicaciones web SPA (Single-Page Applications).
    - Ventajas: Potente, amplia comunidad, soporte de Google.

3. **Vue.js**:
    - Framework progresivo para construir interfaces de usuario.
    - Ventajas: Fácil de aprender, integración gradual, rendimiento.

4. **Django**:
    - Framework de Python, siguiendo el principio "batteries included".
    - Ventajas: Productividad, seguridad, ORM potente.

5. **Flask**:
    - Ligero y modular, adecuado para proyectos pequeños a grandes.
    - Ventajas: Flexibilidad, extensibilidad, fácil de comenzar.

Apliciones de escritorio

Lenguajes de programación:

1. **Java**:
    - Potente, multiplataforma y orientado a objetos. Utilizado ampliamente para aplicaciones empresariales.

2. **Python**:
   Versátil y fácil de aprender. Excelente para interfaces de usuario y aplicaciones de escritorio.

3. **C#**:
   Desarrollo en el ecosistema de Microsoft.
   Ampliamente utilizado para aplicaciones Windows.

Frameworks para desarrollo de aplicaciones de escritorio:

1. **Electron**:
   Utiliza HTML, CSS y JavaScript para crear aplicaciones de escritorio multiplataforma.
   Ventajas: Basado en tecnologías web conocidas, permite el desarrollo para Windows, macOS y Linux.

2. **Qt**:
   Framework C++ para crear aplicaciones de escritorio con interfaces gráficas de usuario.
   Ventajas: Robusto, multiplataforma, gran conjunto de herramientas.

3. **GTK**:
   Biblioteca multiplataforma para crear interfaces gráficas de usuario.
   Ventajas: Flexible, soporta múltiples lenguajes de programación como C, C++, Python, etc.

4. **WPF** (Windows Presentation Foundation): Framework de Microsoft para la creación de aplicaciones Windows con interfaces avanzadas. Ventajas: Potente, permite la creación de aplicaciones ricas visualmente.

5. **Swing**:
   Kit de herramientas GUI para Java.
   Ventajas: Integrado en Java, portabilidad, soporte multiplataforma.

Aplicaciones moviles

Desarrollo nativo:

1. **Swift (iOS)**:
   Lenguaje nativo de Apple para el desarrollo de aplicaciones iOS.
   Ventajas: Altamente optimizado para dispositivos Apple, soporte completo de Apple.

2. **Kotlin (Android)**:
   Lenguaje moderno para el desarrollo de aplicaciones Android.
   Ventajas: Oficialmente respaldado por Google, interoperabilidad con Java.

Frameworks multiplataforma:

1. **Flutter**:
    - Framework de Google para crear aplicaciones nativas con un solo código base.
    - Ventajas: Alto rendimiento, interfaces de usuario personalizables, Hot Reload.

2. **React Native**:
    - Framework de Facebook para crear aplicaciones móviles multiplataforma con JavaScript y React.
    - Ventajas: Reutilización de código, gran ecosistema de librerías.

3. **Xamarin**:
    - Plataforma de Microsoft para desarrollar aplicaciones móviles con C# y .NET.
    - Ventajas: Herramientas integradas, acceso a las APIs de dispositivos.

4. **Ionic**:
    - Framework de desarrollo de aplicaciones móviles híbridas utilizando HTML, CSS y JavaScript.
    - Ventajas: Fácil acceso para desarrolladores web, soporte multiplataforma.

ICPC

Drivers

1. **C**:
   Es el lenguaje más utilizado para escribir drivers.
   Ofrece un alto nivel de control sobre el hardware y es portátil entre diferentes arquitecturas de hardware.

2. **C++**:
   Similar a C, pero con características de programación orientada a objetos. A menudo se utiliza para crear drivers que requieren estructuras más complejas.

3. **Ensamblador**:
   Aunque menos común hoy en día debido a la complejidad y a la dificultad para su mantenimiento, el ensamblador se usa a veces para programar ciertas partes críticas de los drivers donde se necesita un control extremadamente preciso del hardware

4. **Rust**:
   Las características de Rust, como su sistema de tipos que garantiza la ausencia de errores de memoria y su capacidad para administrar la concurrencia de manera segura, hacen que sea atractivo para el desarrollo de drivers y aplicaciones de sistemas de tiempo real donde la seguridad y la fiabilidad son críticas.