Universidad Autónoma de Baja California

Facultad de Ingeniería Arquitectura y Diseño

Ingeniero en computación

Materia: Programación Estructurada / Clave **36276**

Alumno: Carlos Antonio Alvarez Angulo

Matrícula: 366182

Maestro: Pedro Núñez Yépiz

Actividad No. : 12

Tema - Unidad : Archivos.

Ensenada Baja California a 09 de Noviembre del 2023

Universidad Autónoma de Baja California

Facultad de Ingeniería Arquitectura y Diseño

1. INTRODUCCIÓN

En el ámbito de la programación en C, el desarrollo de sistemas de gestión de datos es fundamental para comprender conceptos clave como manipulación de archivos, estructuras de datos y algoritmos de búsqueda y ordenación. En esta práctica, se ha implementado un sistema de gestión de registros de alumnos que aborda diversas competencias, desde la carga inicial de datos hasta la generación de informes ordenados y la gestión eficiente de archivos.

2. COMPETENCIA

La competencia principal desarrollada en esta práctica es la habilidad para programar en C y gestionar archivos de manera efectiva. Se espera que el estudiante demuestre conocimientos sólidos en la manipulación de estructuras de datos, algoritmos de búsqueda y ordenación, así como la implementación adecuada de funciones que respondan a los requisitos específicos del sistema.

3. FUNDAMENTOS

Librerías en C y Creación de Librerías Propias

Las librerías en C son componentes esenciales de la programación en C. Estas librerías son colecciones de funciones predefinidas y estructuras de datos que proporcionan herramientas comunes y reutilizables para los programadores. La ventaja principal de las librerías es la capacidad de reutilizar código, lo que ahorra tiempo y reduce errores al utilizar soluciones probadas.

- Reutilización de Código: Las librerías permiten a los programadores reutilizar funciones y estructuras de datos en diferentes proyectos, lo que promueve la modularidad y la eficiencia.
- Librerías Estándar: El lenguaje C incluye librerías estándar como <stdio.h>, <stdlib.h>, <string.h>, entre otras, que proporcionan funciones esenciales para tareas comunes como entrada/salida, gestión de memoria y manipulación de cadenas de caracteres.
- Inclusión de Librerías: Los programadores pueden incluir librerías en sus programas mediante la directiva #include, lo que les da acceso a las funciones y estructuras definidas en esas librerías.

Creación de Librerías Propias:

- Personalización: Los programadores pueden crear sus propias librerías personalizadas para encapsular y reutilizar código específico de dominio.
- Modularidad: Las librerías propias permiten dividir un proyecto en módulos más pequeños y gestionables, lo que facilita el mantenimiento y la colaboración en el desarrollo de software.
- Compilación Separada: Las librerías propias se pueden compilar por separado y vincular con otros programas, lo que facilita la gestión de proyectos complejos.

Crear librerías propias implica diseñar y desarrollar un conjunto de funciones y estructuras de datos coherentes que resuelvan problemas específicos. La creación de librerías personalizadas es una práctica común para promover la reutilización de código y la modularidad en el desarrollo de software en C.

Métodos de Ordenación

Los métodos de ordenación son algoritmos diseñados para organizar datos de manera sistemática. Ordenar datos es una tarea común en la programación, ya que permite acceder y buscar información de manera más eficiente. Los algoritmos de ordenación trabajan reorganizando los elementos de un conjunto de datos en un orden específico, como ascendente o descendente. Los métodos de ordenación pueden variar en complejidad y eficiencia, y la elección del método adecuado depende de factores como el tamaño de los datos y la disponibilidad de recursos. Comprender estos métodos es esencial para lograr un rendimiento óptimo en aplicaciones y para abordar problemas que requieren una búsqueda rápida y organizada de datos.

- **Método de Burbuja** (Bubble Sort): Este algoritmo compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto. Repite este proceso hasta que todo el conjunto de datos esté ordenado. Es sencillo pero no eficiente para conjuntos de datos grandes.
- **Método de Selección** (Selection Sort): En este método, se selecciona repetidamente el elemento mínimo del conjunto de datos y se coloca en la posición correcta. Es simple pero no es eficiente para grandes conjuntos de datos.
- **Método de Inserción** (Insertion Sort): Este algoritmo construye una lista ordenada uno por uno, tomando elementos no ordenados e insertándolos en la posición correcta. Es eficiente para pequeños conjuntos de datos.
- **Método de Fusión** (Merge Sort): Utiliza un enfoque divide y conquista para ordenar datos. Divide el conjunto en mitades, las ordena por separado y luego combina las mitades ordenadas. Es eficiente y estable.
- **Método Rápido** (Quick Sort): También utiliza la técnica divide y conquista. Elige un elemento como pivote, coloca elementos más pequeños a su izquierda y elementos más grandes a su derecha. Luego, ordena las dos particiones. Es eficiente y ampliamente utilizado.
- **Método de Árbol Binario de Búsqueda** (Binary Tree Sort): Los elementos se insertan en un árbol binario de búsqueda y se recuperan en orden mediante un recorrido inorden del árbol. Es eficiente y útil en estructuras de árbol.
- **Método de Ordenación por Radix** (Radix Sort): Este algoritmo ordena elementos basándose en sus dígitos, comenzando por el dígito menos significativo y avanzando hacia la izquierda. Es eficiente para ordenar números enteros.
- **Método de Conteo (Counting Sort)**: Se utiliza para ordenar números enteros en un rango conocido. Cuenta la frecuencia de cada elemento y coloca los elementos en su posición correcta. Es eficiente para datos con un rango limitado.
- **Método de Cubetas (Bucket Sort)**: Divide el rango de valores en "cubetas" o "buckets" y luego ordena cada cubeta individualmente. Es eficiente para datos distribuidos uniformemente.

- **Búsqueda Secuencial** (Sequential Search): También conocida como búsqueda lineal, este método busca un elemento recorriendo secuencialmente todo el conjunto de datos hasta encontrar una coincidencia. Es efectivo en cualquier tipo de conjunto de datos, pero no es eficiente para conjuntos de datos grandes.
- **Búsqueda Binaria** (Binary Search): Este método solo se aplica a conjuntos de datos ordenados. Divide el conjunto en dos mitades y compara el elemento buscado con el valor medio. Luego, reduce la búsqueda a la mitad relevante. Es eficiente y adecuado para grandes conjuntos de datos ordenados.
- **Búsqueda Hash** (Hash Search): Utiliza una función hash para mapear claves a ubicaciones de almacenamiento. Permite una búsqueda rápida en estructuras de datos como tablas hash. La eficiencia depende de la función de hash.
- **Búsqueda por Interpolación** (Interpolation Search): Este método se utiliza en conjuntos de datos ordenados y utiliza una estimación interpolada para buscar el elemento. Puede ser más eficiente que la búsqueda binaria en algunos casos.
- **Búsqueda por Salto** (Jump Search): Similar a la búsqueda binaria, pero da "saltos" en lugar de dividir el conjunto en dos partes iguales. Puede ser más eficiente que la búsqueda binaria en conjuntos de datos grandes.
- **Búsqueda Exponencial** (Exponential Search): Comienza con saltos pequeños y duplica el tamaño en cada paso hasta que el rango se encuentre. Luego, realiza una búsqueda binaria dentro de ese rango. Es eficiente en conjuntos de datos no necesariamente ordenados.
- **Búsqueda de Texto** (String Search): Utilizado para encontrar patrones en cadenas de caracteres. Algoritmos conocidos incluyen el algoritmo de Boyer-Moore y el algoritmo de Knuth-Morris-Pratt.
- **Búsqueda en Árboles** (Tree Search): Se utiliza en estructuras de árboles, como árboles binarios de búsqueda y árboles AVL, para localizar elementos específicos dentro de la estructura del árbol.
- **Búsqueda en Grafos** (Graph Search): Se aplica a estructuras de datos de grafo para encontrar elementos en relaciones de nodos y bordes.

Archivos de texto

En programación en C, la manipulación de archivos de texto es una habilidad esencial para realizar operaciones de lectura y escritura de datos. Aquí te presento una breve teoría acerca de cómo trabajar con archivos de texto en C.

Apertura de Archivos:
La función principal para abrir archivos en C es fopen(). Esta función toma dos argumentos: el nombre del archivo y el modo de apertura.

```
FILE *archivo = fopen("nombre_archivo.txt", "modo_apertura");
```

Los modos de apertura más comunes son:
"r": Lectura (Read).
"w": Escritura (Write). Si el archivo no existe, se crea; si existe, se sobrescribe.
"a": Añadir (Append). Escribe al final del archivo sin borrar su contenido.
"r+": Lectura y escritura.
"w+": Lectura y escritura, sobrescribe el archivo existente o crea uno nuevo.
"a+": Lectura y escritura, añade al final del archivo.

Cierre de Archivos:
Después de realizar operaciones en un archivo, es esencial cerrarlo correctamente para liberar recursos y evitar problemas. Se utiliza la función fclose().

```
fclose(archivo);
```

Escritura en Archivos:
Para escribir en un archivo, se utilizan las funciones fprintf() o fputc().

```
fprintf(archivo, "Hola mundo cruel y despiadado!\n");
```

Lectura de Archivos:
Para leer desde un archivo, se usan funciones como fscanf() o fgetc().

```
char buffer[50];
fscanf(archivo, "%s", buffer);
```

EOF (End-of-File):
Cuando se lee un archivo, se verifica el indicador de fin de archivo (EOF) para determinar si se ha llegado al final del archivo.

```
while (!feof(archivo)) {
   // Realizar operaciones de lectura
}
```

Manipulación de Posición en el Archivo:
La posición actual en un archivo puede manipularse mediante fseek() y ftell().

fseek(archivo, offset, SEEK_SET); // Moverse al offset desde el inicio
long posicion = ftell(archivo); // Obtener la posición actual

La manipulación de archivos en C es una habilidad clave para el desarrollo de aplicaciones que requieren entrada/salida persistente de datos. A medida que avanzas, podrías explorar conceptos más avanzados como la manipulación binaria de archivos, lectura y escritura de registros, entre otros.

4. PROCEDIMIENTO

INSTRUCCIONES:
1.- Realiza un programa en C que utilice librerías propias
2.- Realiza reporte de práctica
3.- Sube a Blackboard, programa, librería, y reporte de practica y PDF anexo con capturas y código

5. RESULTADOS Y CONCLUSIONES

Los conceptos de librerías en C, métodos de ordenación y métodos de búsqueda son fundamentales en programación e informática. Las librerías permiten la reutilización de código, los métodos de ordenación organizan datos eficientemente y los métodos de búsqueda localizan información rápidamente. Comprender y aplicar estos conceptos es esencial para el desarrollo de software eficiente y resolución efectiva de problemas relacionados con datos.

6. ANEXOS

https://github.com/CarlosAlvarez777/Programacion_Estructurada_23-2/tree/CAAA_PE_ACT12

7. REFERENCIAS

Librerías en C:
Programarya. (s.f.). Bibliotecas o Librerías en C++. Recuperado el 9 de noviembre de 2023, de https://www.programarya.com/Cursos/C++/Bibliotecas-o-Librerias
Métodos de Ordenación:
Architecnologia. (s.f.). Programación en C: Algoritmos de ordenamiento, búsqueda y apuntadores. Recuperado el 9 de noviembre de 2023, de https://architecnologia.es/programacion-c-algoritmos-ordenamiento-busqueda-apuntadores
Métodos de Búsqueda:
Architecnologia. (s.f.). Programación en C: Algoritmos de ordenamiento, búsqueda y apuntadores. Recuperado el 9 de noviembre de 2023, de https://architecnologia.es/programacion-c-algoritmos-ordenamiento-busqueda-apuntadores
Archivos de texto:
Fernández-Valdivia, J. (s.f.). Estructuras de Datos. Capítulo 9: Listas doblemente enlazadas. Recuperado el 9 de noviembre de 2023, de https://ccia.ugr.es/~jfv/ed1/c/cdrom/cap9/f_cap94.htm