UNIVERSIDAD AUTONOMA METROPOLITANA

Programación Avanzada

Práctica 5:

"Algoritmos de Ordenamiento y Búsqueda: Iterativos y Recursivos"

Fecha de Elaboración:
Martes, 22 de julio del 2008

Fecha de Entrega:
Jueves, 31 de julio del 2008

Contenido

AVISOS:

- No se reciben reportes de las prácticas después de la fecha de entrega.
- En la página se encuentra el formato del reporte para las prácticas en donde se describen cada uno de los apartados que deberá considerarse. Se deberá seguir dicha plantilla para el desarrollo del reporte de las prácticas.
- Enviar a la cuenta de correo un paquete zipeado que contendrá:
  - Códigos fuentes (extensión **.c**).
  - Códigos Objetos (extensión **.o**)
  - Ejecutables (sin extensión)
  - En el reporte incluir por actividad (extensión **.doc)**:
    - **Desarrollo** de la actividad.
    - **Resultados** de la actividad.
- En el subject del correo escribir: ***Práctica 5 - <nombre completo>***.
- **Se entregará el reporte los códigos fuentes impresos.**

1 Objetivos

1) Utilizar el diseño de programación ascendente y descendente en el enfoque de **programación modular** (enfoque "divide y vencerás") para la resolución de problemas mediante el uso de **módulos** que serán traducidos a funciones (y/o procedimientos) en el lenguaje C.
2) Utilizar mecanismos de **recursividad** para el desarrollo de algoritmos que permitan su aplicación.
3) Hacer uso de **archivos de texto** para la extracción de datos con los cuales se trabajarán.
4) Utilizar funciones que permitan la **asignación dinámica a memoria** de un tipo de dato básico.
5) Definir los **algoritmos de búsqueda** y **algoritmos de ordenamiento recursivos** para trabajar con arreglos.
6) Reutilizar los **algoritmos de búsqueda** y **algoritmos de ordenamiento iterativos** para trabajar con arreglos.
7) Evaluar el **desempeño de los algoritmos** de ordenamiento y búsqueda de acuerdo al tiempo de ejecución del programa, utilizando las funciones de la librería **time.h** y el comando de Linux **time**.

2 Introducción

En el desarrollo de esta práctica se requiere que el alumno realice cada una de las actividades que se le indica.

Definir los algoritmos de búsqueda y ordenamiento iterativos que se vieron en clase.

Reutilización de los algoritmos de búsqueda y ordenamiento para datos básicos almacenados en un arreglo.

Reutilización de funciones que permiten la extracción de datos de un archivo de texto.

2.1 Complementos

Para completar el reporte de la práctica el alumno deberá de efectuar una breve investigación de los siguientes puntos (máximo 3 párrafos por punto):

- Utilización de funciones para la **asignación dinámica de memoria**:
  - malloc ( ).
  - calloc ( ).
  - free ( ).
- El orden de complejidad de los distintos algoritmos, tanto en su versión iterativa y recursiva, de ordenamiento y búsqueda.

2.2 Ayuda complementaria

- Referencia de la bibliografía contenida en la planeación del curso de Programación Avanzada.
- Notas del Curso de Introducción a al Programación 08-I.
- Guía Básica para el uso de C.
- Practicas anteriores en donde se definen las funciones que se desea reutilizar.

3 Desarrollo

La sección del desarrollo de la práctica se divide en una serie de actividades que permiten englobar, desarrollar y alcanzar cada uno de los objetivos de la práctica. Es necesario leer con atención cada una de las actividades a desarrollar.

3.1 Antecedentes de la asignación dinámica de memoria

3.1.1 Función para asignación dinámica de memoria: *calloc ( )*.

Reserva espacio en memoria para a un arreglo de elementos de un tamaño especifico de datos. El espacio será inicializado con valores por default de acuerdo al tipo de dato de la declaración.

void *calloc (size_t nobjeto, size_t tamanio)

La función **calloc ( )** regresa un apuntador (puntero) al espacio reservado para un arreglo de ***nobjeto***, con el tamaño de cada dato del arreglo especificado en el parámetro ***tamanio***.

Ejemplo:

```
arr = (int *)calloc(tamanio , sizeof(int) );
```

3.1.2 Función para asignación dinámica de memoria: *malloc ( )*.

Reserva espacio en memoria para a un elemento de un tamaño especifico de dato. El valor **No** se inicializa por default.

void *malloc (size_t tamanioDato)

La función **malloc ( )** regresa un apuntador (puntero) al espacio reservado para un objeto de tamaño ***tamanioDato***.

Se puede obtener un espacio reservado para la utilizaron de un arreglo, de elementos de un tipo de datos, de la siguiente manera:

void *malloc (num_elementos * tipo_dato)

Ejemplo:

```
dato = (int *)malloc(sizeof(int) );
arr = (int *)malloc(tamanio *sizeof(int) );
```

3.1.3 Función para liberar la memoria asignada: *free ( )*.

Libera el espacio de memoria previamente asignado por las funciones ***calloc( )*** o ***malloc( )***. El parámetro es el apuntador al espacio de memoria que se desea liberar.

void free (void *apuntador)

Ejemplo:

```
free (dato)
free(arr)
```

3.2 Actividad 1 “Asignación Dinámica y Lectura de Datos de un Archivo de Texto”

Con el desarrollo de esta actividad se alcanzará los objetivos 1,3 y 4 de la práctica.

Analice el código proporcionado para el desarrollo de la actividad y complete el código que hace falta en las siguientes funciones:

int *AsignacionDinaminca(int tamanio)
void LiberarMemoria(int *arreglo)
void LeerDatosArchivo(FILE *Arch, int *arreglo, int *longitud)

3.2.1 Diseño Modular

Desarrolle el diagrama modular de acuerdo al código proporcionado y al desarrollo de los módulos que hacen falta.

3.2.2 Validación de la Asignación Dinámica

Verifique el funcionamiento del programa con el archivo proporcionado: **Archivo20.txt**.

Actividad 2 “Implementación de los Algoritmos de Ordenamiento y Búsqueda: Iterativo y Recursivo”

Con el desarrollo de esta actividad se alcanzará los objetivos del 1 al 6 de la práctica.

Reutilice el código fuente desarrollado en la actividad 1 y complete el código proporcionado de acuerdo a las siguientes especificaciones de las siguientes funciones

OrdenarIterBurbuja (arreglo, longitud);
OrdenarIterSeleccion (arreglo, longitud);
OrdenarIterInserccion (arreglo, longitud);
OrdenarRecQuicksort (arreglo, 0, longitud);
OrdenarRecMergesort(arreglo, 0, longitud);
BusquedaIterSecuencial(arreglo, longitud, elemento);
BusquedaIterBinario(arreglo, longitud, elemento);
BusquedaRecBinario(arreglo, 0, longitud, elemento);

No olvide incluir las funciones que sean necesarias para los algoritmos.

3.2.3 Diseño Modular

Desarrolle el diagrama modular de acuerdo al código proporcionado y al desarrollo de los módulos que hacen falta.

3.2.4 Validación de Algoritmos Recursivos

Verifique el funcionamiento de los algoritmos recursivos con el archivo proporcionado: **Archivo20.txt**.

3.3 Actividad 3 "Comparación del Desempeño de los Algoritmos de Ordenamiento y Búsqueda"

Con el desarrollo de esta actividad se alcanzará el objetivo 7 de la práctica.

Ejecute los algoritmos para los siguientes archivos:

Valores Ascendentes: ArchivoAsc1.txt Número de datos: _____
Valores Descendentes: ArchivoDesc1.txt Número de datos: _____
Valores Aleatorios: ArchivoAlea1.txt Número de datos: _____

Valores Ascendentes: ArchivoAsc2.txt Número de datos: _____
Valores Descendentes: ArchivoDes2.txt Número de datos: _____
Valores Aleatorios: ArchivoAlea2.txt Número de datos: _____

Tabla de Tiempo de ejecución (_____ datos)

	Ascendente	Descendente	Aleatorio
Ordenamiento Iterativo			
Burbuja	<<time.h / time>>		
Selección			
Inserción			
Ordenamiento Recursivo			
Quicksort			
Mergesort			
Búsqueda Iterativo			
Secuencial			
Binario			
Búsqueda Recursivo			
Binaria Recursiva			

Tabla de Tiempo de ejecución (_____ datos)

	Ascendente	Descendente	Aleatorio
Ordenamiento Iterativo			
Burbuja	<<time.h / time>>		
Selección			
Inserción			
Ordenamiento Recursivo			
Quicksort			
Mergesort			
Búsqueda Iterativo			
Secuencial			
Binario			
Búsqueda Recursivo			
Binaria Recursiva			