UNIVERSIDAD AUTONOMA METROPOLITANA

Programación Avanzada

Práctica 2:

"Algoritmos iterativos de Ordenamiento"

(Comparación de desempeño)

Fecha de Elaboración:
Martes, 1 de julio del 2008

Fecha de Entrega:
Martes, 8 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 2 - <nombre completo>***.

1 Objetivos

1) Utilizar **comandos** dentro del entorno de Linux para facilitar las fases de codificación, compilación, ejecución y pruebas, en el desarrollo de programas.
2) 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.
3) Utilizar el paso de **parámetros** al programa utilizando **gcc**.
4) Utilizar funciones que permiten determinar el **tiempo de ejecución** de un programa.
5) Implementar los **algoritmos de ordenamiento**: "Burbuja", "Inserción Directa" y "Selección Directa".
6) Integrar un mecanismo de medición de tiempo para determinar el **desempeño de los algoritmos** con distintos datos, con el objetivo de determinar las diferencia del tiempo de respuesta de cada uno de los algoritmos de ordenamiento.

2 Introducción

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

En las actividades referentes al entorno Linux, el alumno ingresara en la línea de comandos algunas instrucciones y observará los resultados generados.

La implementación de los algoritmos de ordenamiento se requiere la traducción de los algoritmos de pseudocódigo a código en C.

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):

- Definir el término de orden de complejidad.
- Investigar el orden de complejidad de los algoritmos de iterativos de ordenamiento.
- Las estructuras y funciones que se definen en la librería **time.h**.
- Apertura, lectura, escritura y cierre, de archivos de texto y binarios.

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.

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 Actividad 1 "Comandos en el Entorno de Linux (2)"

Con el desarrollo de esta actividad se alcanzará el objetivo 1 de la práctica.

Para el desarrollo de esta actividad siga en orden los siguientes pasos, verifique que los resultados son los mismos que se mencionan a continuación:

1) Entrar al sistema escribiendo el nombre de usuario (login) y contraseña (password).
2) Abrir una ventana de "shell" (Terminal) para practicar varios de los comandos de línea permitidos por el sistema.

3) Practicar los siguientes comandos y observe los resultados:

- **who**
  Muestra los usuarios conectados al sistema.

```
{~}[gustavob@tenampak]$ who
        gustavob pts/0        2008-06-24 15:17 (148.206.46.26)
```

- **whoami**
  Muestra el usuario actual.

```
{~}[gustavob@tenampak]$ whoami
        gustavob
```

- **passwd**
  Permite cambiar el password. Requiere el password actual.

```
{~}[gustavob@tenampak]$ passwd
        Changing password for gustavob
        (current) UNIX password:
        Enter new UNIX password:
        Retype new UNIX password:
```

- **ssh**
  Es utilizado para conectarse a otra maquina.
  Dentro del laboratorio las computadoras tienen el seudónimo de quetzal#, es por ello que se puede conectarse a otra computadora con el siguiente comando:

```
{~}[gustavob@tenampak]$ ssh quetzal17
The authenticity of host 'quetzal17 (192.168.1.17)' can't be established.
RSA key fingerprint is 2f:82:4c:fe:c2:5a:8f:9f:1e:6e:7a:94:b5:c1:64:73.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'quetzal17,192.168.1.17' (RSA) to the list
of known hosts.
```

```
gustavob@quetzal17's password:
{~}[gustavob@quetzal17]$
{~}[gustavob@quetzal17]$ w
 15:45:23 up  4:24,  3 users,  load average: 0.02, 0.04, 0.03
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
aze      tty7     :0               15:17    3.00s 19.14s  0.16s x-session-manag
aze      pts/0    :0.0             15:23    3.00s  0.18s  0.18s bash
gustavob pts/1    192.168.1.251    15:42    0.00s  0.16s  0.00s w
```

- **rmdir**

  Permite cambiar borrar una carpeta o subdirectorio. La condición para que se efectué el borrado es que este subdirectorio debe de estar vació.

```
{~}[gustavob@tenampak]$ ls
        Examples  hola.txt  Practica1

{~}[gustavob@tenampak]$ mkdir practicaN

{~}[gustavob@tenampak]$ ls
        Examples  hola.txt  Practica1  practicaN

{~}[gustavob@tenampak]$ rmdir practicaN/

{~}[gustavob@tenampak]$ ls
        Examples  hola.txt  Practica1
```

Para los siguientes comando se requiere que se tengan creados algunos archivos creados: *.txt, *.c, *.o y (un ejecutable sin extensión).

Ejemplo:

```
{~/Practica1}[gustavob@tenampak]$ ls
        ConverTemperatura.c  ConverTemperatura.o  ConvTemp  Hi
        Hola_gcc.c  Hola_gcc.o  hola.txt
```

- **more**

  Muestra el contenido de un archivo de texto, muestra el contenido pantalla a pantalla. Para salir presiones “q”.

```
{~/Practica1}[gustavob@tenampak]$ more ConverTemperatura.c
        /*
          Archivo: ConverTemperatura.c
          Descripción: Este programa efectúa la conversión de
        temperaturas de Celsius a Fahrenheit. La lectura de la t
        emperatura se efectúa desde teclado.
            La salida del resultado se efectúa a consola.
            Objetivo:
              Utilización de funciones con el paso de parametros
          Autor: Basurto Páez Gustavo
          Fecha: 15 de abril de 2008
        */
```

- **head**

  Visualiza las primeras ‘n’ líneas del archivo.

```
{~/Practica1}[gustavob@tenampak]$ head ConverTemperatura.c
        /*
```

Archivo: ConverTemperatura.c

```
{~/Practica1}[gustavob@tenampak]$ head -20 ConverTemperatura.c
/*
  Archivo: ConverTemperatura.c
  Descripción: Este programa efectúa la conversión de temperaturas de Celsius a Fahrenheit. La lectura de la temperatura se efectúa desde teclado.
     La salida del resultado se efectúa a consola.
     Objetivo:
      Utilización de funciones con el paso de parametros
  Autor: Basurto Páez Gustavo
  Fecha: 15 de abril de 2008
*/

/*Incluir librerías utilizadas en el programas*/
#include <stdio.h>

//Definicion de Constantes


/*Prototipo de Funciones*/
float FuncConversion (float parametro);
```

- **tail**

  Visualiza las últimas ‘n’ líneas del archivo.

```
{~/Practica1}[gustavob@tenampak]$ tail ConverTemperatura.c
    //Calculo de la temperatura
    tempF = (9.0/5)*parametro + 32;
    //NOTA: si la división no es flotante genera otro resultado

    //modificacion de la variable Global
    ResultadoGlobal = tempF;

    return ( tempF);
}

{~/Practica1}[gustavob@tenampak]$ tail -15 ConverTemperatura.c
float FuncConversion (float parametro)
{
    //Declaracion de variables locales
    float tempF =0.0;

    //Calculo de la temperatura
    tempF = (9.0/5)*parametro + 32;
    //NOTA: si la división no es flotante genera otro resultado

    //modificacion de la variable Global
    ResultadoGlobal = tempF;

    return ( tempF);
}
```

- **file**

  Proporciona información acerca del tipo de archivo.

{~/Practica1}[gustavob@tenampak]$ **file** hola.**txt**
hola.txt: ASCII text

{~/Practica1}[gustavob@tenampak]$ **file** ConvTemp
ConvTemp: ELF 32-bit LSB **executable**, Intel 80386, version 1 (SYSV), for GNU/Linux 2.6.8, dynamically linked (uses shared libs), for GNU/Linux 2.6.8, not stripped

{~/Practica1}[gustavob@tenampak]$ **file** ConverTemperatura.**c**
ConverTemperatura.c: ISO-8859 **C program text**, with CRLF line terminators

{~/Practica1}[gustavob@tenampak]$ **file** ConverTemperatura.**o**
ConverTemperatura.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped

- **time**
  Este comando permite obtener el tiempo de ejecución de las aplicaciones. Presenta un resumen del ***tiempo real***, el tiempo del ***uso de CPU*** por el ***usuario***, y el ***tiempo de uso de CPU*** por parte del ***sistema***.

{~/Practica2}[gustavob@quetzal21]$ time ./Act1_B 100

real 0m0.004s
user 0m0.000s
sys 0m0.000s
{~/Practica1}[gustavob@tenampak]$

- **claer**
  Permite limpiar el contenido de la pantalla.

{~/Practica2}[gustavob@quetzal21]$ **clear**

3.2 Actividad 2 “Paso de parámetros y medición de tiempo de ejecución de un programa.”

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

3.2.1 Paso de parámetros desde la línea de comandos

Algunos programas requieren que se envíen una serie de datos que permitan la ejecución de los mismos. Estos datos son pasados como parámetros en el momento de ejecución de los mismos.

Primero se invoca la ejecución del programa se deja un espacio y después la lista de parámetros, cada parámetro es separado por un espacio.

{~/Practica1}[gustavob@quetzal20]$ ./ConvTemp **1 2 3 4 5**

Dentro del programa es necesario especificar que se reciben parámetros desde la línea de comandos. En la función **main** se definen dos parámetros:

int **main**(**int argc, char *argv[]**)

- **int argc** : determina el numero de argumentos que se ingresaron al ejecutar el programa.
- **char *argv[]**: es un apuntador a cadenas (matriz de cadenas), que almacenan cada una uno de los parámetros listados como una cadena.

{~/Practica2}[gustavob@quetzal20]$ ./Act1 **1 2 3 4 5 hola**

Número de Parámetros: 7
El primer parámetro es el nombre del programa ejecutable.
Argv [0] : **./Act1**
Los siguientes son los parámetros que se reciben como cadenas:
Argv [1] : **1**
Argv [2] : **2**
Argv [3] : **3**
Argv [4] : **4**
Argv [5] : **5**
Argv [6] : **hola**

3.2.2 Funciones de la librería "time"

En la introducción se pide que se efectué una investigación sobre la librería **time.h**, sin embargo aquí se explicara a grande rasgos algunas funciones que se utilizaran para el desarrollo de está actividad.

La estructura de dato **time_t** define un tipo de dato que representa el tiempo como un entero largo.

time_t inicial, final;

La función **time()** permite obtener el tiempo en un formato condesado, el valor de regreso es un tipo de dato **time_t**. al pasar el parámetro de Null se indica que tome el tiempo actual.

inicial = **time(NULL)**;

La función **difftime(timepof, timepoi)** calcula la diferencia de tiempos, del tiempo final (tiempof) con respecto al tiempo inicial (timepoi). El valor de regreso es un double que representa los segundos de diferencia.

Tejecucion = **difftime(final, inicial);**
printf("\n\t El tiempo total es: %**lf**", Tejecucion);

3.2.3 Conversión de cadena a un tipo de dato numérico.

Existen funciones en la librería **stdlib.h** que permiten la conversión de cadenas a valores entero, entero largo o flotante.

La función **atol()** recibe como argumento una cadena, la cual será transformada a entero largo.

ValorEnteroLargo = **atol**("10000");

La función **atoi()** recibe como argumento una cadena, la cual será transformada a entero.

ValorEntero = **atoi**(“115”);

La función **atof()** recibe como argumento una cadena, la cual será transformada a entero largo.

ValorFlotante = **atof**(“100.15”);

3.2.4 Problema 1 “Paso de parámetros al programa”

Desarrollar un programa que reciba argumentos al momento de invocar su ejecución. El programa deberá de ser capas de saber el número de parámetros son, leerlos y escribirlos a pantalla.

3.2.5 Problema 2 “Medición de tiempo de ejecución con funciones de la librería time.h y con el comando time de Linux”

Modificar el programa obtenido en el problema 1 creando una función que reciba como parámetros el primer argumento como un valor entero. Y dentro de la función efectué lo siguiente:

1. Tome el tiempo inicial de ejecución del cuerpo de la función.
2. Defina dos ciclos for anidados como condición de paro sea menor al valor recibido como parámetro.
3. Tome el tiempo final de la ejecución.
4. Calcule la diferencia de tiempo.
5. Imprima el resultado del tiempo de ejecución.

Al ejecutar el programa utilice también el comando de Linux para comparar resultados.

3.3 Actividad 3 “Implementación de Algoritmos Iterativos de Ordenamiento”

Con el desarrollo de esta actividad se alcanzará los objetivos 2 y 5 de la práctica.

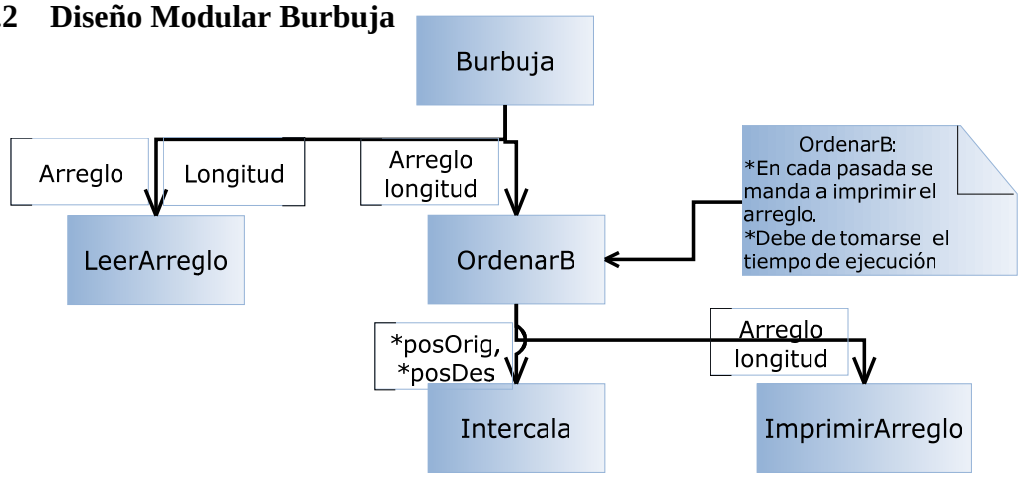
3.3.1 Burbuja

Efectué la implementación del algoritmo de ordenamiento Burbuja (**Burbuja.c**). Siguiendo el siguiente diagrama de flujo.

Entrada: Numero de elementos a ingresar y los números enteros que se almacenarán en un arreglo.

Salida: El arreglo ordenado paso por paso hasta obtener el arreglo ordenado con el método de burbuja y el tiempo de ejecución.

3.3.2 Diseño Modular Burbuja



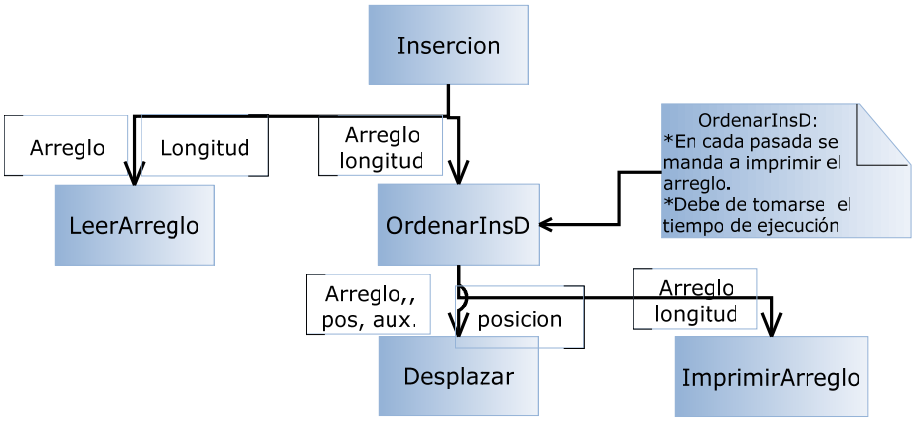
3.3.3 Inserción Directa

Efectué la implementación del algoritmo de ordenamiento Inserción Directa (**Insercion.c**). Siguiendo el siguiente diagrama de flujo.

Entrada: Numero de elementos a ingresar y los números enteros que se almacenarán en un arreglo.

Salida: El arreglo ordenado paso por paso hasta obtener el arreglo ordenado con el método de inserción directa y el tiempo de ejecución.

3.3.4 Diseño Modular Inserción Directa



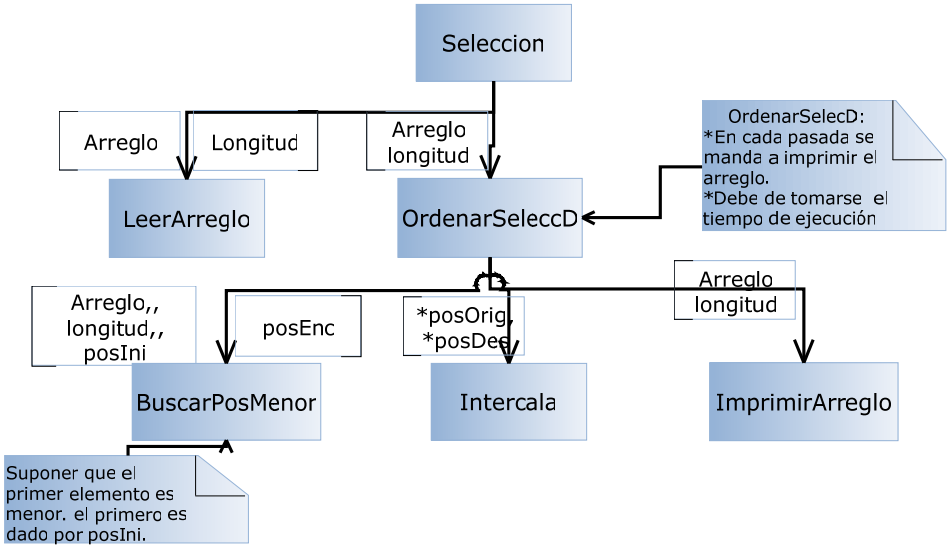
3.3.5 Selección Directa

Efectué la implementación del algoritmo de ordenamiento Selección Directa (**Directa.c**). Siguiendo el siguiente diagrama de flujo.

Entrada: Numero de elementos a ingresar y los números enteros que se almacenarán en un arreglo.

Salida: El arreglo ordenado paso por paso hasta obtener el arreglo ordenado con el método de selección directa y el tiempo de ejecución.

3.3.6 Diseño Modular Selección Directa



3.3.7 Datos de Prueba de los Programas de Ordenamiento.

Orden ascendente: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,14, 15, 16, 17, 18, 19, 20.
Orden descendente: 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1.
Pseudoaleatorio: 3, 10, 9, 1, 8, 2, 7, 5, 4, 6, 20, 11, 19, 12, 18, 13, 17, 14, 15,16.

3.3.8 Reporte de la actividad.

Los siguientes puntos se deben de considerar dentro de código fuente para obtenerlos como resultado de la ejecución del programa:

- La impresión de paso a paso de cómo se va ordenando el arreglo (incluyendo el número de pasada).
- Tiempo de ejecución con librería time.h y el comando time de Linux.

Con los resultados obtenidos para cada algoritmo respecto a un tipo de dato de prueba, efectué una tabla comparativa, en donde incluya sus conclusiones.

Datos de Prueba \ Algoritmo	**Burbuja**	**Inserción**	**Selección**
Ascendente	<#pasadas>, <tiempo>	10, 15.2 seg	
Descendente			
Pseudoaleatorio			

Tabla 1.- Comparativa de los algoritmos de ordenamiento iterativo.