

Universidad de Buenos Aires
Facultad de Ingeniería

75.41 – Algoritmos y Programación II

Cátedra Ing. Patricia Calvo

1º Cuatrimestre 2010

Trabajo Práctico

Administrador de Archivos

Versión 1.1

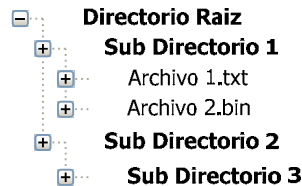
Índice

<i>Índice</i>	<i>2</i>
<i>1. Enunciado.....</i>	<i>3</i>
<i>1.1 – Etapa de Consultas.....</i>	<i>5</i>
<i>1.2 – Presentación de Resultados.....</i>	<i>6</i>
<i>2 - Objetivos del TP</i>	<i>8</i>
<i>Apéndice A</i>	<i>9</i>
<i>Apéndice B</i>	<i>12</i>

1. Enunciado

El trabajo práctico consiste en modelar una aplicación para la “Administración de Archivos”. La aplicación deberá administrar tanto a los archivos, como a sus contenedores que son los directorios.

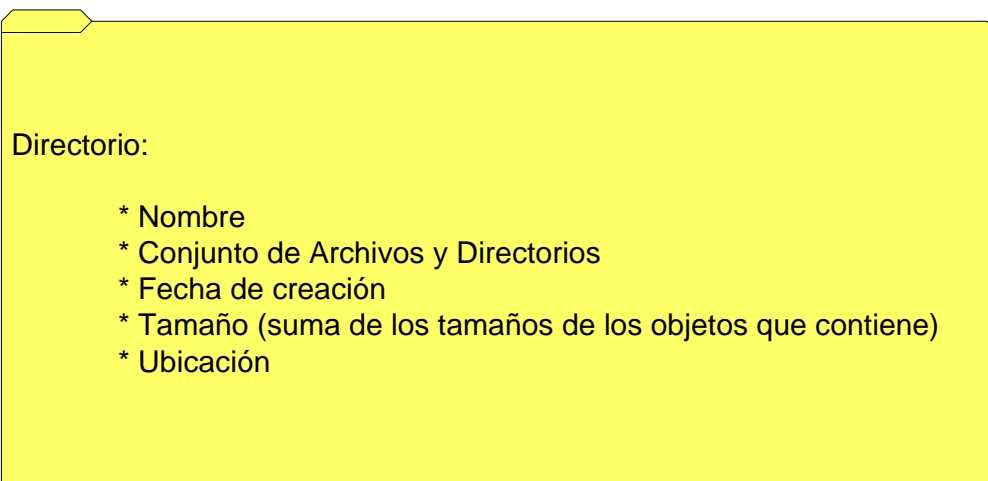
En la siguiente imagen se muestra la estructura que se puede formar mediante archivos y directorios:



Uno de los parámetros recibidos por la aplicación es el directorio a administrar. En una primera etapa la aplicación deberá recorrer este directorio y armar la estructura diseñada por el grupo para poder procesar las consultas de la segunda etapa.

Para poder armar la estructura, vamos a definir las propiedades de los directorios y archivos.

Un directorio tiene las siguientes propiedades:



El nombre del directorio es un texto que identifica el directorio dentro del contexto en que esta, es decir, no puede haber otro directorio o archivo con el mismo nombre dentro del directorio padre. Por ejemplo dentro del “Directorio Raíz” solo puede haber un directorio o archivo llamado “Sub Directorio 1”.

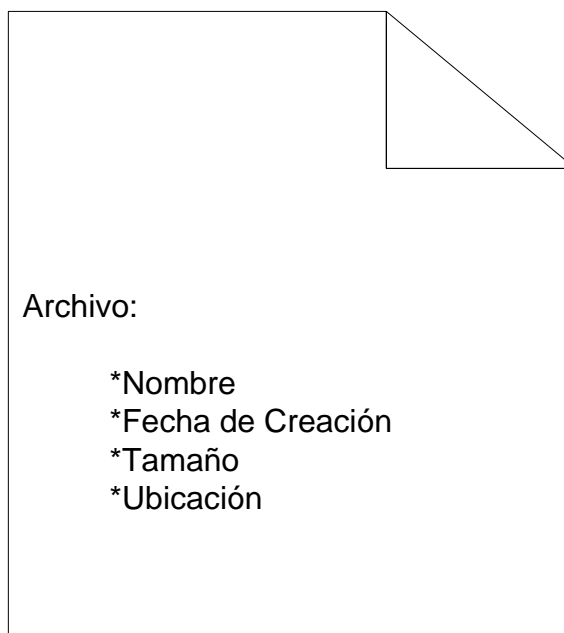
El conjunto de archivos y directorios que este directorio contiene, que es la principal funcionalidad de un directorio, son los archivos y directorios que están ubicados dentro de él.

La fecha de creación es un dato que tienen los directorios para informar la fecha de creación de los mismos.

El tamaño de un directorio se calcula como la suma de los tamaños de los archivos y directorios que están en el conjunto de archivos y directorios.

La ubicación es la concatenación de los nombres de todos los subdirectorios que lo contienen hasta llegar al directorio pasado por parámetro a la aplicación separados por el carácter "/". Por ejemplo si queremos calcular la ubicación de "Sub directorio 3", sería: "Directorio Raiz/Sub Directorio 2/Sub Directorio 3".

Los archivos tienen las siguientes propiedades:



El nombre del archivo es un texto que identifica el archivo dentro del contexto en que está, es decir, no puede haber otro directorio o archivo con el mismo nombre dentro del directorio padre. Por ejemplo dentro del "Sub Directorio 1" solo puede haber un directorio o archivo llamado "Archivo 1.txt".

La fecha de creación es un dato que tienen los archivos para informar la fecha de creación de los mismos.

El tamaño de un archivo es un atributo que se debe leer del archivo físico en el sistema operativo.

La ubicación es la concatenación de los nombres de todos los subdirectorios que lo contienen hasta llegar al directorio pasado por parámetro a la aplicación separados por el carácter "/". Por ejemplo si queremos calcular la ubicación de "Archivo 1.txt", sería: "Directorio Raiz/Sub Directorio 1/Archivo 1.txt".

Toda esta información se deberá almacenar en la estructura diseñada por el grupo, para poder resolver las consultas realizadas en la etapa 2.

1.1 – Etapa de Consultas

El fin de la aplicación es poder realizar consultas sobre los atributos de los archivos y directorios contenidos en el directorio pasado por parámetro.

El objetivo es poder buscar archivos y directorios a través de su nombre refinando la búsqueda por tamaño, es decir que se puede fijar un tamaño máximo, o uno mínimo o uno máximo y uno mínimo para la búsqueda que se esta ejecutando.

Para la búsqueda de texto se podrá utilizar la palabra exacta, por ejemplo "Sub Directorio 1" y eso devolverá el resultado del directorio, se podrá utilizar el carácter "?", que reemplaza un carácter comodín, por ejemplo si se busca "Sub ?irectorio 1" devolverá "Sub Directorio 1", pero si se busca "Sub Directorio ?" devolverá los 3 resultados.

Otro carácter comodín es el "*", que reemplaza un conjunto de caracteres cualesquiera, por ejemplo si se busca "*rio 1", devolverá "Sub Directorio 1", pero si se busca "*1" devolverá este directorio y el archivo. Del mismo modo si la búsqueda es "*", devolverá toda la estructura de resultado.

1.2 – Presentación de Resultados

La elaboración de la presentación no es la finalidad del TP, solo se exige que los resultados de las corridas muestren la información completa por pantalla y en un archivo de texto de manera entendible.

Por ejemplo:

Directorio Raíz: "Directorio pasado por parámetro".

Texto buscado: "Texto que se ingreso a buscar".

Condiciones: "si hay condiciones sobre el tamaño de los archivos".

Resultado de directorios:

Índice	Nombre	Cantidad de sub Elementos	Fecha	Ubicación
1	Dir 1	10	10-01-10	ruta
2	Dir 2	1	10-01-10	ruta
3	Dir 3	15	10-01-10	ruta

Resultado de Archivos:

Índice	Nombre	Tamaño	Fecha	Ubicación
1	Arch 1	10Mb	10-01-10	ruta
2	Arch 2	1Mb	10-01-10	ruta

2 - Conclusiones

Escribir las conclusiones del trabajo práctico, explicando las complicaciones encontradas y la solución que se tomo.

2 - Objetivos del TP

Primera Parte:

- 1) Diseñar la solución al problema planteado.
- 2) Realizar un diagrama con la solución y las relaciones entre las clases.
- 3) Adjuntar la división de tareas por cada integrante del grupo y tareas comunes.
- 4) Implementar la solución al problema en C++ utilizando la clase Vector Dinámico y Lista.
- 5) Comparar los tiempos de búsqueda (Utilizando la clase Cronómetro) entre la corrida con la clase Vector Dinámico y Lista.
- 6) Adjuntar la bibliografía y sitios de Internet consultados.

Segunda Parte:

- 7) Diseñar la clase Árbol Eneario.
- 8) Implementar la clase Árbol.
- 9) Utilizar la estructura Árbol en la estructura del TP.
- 10) Adjuntar la división de tareas por cada integrante del grupo y tareas comunes.
- 11)
- 12) Implementar el algoritmo Búsqueda Binaria para la clase Vector Dinámico y comparar los tiempos con el Árbol Eneario.
- 13) Adjuntar la bibliografía y sitios de Internet consultados.

Clase Cronómetro

La clase cronometro mide los tiempos que transcurren en determinados intervalos (Cuento los ciclos del procesador, para pasar a segundos, deberán dividir el valor por los ciclos por segundo del procesador).

Las secuencias de medición pueden ser:

a)
Constructor();
Parar();

Acumula el tiempo transcurrido entre que se ejecuto el constructor y se llamo el método parar.

b)
Iniciar();
Parar();

Acumula el tiempo transcurrido entre que se ejecuto el método iniciar y se llamo el método parar.

c)
Iniciar();
Pausar();
Continuar();
Parar();

Acumula el tiempo transcurrido entre que se ejecuto el método iniciar y se llamo el método parar, más el tiempo transcurrido en la llamada al método continuar y la segunda llamada al tiempo parar.

A continuación se brinda la clase Cronometro para dar una posible forma de implementación, la misma puede ser modificada y corregida según el criterio del grupo.

```

/*****
* Algoritmos y Programación II - 75.41 *
* Cátedra Ing. Patricia Calvo *
* Facultad de Ingeniería - Universidad de Buenos Aires *
*****/
/* TDA Cronometro
* Archivo : Cronometro.h
* Versión : 1.0
*/
#ifdef __CRONOMETRO_H__
#define __CRONOMETRO_H__
#include <time>
#include <sstream>
/*****
/* Definiciones de Tipos de Datos */
/*-----*/
/* Definición del TDA Cronometro */
class Cronometro{
private:
/*****
/* Definicion de Atributos */
/*-----*/
/* El atributo inicio guarda el tiempo en que se inicia el cronometro*/
clock_t inicio;
/* El atributo cronometro guarda la suma de los intervalos de tiempo entre que se
inicia el cronometro y se pausa, o se inicia y se detiene*/
long contador;
/* El atributo pausado guarda el estado actual del cronometro, si esta pausado
o no*/
bool pausado;
/*****
/* Definicion de Primitivas */
/*-----*/
/*
pre : ninguna.
post: Crea un Cronometro inicializado en cero.
*/
public:
Cronometro(){
iniciar();
}
/*
pre : el cronometro debe haber sido creado con el constructor.
post: Borra todos los tiempos acumulados y estable el instante en que se
comienza a contar el tiempo.
*/
void iniciar(){
contador = 0;
inicio = clock();
this->pausado = false;
}
/*
pre : el cronometro debe haber sido creado con el constructor.
post: Acumula el tiempo transcurrido desde la creacion, inicio o continuar
(lo ultimo que haya pasado).
*/
void pausar(){
clock_t fin;
fin = clock();
contador += fin - inicio;
this->pausado = true;
}
/*
pre : el cronometro debe haber sido creado con el constructor.
post: Estable el instante a partir del cual se cuenta el tiempo.
*/
void continuar(){
if (this->pausado){
inicio = clock();
this->pausado = false;
}
}
}

```

```

    }

/*
pre : El cronometro debe haber sido creado con el constructor.
post: Finaliza la cuenta y acumula todos los tiempos transcurridos.
*/
void parar(){
    if (!this->pausado){
        this->pausar();
        this->pausado = true;
    }
}

/*
pre : El cronometro debe haber sido creado con el constructor.
post: Devuelve una leyenda con los milisegundos que estan acumulados.
*/
std::string toString(){
    std::stringstream convertidor;
    convertidor << this->contador;
    return "Transcurrieron " + convertidor.str() + " milisegundos";
}

/*
pre : El cronometro debe haber sido creado con el constructor.
post: Devuelve la cantidad de milisegundos que estan acumulados.
*/
long getTiempoTranscurrido(){
    return this->contador;
}
};
#endif /* __CRONOMETRO_H__ */

```

Parámetros de una aplicación

La función **main()** recibe parámetros con alguno de los siguiente encabezados:

- `int main(int argc, char* argv[])`
- `int main(int argc, char ** argv)`
- `int main(int argCount, char * const argv[])`

Todas las formas de arriba son válidas, y todas ellas definen dos parámetros, pero nosotros utilizaremos el primero. Esos parámetros contienen información perteneciente a la reconstrucción de los argumentos de la línea de comandos pasados dentro del programa desde el proceso padre (una consola de línea de comandos, un administrador de ventanas, etc.) El ejemplo es un programa *main* simple que imprime sus argumentos de línea de comando.

Ejemplo

Ejemplo: `src/main/programa/programa.cpp`

```
int main (int argc, char* argv[]) {  
    for (int i=0; i<argc; ++i) {  
        cout << "argv# " << i << " es " << argv[i] << endl;  
    }  
    return 0;  
}
```

argv es un array bidimensional de cadenas de texto. **argc** es la dimensión de **argv**. **argv** contiene en cada posición el texto (el carácter espacio delimita una nueva posición) que se envió desde la línea de comando.

int main() "devuelve" un entero, que debería ser 0 si todo ha ido bien, o un código de error distinto de cero si algo salió mal.

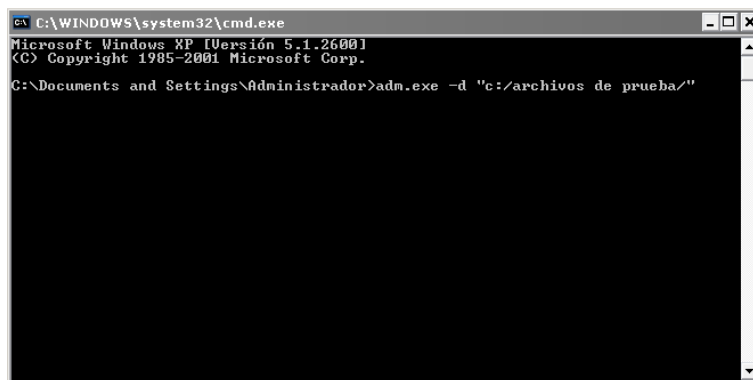
Si corremos este programa con argumentos de línea de comandos, veremos algo como esto en la salida:

```
programa>programa.exe spam huevos "guerras espaciales" 123  
argv# 0 es /src/main/programa/programa.exe  
argv# 1 es spam
```

argv# 2 es huevos
argv# 3 es guerras espaciales
argv# 4 es 123

El primer argumento es el nombre de la ruta del ejecutable, Los otros argumentos son tomados desde la línea de comandos como cadenas separadas por espacios o tabulaciones. Para pasar una cadena que contenga espacios en un argumento simple, debes encerrar la cadena entre comillas.

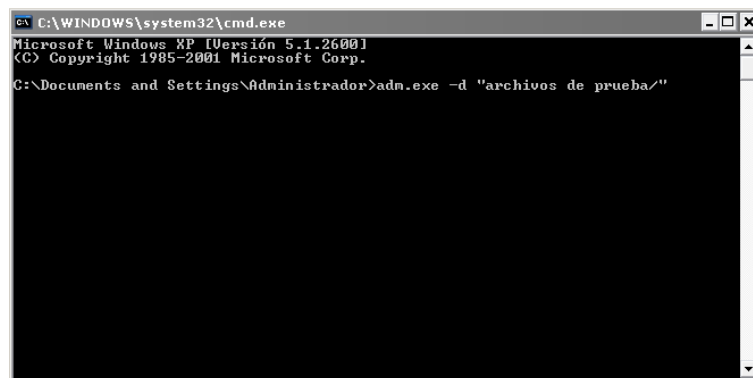
La aplicación del TP se ejecutara de la línea de comandos de la siguiente manera (El ejemplo esta en Windows, pero de forma similar se hará en Linux):



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Versión 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrador>adm.exe -d "c:/archivos de prueba/"
```

Donde adm.exe es el nombre de la aplicación, y el parámetro -d indica que el siguiente parámetro es el directorio con el cual se desea trabajar. La ruta puede ser relativa o no a donde se encuentra el exe, como ser:



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Versión 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrador>adm.exe -d "archivos de prueba/"
```