

**Tecnológico de Costa Rica**

Escuela de Ingeniería en Computación

Carrera de Ingeniería de Computación, Plan 410.

Tarea Programada:

Simulación de un Sistema de Administración de Archivos

IC-1803 Introducción a la Programación

Profesor:

Juan Gómez Pereira

Estudiante:

Álvaro Castro Venegas

Sebastían Salas García

Josué Jiménez Alfaro

San José, 21 de Abril del 2016.

**Contenido**

[Interfaz Gráfica 3](#_Toc447476970)

[Implementación de Clases 5](#_Toc447476971)

[Distribución del Trabajo 13](#_Toc447476972)

[Cronograma 15](#_Toc447476973)

[Resultados Obtenidos 16](#_Toc447476974)

[Conclusiones 17](#_Toc447476975)

# Interfaz Gráfica

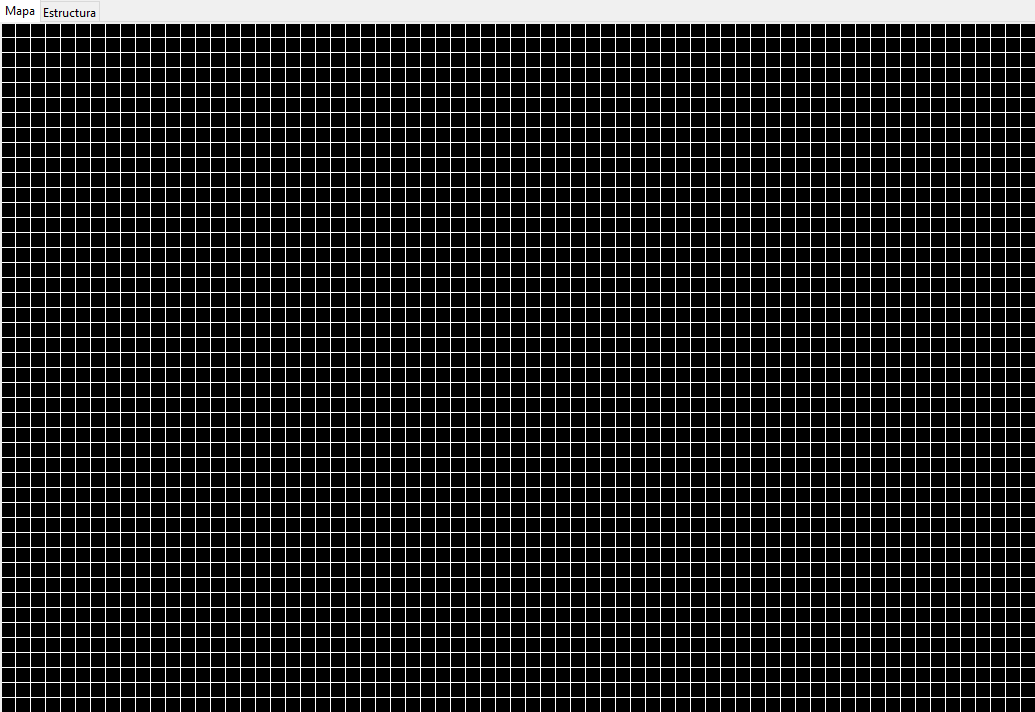
Línea de Comandos

Los comandos se escriben en un cuadro de texto implementado por Tkinter. En el cual se leen las líneas por medio de índices en el sistema, los cuales se especifican cada vez q se escribe un comando y se ejecuta.



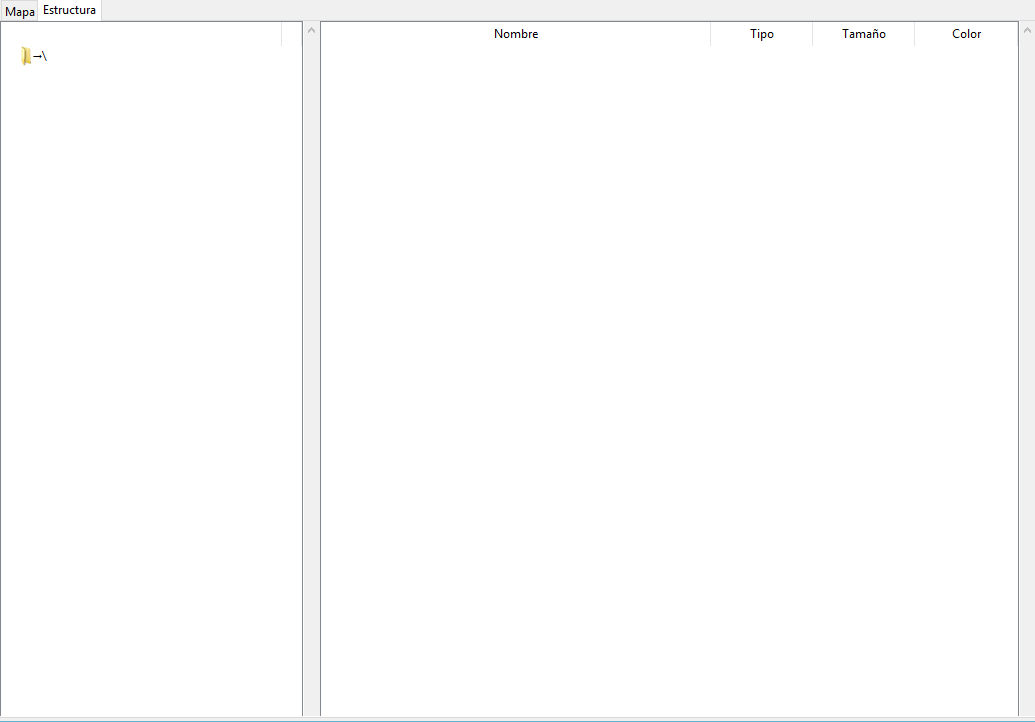
Mapa

El mapa es la representación de la memoria, loa cual se presenta como una cuadricula dibujada sobre un canvas, en el cual se verá la estructura de los espacios disponibles en memoria.

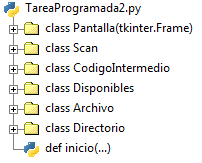


Estructura

La estructura se representa por medio de Treeview los cuales agregan los diferentes datos que posea el directorio o la representación gráfica de los comandos dados.

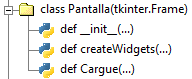


# Implementación de Clases



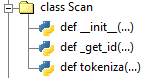
Con el objetivo de que el programa estuviera bien moldeado y estructurado se llevaron a cabo una serie de clases para poder crear el sistema eficientemente. A continuación, se presentaran una tabla de clases y métodos que se fueron creados a lo largo del programa para realizar la simulación del administrador de archivos.

## Clase Pantalla



La clase de Pantalla tiene como función moldear toda la parte de la interfaz gráfica. El constructor de la clase recibe como parámetros la memoria disponible inicializada, el tipo de interfaz que en este caso utilizamos es Tkinter y por último el nombre del directorio raíz de nuestro programa. El método createWidgets construye todos los botones, tiras y listas que se utilizarán en la pantalla principal. Al crear todos los objetos de la pantalla se inicia con el método cargue quien es donde se maneja el programa, lee la instrucción y la manda a escanear con la clase Scan que se procederá a explicar.

## Clase Scan



La clase Scan va a ser quien va a tratar la instrucción digitada por el usuario. Esta va a recibir como parámetros línea (instrucción), memoria (base de datos de nuestro administrador), treeview (mostrador de directorios y archivos), padre (raíz), lista de los directorios existentes y la lista de los archivos existentes. El método tokeniza descompondrá la instrucción carácter por carácter y gracias al método get\_id podemos ir validad que se está leyendo y poder obtener los datos necesarios. A continuación, se mostrará los diferentes mandatos que podrá usar el usuario para usar nuestro sistema.

|  |  |  |
| --- | --- | --- |
| Mandato | Estructura | Efecto |
| Formatear | Format nnn | Todos los bloques del espacio de almacenamiento son disponibles y existe un único directorio (el directorio raíz).  El espacio total disponible de bloques es de nnn.  Nota: suponga que inicialmente se parte de un espacio de almacenamiento de 200 bloques. Esto podría ser variado en cualquier momento con el mandato format. |
| Listar | dir | Despliega una lista de los archivos existentes en el directorio actual. |
| Crear Archivo | file nombre\_archivo tam color | Crea un archivo en el directorio actual con nombre nombre\_archivo, el cual va a utilizar tam bloques y el color que utilizará en el mapa de uso de espacio. Los bloques deberán solicitarse de la lista de bloques disponibles. Los nombres deben ser secuencias de letras. Tam corresponde a un valor entero. Debe verificarse que haya espacio disponible!! Ejemplo: create texto 20 #32C18F |

|  |  |  |
| --- | --- | --- |
| Listar información de un archivo | dir id | Verifica que el archivo o directorio exista utilizando el camino hasta ese directorio, ej: \nombreDir. Si es un directorio indica la cantidad de archivos y subdirectorios que tiene ese directorio. Si es un archivo indica el color del archivo, número de bloques y los bloques que posee (recuerde que el orden es importante). Para que el despliegue de la lista de bloques no se haga tan larga utilice rangos de la forma bloque\_inicial – bloque\_final. |
| Expandir un archivo | expand nombre\_archivo tam | Solicita tam bloques de la lista de bloques disponibles y los agrega al archivo indicado. El archivo debe existir. También el espacio disponible debe ser suficiente. Si cualquiera de estas condiciones no se cumple reporta error. Ejemplo: expand texto 10 |
| Borrar un archivo o directorio | del nombre\_archivo  \nombre\_directorio | Borrar el archivo indicado, el cual debe existir. Si no reporta error. Los bloques que utilizaba ese archivo se devuelven a la lista de bloques disponibles. Ejemplo: del texto Borra el directorio indicado (incluye todo lo que hay bajo él), el cual debe existir en el directorio actual. Si no reporta error. |
| Crear directorio | md nombre\_directorio | Crea el directorio indicado bajo el directorio actual. Si bajo el directorio actual existe un archivo o directorio con el mismo nombre reporta error. |
| Cambiar el directorio actual | cd \nombre\_directorio | El directorio indicado puede ser:   * “.” Se refiere al directorio actual * “..” Se refiere al directorio padre del directorio actual * “\” Directorio raíz * “Id” corresponde a un identificador   Se verifica que el directorio destino exista. Si no existe reporta error. “cd” Reporta el directorio actual. |
| Remover directorio | rd \nombre\_directorio | Elimina el directorio indicado y todos sus subdirectorios. El directorio se especifica igual que en el mandato cd.  Notas:  a. No puede remover el directorio actual, por lo tanto “rd“, es inválido.  b. “rd \” borra todos los archivos y subdirectorios existentes, pero el directorio raíz se mantiene. |

## Clase CodigoIntermedio



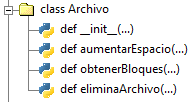
Esta clase recibe los mismo parámetros que Scan, solo que este lo que hace es que una vez que se escanea la instrucción y esta es correcta se manda a la clase de código intermedio donde mediante la clase verificar entra al módulo del mandato para que se mande a ejecutar.

## Clase Disponible

## 

La clase disponible permite mantener la lista de los bloques que se encuentran valga la redundancia disponibles. El constructor va a recibir la cantidad total de bloques disponible y mantiene una lista de los números de bloques disponibles. Es importante hacer énfasis en que el orden sí importa. El procedimiento agregarEspacio recibe el número de bloques solicitados y retorna una lista de bloques, esto los toma del inicio de la lista de bloques disponibles. Devolver va a recibir una lista de bloques y los incluye al final de la lista de bloques disponibles (se elimina un archivo y se liberan los espacios donde se encontraban). Reiniciar lo que hace es abilitar todos los bloques como tal.

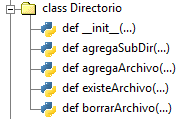
## Clase Archivo



La clase archivo va a mantener la información de cada archivo, esta va a tener los siguientes atributos: nombre de archivo, el tamaño, lista de los bloques asignados, color del bloque y el directorio en que se encuentra el archivo. El procedimiento aumentarEspacio recibe una lista de bloques adicionales los cuales se incluirán en la lista de bloques asignados al archivo (vienen de los bloques disponibles).

El método obtenerBloques va a retorna la lista de bloques asignados al archivo que se ocupe conocer y eliminarAchivo como dice su nombre saca de la lista de archivo y del treeview el archivo a eliminar.

## Clase Directorio

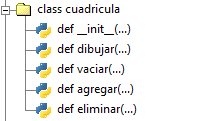


Esta clase mantendrá los archivos que han sido creados por el usuario; cuenta con los atributos: cantidad de archivos, lista de archivos y la referencia al directorio padre. El constructor va permite crear un directorio vacío bajo un directorio. El procedimiento agregarArchivo recibe como parámetro una instancia de la clase archivo y la incluye en la lista de archivos. ; existeArchivo recibe como parámetro el nombre de un archivo y va retorna True si el mismo existe. Por último, el método borrarArchivo borrará el archivo de la lista de archivos.

## Clase cuadrados

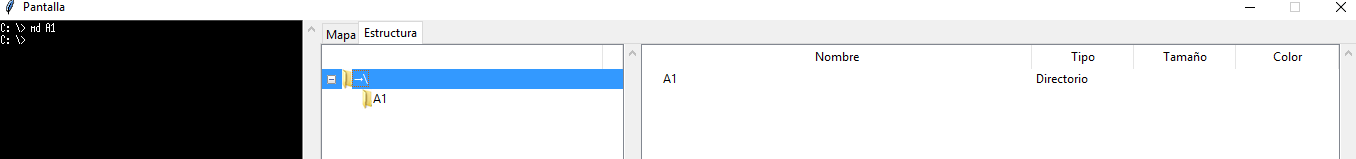
La clase cuadrados lo que permite es crear cuadrados de cierto tamaño en el canvas. Contiene los siguientes parámetros: nombre del cuadrado, color, posiciones en el eje donde va a ir el cuadrado y el canvas sobre el cual se va a dibujar.

## Clase cuadricula

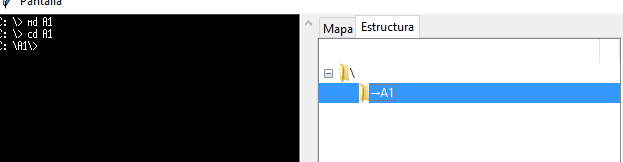
Clase que permite crear una caudricula en el canvas y además cambiar,expandir y eliminar individualmente o toda la cuadricula. Sus métodos son: dibujar,vaciar,agregar y elminar. El constructor lo único que hace es recibir la cantidad de bloques se desean hacer y el canvas donde se dibuja, dibujar crea la cuadricula utilizando esos dos parámetros, vaciar borra toda la cuadricula, agregar le anade espacio a algún archivo y elminar elimina los bloques de un archivo en especifico.

# Demostracion

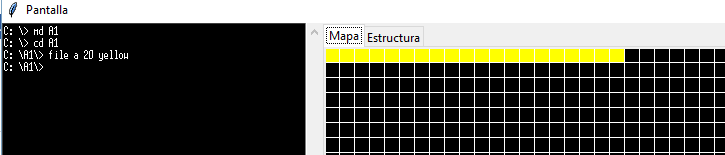
Se crea un directorio

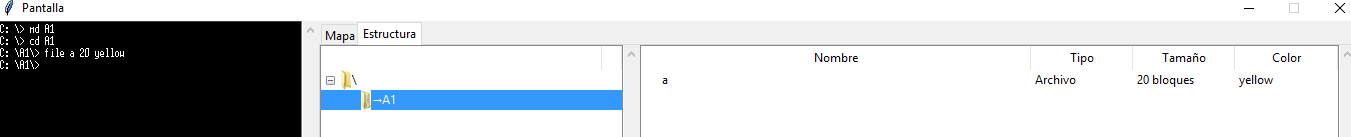


Se navega a ese directorio

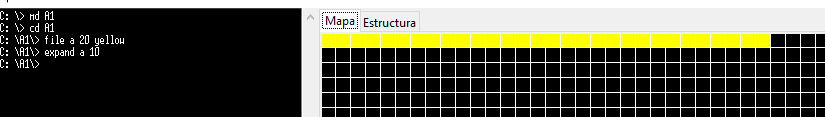


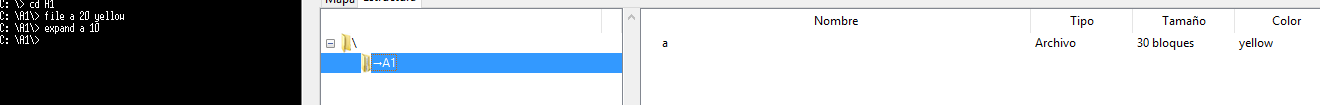
Se crea un archivo en el nuevo directorio



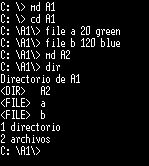


Se expande el tamaño del archivo

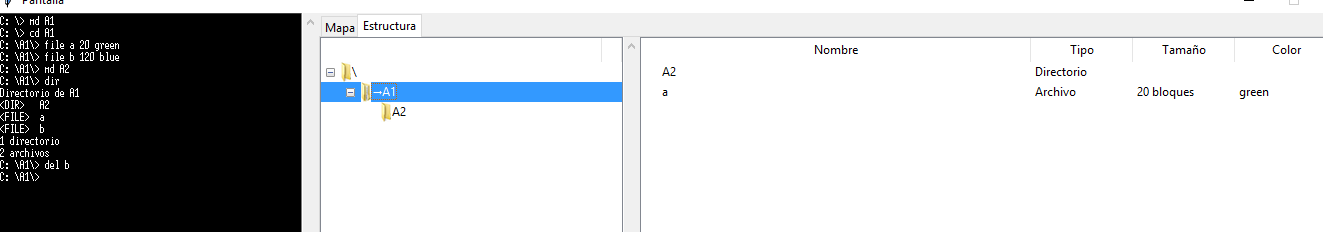




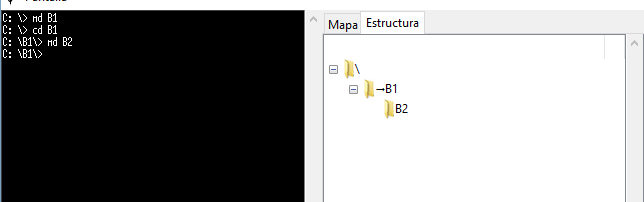
Se crea un directorio, juntos con varios archivos y otro directorio dentro de el y se verifica que es lo que tiene el directorio padre

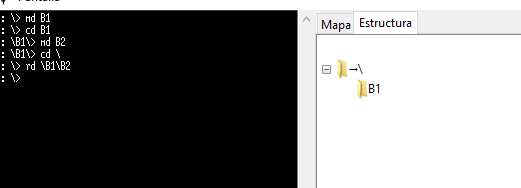


Se borra el archivo b



Se borra un directorio utilizando el camino donde esta





# Distribucion del trabajo

Con el fin de llevar a cabo exitosamente el proyecto de simulación de un administrador de archivos se distribuyeron tareas específicas para poder ir desarrollándolo y es importante aclarar que mayor parte del trabajo nos reuníamos y trabajamos todos sobre el mismo proyecto por lo que al final se realiza un trabajo en conjunto.

|  |  |  |
| --- | --- | --- |
| Persona | Parte que realizó | Clases que elaboró |
| Álvaro Castro Venegas | Memoria | * Disponible * Cuadrícula * Cuadrados |
| Josué Jiménez Alfaro | Interfaz Gráfica  Verificación de Errores | * Pantalla * CodigoIntermedio * Directorio |
| Sebastián Salas García | Tokeniza | * Tokeniza * Scan * Archivo |

# Cronograma

A continuación, la fechas de control del trabajo establecidas por el profesor.

|  |  |  |
| --- | --- | --- |
| **Fecha** | **Objetivo** | **¿Realizado?** |
| Jueves 17 de marzo | Consulta | No se logró reunirse por complicaciones. |
| Jueves 31 de marzo | Diseño de las clases y distribución del trabajo. | Se realizó las consultas necesarias, se presentó la distribución del trabajo y se pasó el diseño de las clases para la siguiente reunión. |
| Jueves 7 de abril | Punto de Control No.1 | Por realizar. |
| Jueves 14 de abril | Punto de Control No.2 | Por realizar. |
| Jueves 21 de abril | Entrega final (Revisión)  Expo. | Por realizar. |

# Resultados Obtenidos

Como resultados obtenidos podemos afirmar que el trabajo se logró completar satisfactoriamente, donde el trabajo en equipo arduo nos solicitó poder completar la tarea rápidamente.

Se realizó bastantes pruebas con el objetivo de ver errores posibles en el sistema y así pulir los detalles para que en conclusión el trabajo no tuviera errores.

# Conclusiones

Gracias a este proyecto pudimos reforzar las destrezas de abstracción, representación y transformación para desarrollar un simulador de un sistema de administración de archivos. Se implementó por otro lado, el uso de clases, listas y diccionarios como estructuras centrales para moldear el problema.

Por otro lado, desarrollamos la interfaz gráfica utilizando tkinter, se investigó las técnicas de treeviews, creación de cuadros, tabs, entre otros.

En conclusión, creemos que este trabajo nos ha ayudado tanto en la parte didáctica como en la parte de trabajo en conjunto. Vemos como el trabajo en equipo es vital para poder desarrollar trabajos grandes en tiempos determinados, donde aplicando el dicho de “Divide y vencerás” fue de gran utilidad para salir abantes.