UNIVERSIDAD TECNOLÓGICA

Manejador de Archivos

Estudiante:
Ana Lucía Santos Ordoñez

Número de Cuenta:
11141163

Catedrático:
Carlos Arias

Lunes, 23 de septiembre de 2013

Introducción

El proyecto consiste en crear un programa la cual pueda crear archivos de registros y realizar las operaciones que conlleva de forma eficiente, para esto fueron necesarios los conocimientos de archivos, la definición de su estructura y como manejarlo.

Fue desarrollado con el lenguaje C++ y como herramienta gráfica Qt

Marco Teórico

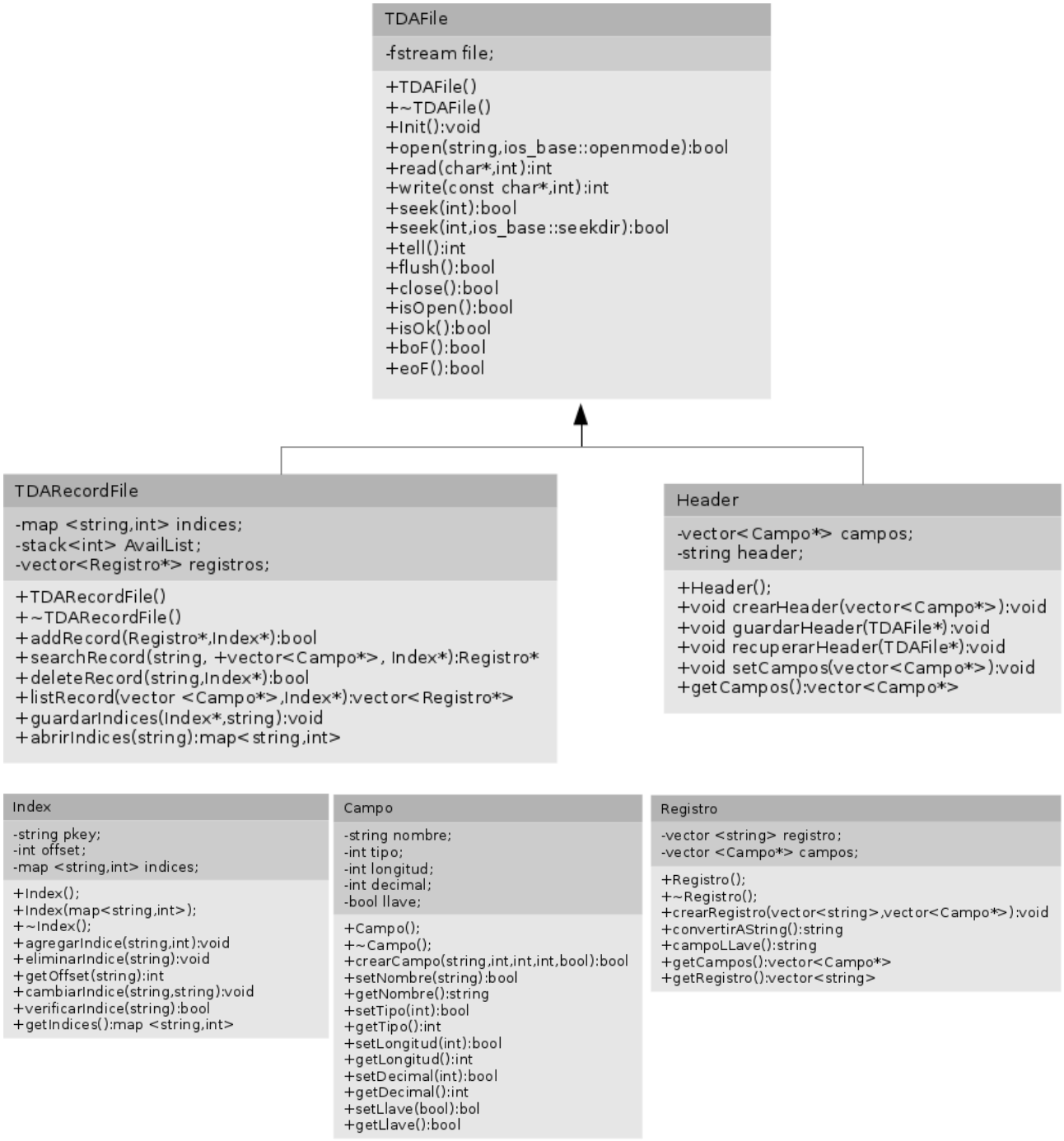
Un archivo es una colección de bytes que representa información relacionada, estos son colocados en el almacenamiento secundario, ya que esta es no volátil y así se puede conservar esa información. Es inevitable el uso del almacenamiento secundario, pero siempre se procura minimizar su uso. El almacenamiento secundario necesita más tiempo para acceder a los datos, razón por la cual para poder acceder de manera eficaz a los archivos se usan estructuras de archivos. Las estructuras de archivos para este proyecto incluyen campos, registros, indices, árboles.

Cuando se habla de archivos en el disco duro, se refiere a archivos físicos, es decir, son un grupo de bytes almacenados físicamente. Cuando el programa accede al archivo, se crea una conexión entre el disco duro y el programa a la cual se le llama archivo lógico, y el sistema operativo es el que se encarga de cerrar los archivos físicos.

Los archivos de registros son archivos compuestos de registros, que estos a su vez, están compuestos de campos. Un campo es la unidad de información lógicamente significativa más pequeña en un archivo. Entonces, un registro es un conjunto de campos agrupoados bajo la perspectiva de un archivo de nivel más alto de organización.

Para acceder a estor registros se puede hacer una busqueda secuencial o acceso directo por medio de índices. Para poder hacer acceso directo a los registros se necesita conocer el Numero relativo de registro (NRR) o su posición física, y una llave primaria que identifica al registro de manera única, a la estructura de datos que guarda esta información en memoria es a la que se le conoce como Índices.

Implementación



La clase TDAFile

open(string,ios_base::openmode) abre el archivo,recibe como parametros la ubicación del archivo y el modo de abrirlo.

read(char*,int) recibe un apuntador al buffer y la cantidad de bytes que va leer.

write(const char*,int) recibe un apuntador al bufer y la cantidad de bytes que va escribir.

seek(int) mueve el puntero a la posición que recibe como parametro.

seek(int,ios_base::seekdir) mueve el puntero.

int tell() retorna la posición en que se encuentra el punter.

flush() escribe en el archivo lo que se encuetra en el buffer del os.

close() cierra el archivo.

isOpen() retorna un boolean si el archivo está abierto o no.

bool isOk() retorna un boolean de acuerdo a las banderas de error.

bool boF() retorna un boolean si el archivo está al principio del archivo.

bool eoF() retorna un boolean si el archivo está al final del archivo.

La clase TDARecordFile

addRecord(Registro*,Index*) recibe un apuntador al registro y guarda el registro en el archivo.

searchRecord(string, vector<Campo*>, Index*) retorna un registo que busca un registro por medio de la llave recibe y los indices.

deleteRecord(string,Index*) borra un registro de los indices y lo marca en el archivo.

listRecord(vector <Campo*>,Index*) retorna un vector de registros.

guardarIndices(Index*,string); crea un archivo de indices y los guarda.

abrirIndices(string) abre el archivo de índices y los retorna.

guardarXML(vector <Campo*>, Index*,string) exporta los registros a un archivo xml

La clase Header

crearHeader(vector<Campo*>) crea un string con los datos sobre los campos.

guardarHeader(TDAFile*) guarda el header en el archivo.

void recuperarHeader(TDAFile*) lee el header del archivo guarda la estructura en memoria.

setCampos(vector<Campo*>) recibe un vector con los campos.

getCampos() retorna un vector con los campos.

La clase Index
void agregarIndice(string,int) agrega la llave y su posición en el mapa que manea los índices.

void eliminarIndice(string) elimina un llave del mapa.

int getOffset(string) recibe la llave y retorna la posición del registro.

void cambiarIndice(string,string) actualiza la llave si se cambió un registro.

bool verificarIndice(string) revisa si ya existe la llave en la estructura de índices.

getIndices() retorna el mapa donde se manejan los índices

La clase Registro
crearRegistro(vector<string>,vector<Campo*>) crea un registro.

ConvertirAString() convierte el registro en un string.

campoLLave() retorna el campo llave.

getCampos() retorna los campos

getRegistro() retorna los campos de un registro

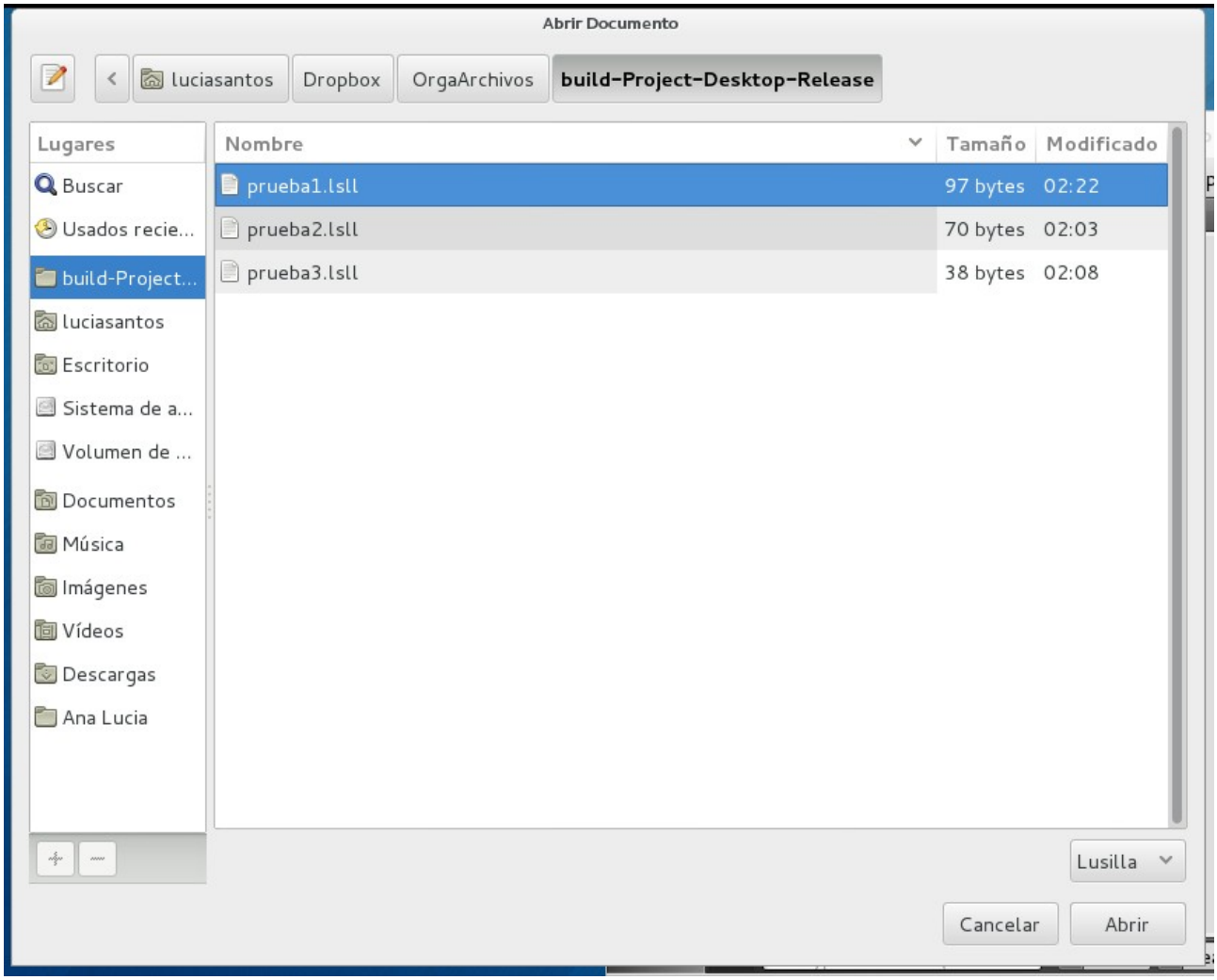
La clase Campo
crearCampo(string,int,int,int,bool) crea un campo

Obtiene o modifica cada atributo de un campo
bool setNombre(string);
string getNombre();
bool setTipo(int);
int getTipo();
bool setLongitud(int);
int getLongitud();
bool setDecimal(int);
int getDecimal();
bool setLlave(bool);
bool getLlave();

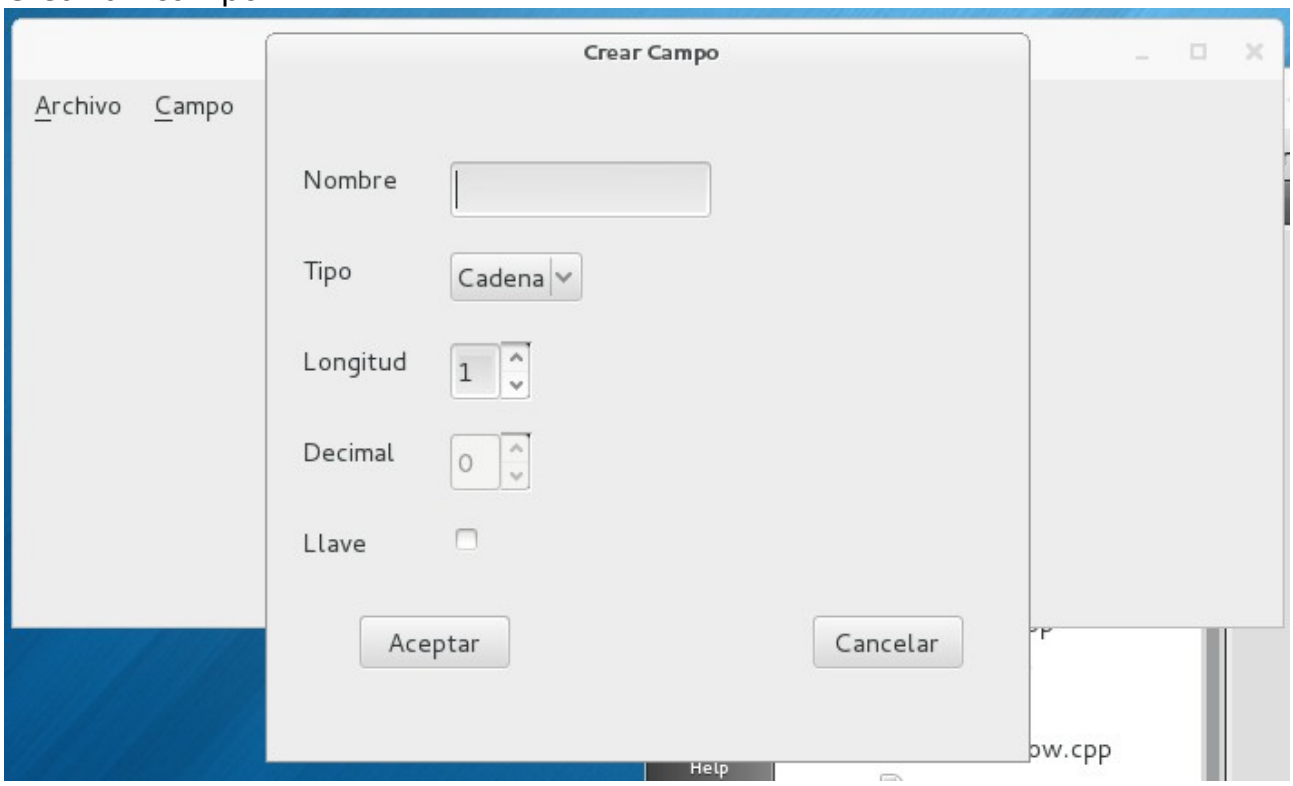
Muestra de Corrida

Pantalla principal

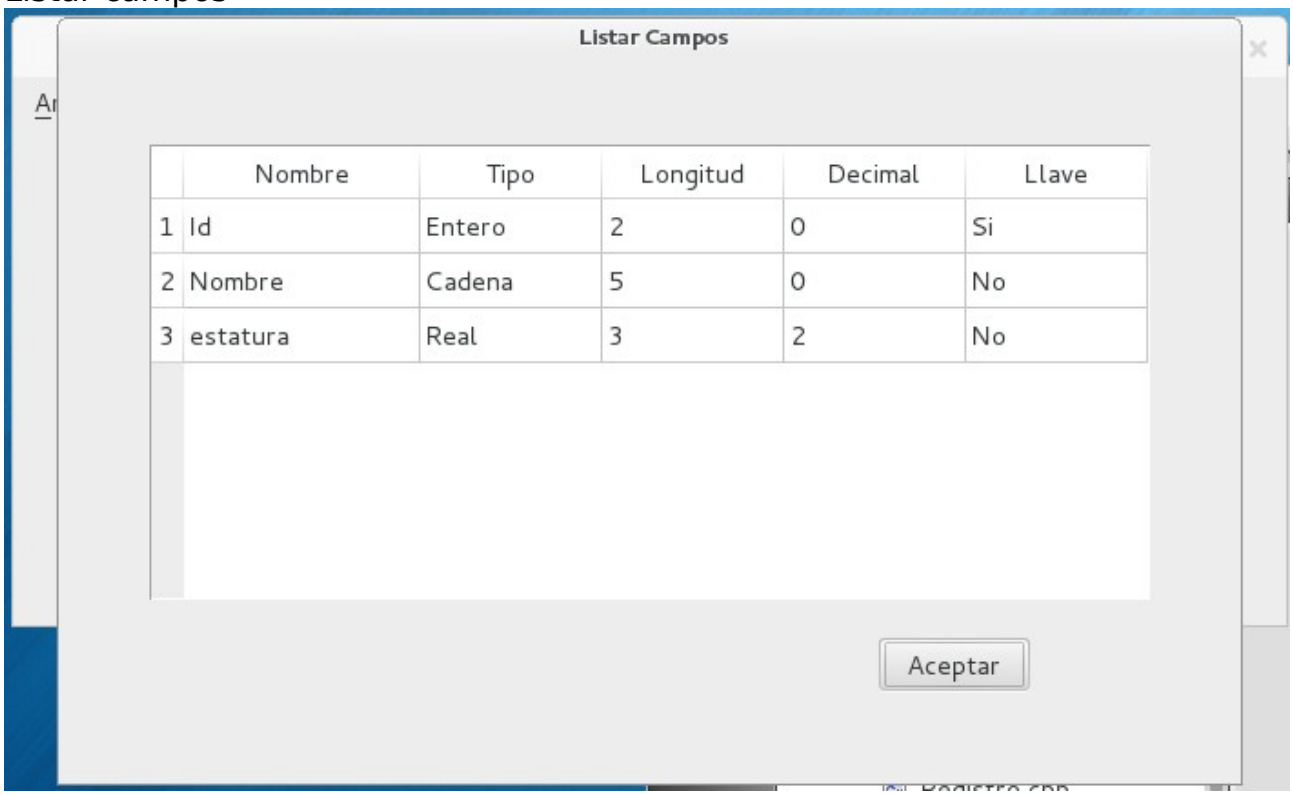
Abrir un archivo



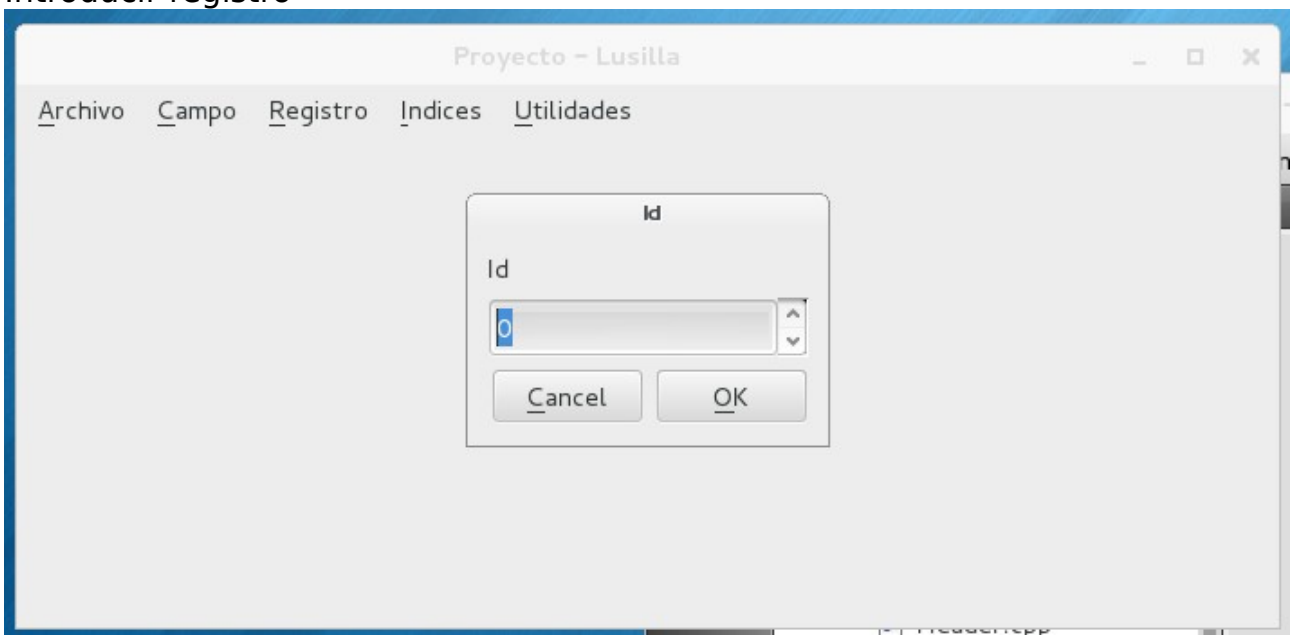
Crear un campo



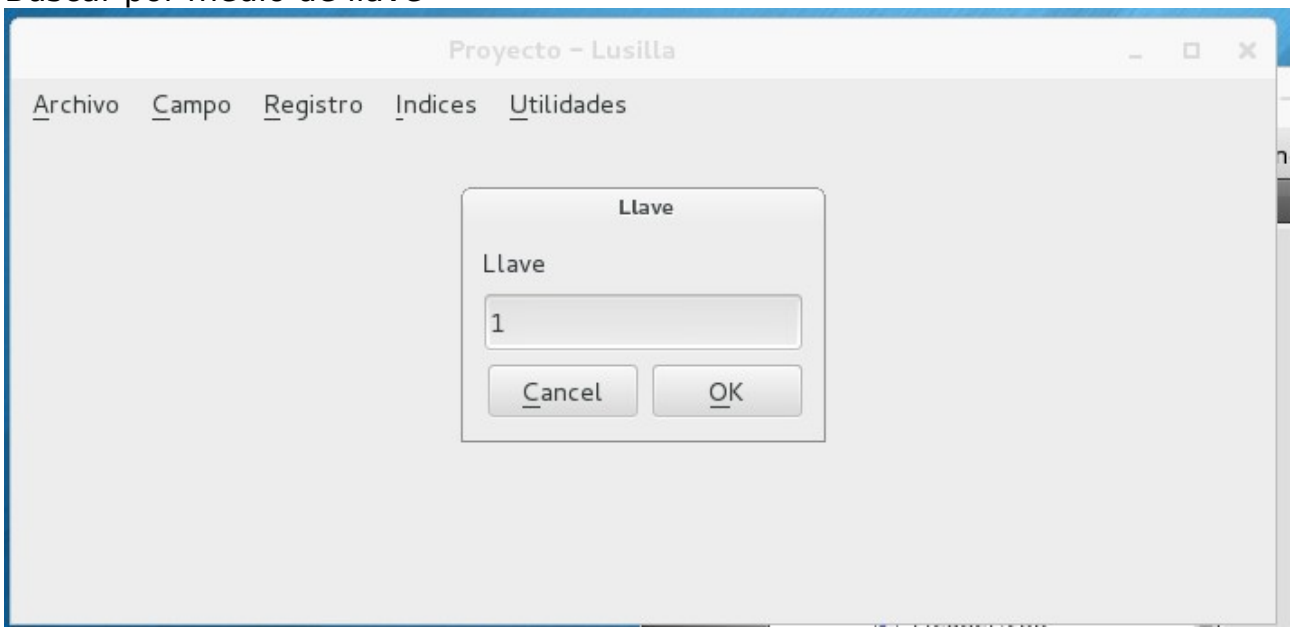
Listar campos



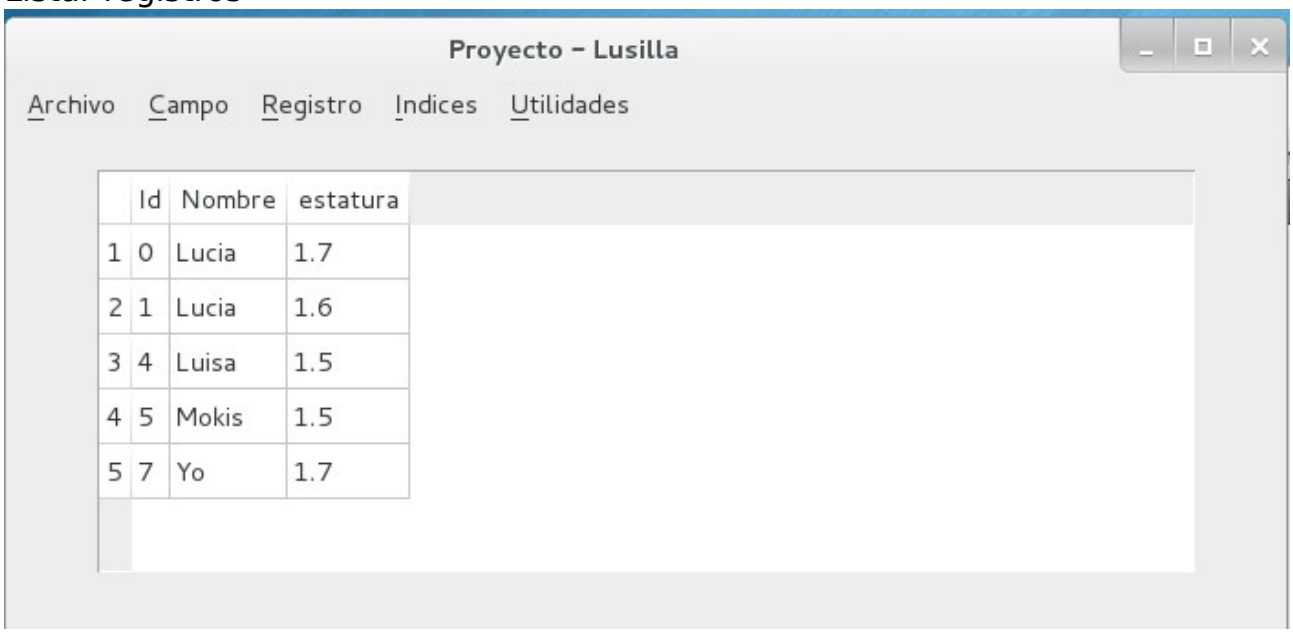
Introducir registro



Buscar por medio de llave



Listar registros



Experiencia con Git

Usar git resultó útil ya que tenía un respaldo del proyecto, cuando había modificado algo que hizo que el programa no funcionara pude restaurar el proyecto sin ningun problema.

Para trabajar en grupo es importante primero hacer un pull, porque si no se hace se generan lineas de error dentro del código y hay que borrarlas manualmente.

En general, fue satisfactorio aprender a usar una herramienta para manejar los repositorios, compartirlos y recuperar sus versiones anteriores fácilmente.

El repositorio donde se encuentra el proyecto es:
https://github.com/LuciaSantos89/Orga

Conclusiones

Es importante conocer la implementacion de estructuras de archivos para que, aun cuando tengamos que usar el almacenamiento secudario, los accesos a este sean la menor cantidad posible, de esa forma se ahorra tiempo. También es importante conocerlas para usar la memoria principal de forma de no ocuparla completamente.