

Añadiendo memoria a la aplicación

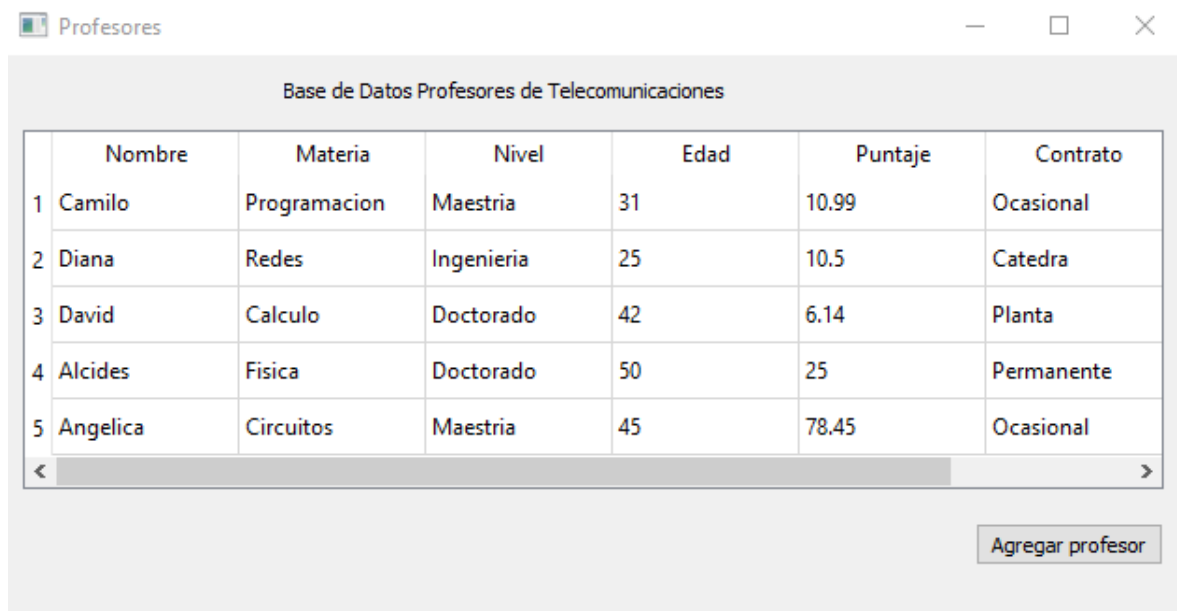
Lectura y escritura de archivos de valores separados por comas (CSV)

Objetivo: Añadir registro en memoria a la aplicación mediante la grabación de datos en un archivo separado por comas y a su vez desarrollar una inicialización de la aplicación que permita cargar los datos ingresados en sesiones anteriores.

Nota: Este instructivo se trabajará a partir de lo que se desarrolló en el Paso a paso 3. Trabaje en el mismo proyecto y simplemente adicione las nuevas características que se verán en este documento.

1. Archivo de valores separados por comas (.csv)

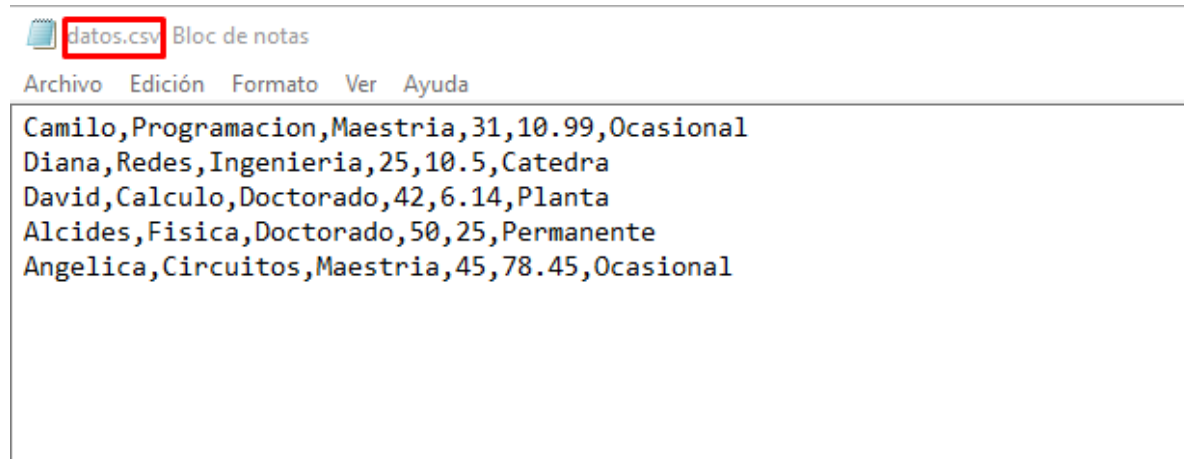
En una carpeta de su computador, preferiblemente en una ruta que usted destine para la aplicación agregue un archivo de valores separados por comas. El archivo debe contener por lo menos un renglón relleno con información perteneciente a uno de los registros que su aplicación muestra en la tabla creada con el elemento QWidget. Por ejemplo, en mi aplicación de prueba la tabla muestra los siguientes datos de los profesores del departamento de telecomunicaciones: Nombre, Edad, Materia, Nivel académico, Tipo de vinculación a la universidad y Puntaje de calificación del profesor.



	Nombre	Materia	Nivel	Edad	Puntaje	Contrato
1	Camilo	Programacion	Maestria	31	10.99	Ocasional
2	Diana	Redes	Ingenieria	25	10.5	Catedra
3	David	Calculo	Doctorado	42	6.14	Planta
4	Alcides	Fisica	Doctorado	50	25	Permanente
5	Angelica	Circuitos	Maestria	45	78.45	Ocasional

Por tanto, el archivo de valores separados por comas debe contener registros de datos que encajen con registros como los que se muestran en la tabla. Por ejemplo, el primer registro puede guardarse en el archivo .csv como: Camilo,Programacion,Maestria,31,10.99,Ocasional. Y de la misma manera los demás

registros. La idea es automatizar ese proceso, que sea la aplicación misma la que se encargue de guardar los registros y pueda leerlos luego cuando se vuelva a utilizar la aplicación. Un ejemplo del archivo para la aplicación de profesores puede verse a continuación, recuerde que el archivo de valores separados por comas es un archivo con extensión .csv



2. Guardando los registros de la tabla en un archivo de valores separados por comas .csv

En el slot clicked() del Push Button que se agregó a la clase Widget (el botón de la ventana donde agregó el widget QTableWidgetItem) se debe agregar la lógica para que a la vez que muestra en la tabla el nuevo registro ingresado también lo guarde en el archivo .csv para que pueda ser almacenado en disco.

En primer lugar, se debe crear un vector de vectores del tipo de dato string que va servir para almacenar de manera segmentada y ordenada todos los registros de la tabla con todos los datos de las columnas de la tabla. Es decir, a cada vector de vectores string se le mapea un registro y a cada vector string interno se le mapean los strings de los datos de las columnas, `vector<vector<string>> csvData`. En la clase Widget (archivo widget.h) agregue un atributo que cumpla con estas características, es decir un vector de vectores string. No olvide incluir las librerías vector y string.

```
TablaProfes.h*  csvData: vector<vector<string>>
#define TABLAPROFES_H

#include <QWidget>
#include<vector>
#include<string>

using namespace std;

namespace Ui {
class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();

private slots:
    void on_agregar_clicked();

    void readCSV(istream &input, vector< vector<string> > &output);

    void initData(vector< vector<string> > &datos);

    void saveData(vector< vector<string> > &output);

private:
    Ui::Widget *ui;
    vector< vector<string> > csvData;
};

#endif // TABLAPROFES_H
```

Agregue a la clase un método en el que desarrolle la lógica que permite guardar en un archivo .csv los registros recientemente agregados a la tabla. En este ejemplo el método se llama **void saveData(vector <vector<string>> &output)**, puede verlo declarado en la imagen anterior y su desarrollo en la imagen siguiente (en el archivo widget.cpp)

```

void Widget::saveData(vector< vector<string> > &output)
{
    ofstream myfile;
    string a;

    myfile.open("C:/Users/juancorrea/Documents/Cursos/2018_2/Informatica_2/TablaProfes/datos.csv");

    for(vector<vector<string>>::iterator i = output.begin(); i != output.end(); ++i)
    {
        for(vector<string>::iterator j = i->begin(); j != i->end(); ++j)
        {
            a=*j;
            myfile <<a;
            if(j != (i->end())-1)
                myfile<<",";
        }
        myfile <<"\n";
    }
    myfile.close();
}

```

Este metodo en primer lugar declara una variable “myfile” que sirve para que se pueda abrir el archivo y modificarlo luego, no olvide agregar la ruta de su archivo en particular en el llamado a la función `myfile.open()` . Luego simplemente con un iterador de la clase `vector` se iteran todos los registros de la tabla y cada uno de los datos de las columnas y con el operador “<<” se escriben en el archivo. En el dato de la última columna se evita poner de nuevo la coma y se agrega un salto de línea en su lugar. No olvide incluir las siguientes librerías: `iostream`, `fstream`, `sstream`, `string` y `vector`. Y el **using namespace std;** en caso de no tenerlo.

Luego en el metodo `clicked()` del Push Button de la clase `Widget`, justo después de la parte donde agrega los registro en la tabla, agregue la parte en la que va guardar los registros en el archivo. Almacene todos los datos de las columnas en un vector de strings y luego cada registro agreguelo al vector de vectores string que añadío como atributo a la clase `Widget`. Observe en la imagen siguiente que se hace uso del atributo y del método recién creados en el paso anterior.

```

ui->tabla->insertRow(ui->tabla->rowCount()); //Inserte una fila en la tabla
fila = ui->tabla->rowCount()-1; //Identifique la fila en la que va escribiendo la tabla
ui->tabla->setItem(fila, 0, new QTableWidgetItem(nombre)); //Escriba los datos capturados en el dialogo
ui->tabla->setItem(fila, 1, new QTableWidgetItem(materia)); //en las columnas de la tabla
ui->tabla->setItem(fila, 2, new QTableWidgetItem(nivel)); //columnas de la 0 a la 5 (6 columnas)
ui->tabla->setItem(fila, 3, new QTableWidgetItem(QString::number(edad)));
ui->tabla->setItem(fila, 4, new QTableWidgetItem(QString::number(puntaje)));
ui->tabla->setItem(fila, 5, new QTableWidgetItem(contrato));

vector<string> new_data;
new_data.push_back(nombre.toStdString());
new_data.push_back(materia.toStdString());
new_data.push_back(nivel.toStdString());
new_data.push_back(QString::number(edad).toStdString());
new_data.push_back(QString::number(puntaje).toStdString());
new_data.push_back(contrato.toStdString());

csvData.push_back(new_data);
saveData(csvData);
}

```

3. Leyendo los registros del archivo .csv para inicializar la tabla con los datos que se agregaron con anterioridad.

Agregue a la clase Widget un método para leer datos de registros guardados en un archivo de valores separados por comas. El método debe recibir dos parámetros de entrada, uno para pasarle al método el stream de datos leídos de un archivo almacenado en disco y otro para almacenar los datos leídos en una estructura de datos fácil de manipular y que permita agilizar tanto el proceso de cargar los datos antiguos en la tabla al inicializarla; como el proceso de guardado de nuevos datos ingresados (concatena los nuevos registros a los antiguos para que el archivo se conserve sincronizado y completo).

```
#define TABLAPROFES_H

#include <QWidget>
#include<vector>
#include<string>

using namespace std;

namespace Ui {
class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();

private slots:
    void on_agregar_clicked();

    void readCSV(istream &input, vector< vector<string> > &output);

    void initData(vector< vector<string> > &datos);

    void saveData(vector< vector<string> > &output);

private:
    Ui::Widget *ui;
    vector< vector<string> > csvData;
};

#endif // TABLAPROFES_H
```

La lógica del método para leer archivos de valores separados por comas es muy sencilla, simplemente se captura cada uno de los renglones del archivo en una variable de tipo `istream` y luego se itera cada renglón separando los componentes del string cada que se encuentre una coma (,) cada string se va almacenando en un vector de strings y luego ese vector se almacena en el atributo vector de vectores strings de la clase. De ese modo se almacenan todos los registros del archivo en el vector de vectores string.

```
void Widget::readCSV(istream &input, vector< vector<string> > &output)
{
    string csvLine;

    // se lee cada renglon del archivo .csv
    while( getline(input, csvLine) )
    {

        istringstream csvStream(csvLine);
        vector<string> csvColumn;
        string csvElement;

        //De cada renglon se van separando las palabras
        //cada que se encuentre una coma (,)
        while( getline(csvStream, csvElement, ',') )
        {
            csvColumn.push_back(csvElement);
        }
        output.push_back(csvColumn);
    }
}
```

4. Lectura del archivo e inicialización de la tabla

La lectura de los registros y posterior inicialización de la tabla se lleva a cabo en el constructor de la clase Widget, justo después de crear e inicializar los parámetros de inicio del widget `QTableWidget` (título, columnas, nombre de las columnas, etc.). El archivo se lee con ayuda de un objeto de tipo `fstream` al cual se le indica la ruta donde esta almacenado el archivo y que el archivo es de entrada. Luego se verifica si el archivo está abierto, sino la tabla se inicializa vacía. Luego se llama el método recién creado para leer el archivo y un método adicional para inicializar la tabla con los registros antiguos: `void initData(vector< vector<string> > &datos)`

```

Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);
    setWindowTitle("Profesores"); //Dandole nombre a ventana de la tabla
    QStringList columnas; //Defina una lista para los nombres de las columnas
    columnas<<"Nombre"<<"Materia"<<"Nivel"<<"Edad"<<"Puntaje"<<"Contrato"; //Agregue los nombres a la lista
    ui->tabla->setColumnCount(6); //Defina el numero de columnas de la tabla
    ui->tabla->setHorizontalHeaderLabels(columnas); //Asigne los nombres de la lista a la tabla

    fstream file("C:/Users/juancorrea/Documents/Cursos/2018_2/Informatica_2/TablaProfes/datos.csv", ios::in);

    if(file.is_open())
    {
        readCSV(file, csvData);
        initData(csvData);
    }
    else
        cout << "La ruta del archivo no se encontro, la tabla inicia vacia\n";
}

```

El método para inicializar la tabla se agrega a la clase Widget, este método simplemente itera sobre todos los datos que fueron leídos del archivo de valores separados por comas .csv y que se encuentran almacenados en el atributo vector de vectores string de la clase Widget (recuerde que esa es la funcionalidad implementada en el método de lectura) y los agrega a la tabla de la misma manera que se hace cuando se da clic en el Push Button de la clase Widget, por cada registro se agrega una nueva fila a la tabla y cada dato del vector de strings se agrega a una columna individual de la tabla.

```

#define TABLAPROFES_H

#include <QWidget>
#include<vector>
#include<string>

using namespace std;

namespace Ui {
class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();

private slots:
    void on_agregar_clicked();

    void readCSV(istream &input, vector< vector<string> > &output);

    void initData(vector< vector<string> > &datos);

    void saveData(vector< vector<string> > &output);

private:
    Ui::Widget *ui;
    vector< vector<string> > csvData;
};

#endif // TABLAPROFES_H

```

```
void Widget::initData(vector< vector<string> > &datos)
{
    int fila = 0;

    for(int i = 0; i<datos.size(); ++i)
    {
        ui->tabla->insertRow(ui->tabla->rowCount()); //Inserte una fila en la tabla
        fila = ui->tabla->rowCount()-1; //Identifique la fila en la que va escribiendo la tabla
        for(int j = 0; j<datos[i].size(); ++j)
        {
            //Agregando un dato a una columna
            ui->tabla->setItem(fila, j, new QTableWidgetItem(QString::fromStdString(datos[i][j])));
        }
    }
}
```