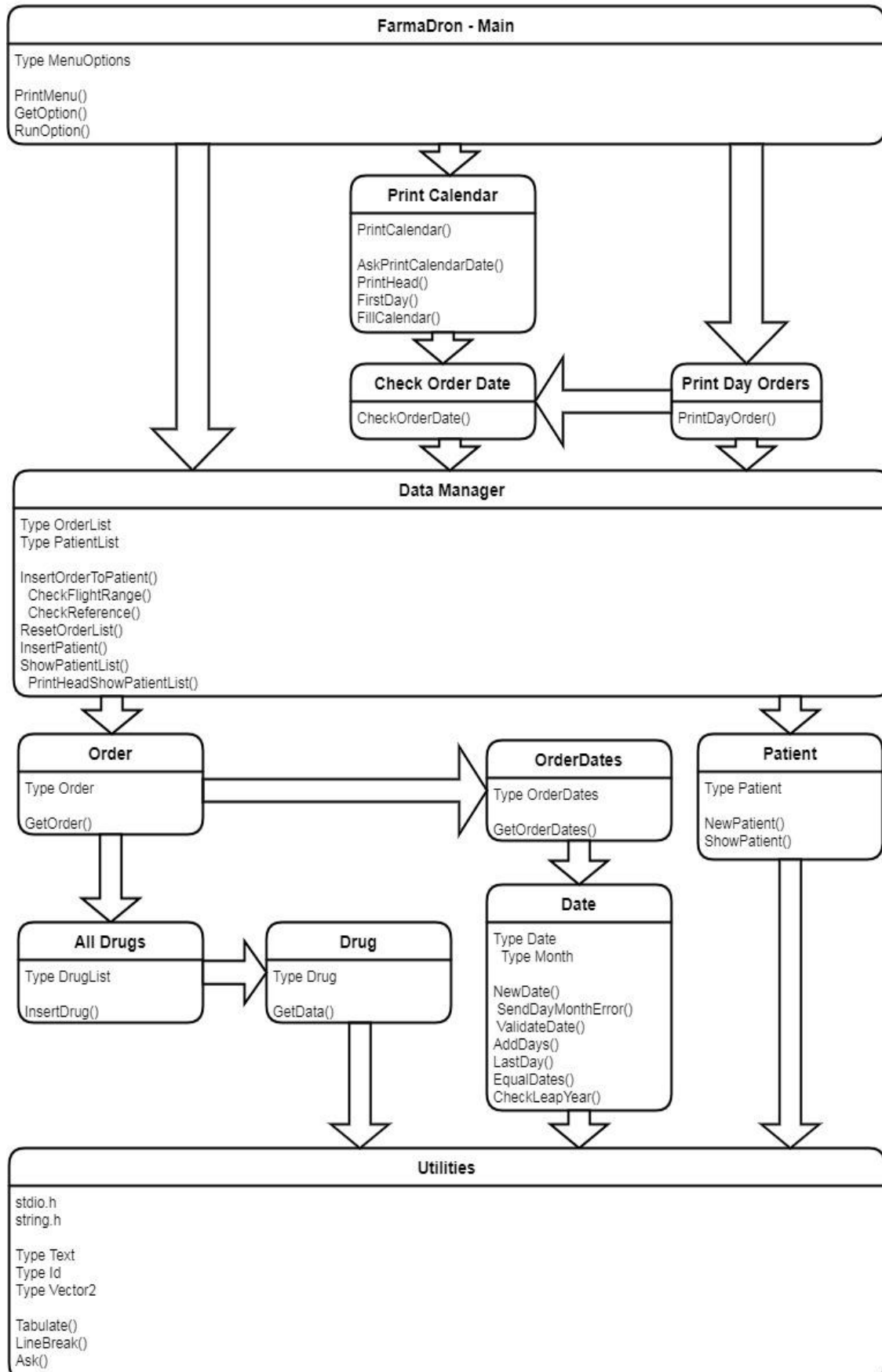


# FarmaDron



## 1.- Presentación.

La práctica consiste en realizar un programa para la gestión de la distribución de fármacos; en tener registrados pacientes con los datos que los identifiquen y pedidos con, a su vez, datos que puedan identificarlos.

La aplicación, además de registrar los datos de pacientes y pedidos, debe hacer distintas exposiciones de los datos que almacena; gestionarlos y ordenarlos para presentarlos como lo demande el usuario.

Una primera aproximación lleva a dividir el programa en 3 bloques:

1. Datos.
2. Gestión de los datos.
3. Interfaz aplicación usuario.

### 1.1.- Presentación del *Bloque Datos*.

El lenguaje de programación C++ está dotado de tipos de datos y operaciones básicas con los que trabajar; por desgracia, ni los datos de tipo *paciente*, ni los datos de tipo *pedido* ni las operaciones necesarias para tratar ninguno de los datos que reconoce la aplicación –*pacientes* y *pedidos*– están incluidos entre los datos y operaciones básicas del lenguaje C++.

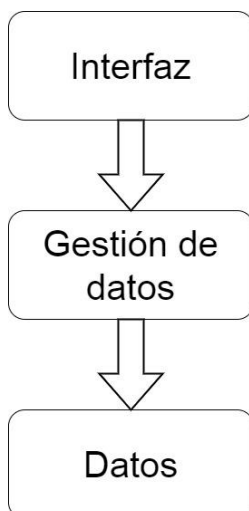
En este bloque del programa, *Datos*, a partir de tipos de datos y operaciones básicas para el lenguaje C++, se crearán los tipos de datos y operaciones con los que podrá operar la aplicación.

### 1.2.- Presentación del *Bloque Gestión de los Datos*.

En el bloque de *Gestión de los Datos* se ejecutarán las instrucciones que el usuario dé a la aplicación tanto para registrar datos de pacientes y pedidos como para consultar información que tenga almacenada exponiendo la información registrada como el usuario solicite.

### 1.3.- Presentación del bloque *Interfaz*

La aplicación ofrece al usuario las opciones que tiene a su disposición y recibe sus instrucciones a través de este bloque, puede considerarse como la vía de comunicación entre los datos y sus gestores y el usuario.



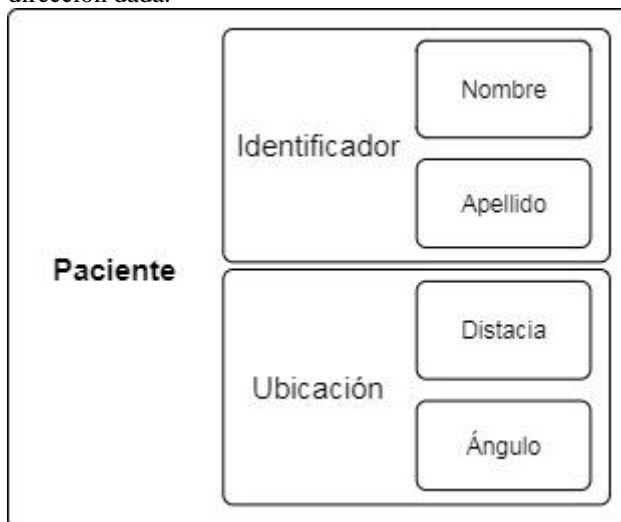
## 2.- Primer Bloque: *Datos*.

La aplicación reconoce dos tipos de datos: *pacientes* y *pedidos*, creando dos categorías para los tipos de datos.



### 2.1.- Pacientes.

Un paciente tiene un *identificador* y una *ubicación*. El *identificador* está compuesto por el *nombre* y *apellido* del paciente y la *ubicación* por la *distancia* a la base y el *ángulo* que forma la distancia con una dirección dada.



En este punto, se puede identificar cada una de las referencias del tipo de dato *paciente* con tipo de datos que sí reconocer el lenguaje C++.

- **Nombre** → Cadena de caracteres.
- **Apellido** → Cadena de caracteres.
- **Distancia** → Número entero.
- **Ángulo** → Número entero.

Cualquier variable del tipo de dato *paciente* que se cree tendrá 2 funciones:

- **NewPatient.** La función inicializa la variable del dato tipo *paciente* preguntando por pantalla los datos de cada uno de los campos al usuario, discriminando datos que no tengan sentido para la aplicación como distancias fuera del radio de acción del Dron.

```

C:\Users\Ricardo\Documents\C+-\Practicas\FarmaDron\FarmaDron2\bin\Release\FarmaDron2.exe
FarmaDron: Distribucion de Farnacos con Dron
Alta nuevo paciente (Pulsar A)
Ubicar pacientes (Pulsar U)
Nuevo pedido (Pulsar N)
Lista diario de pedidos (Pulsar L)
Calendario mensual de pedidos (Pulsar C)
Salir (Pulsar S)
Teclear una opcion valida (A;U;N;L;C;S)? A
Alta nuevo paciente. Referencia paciente: 1
Identificador (entre 1 y 20 caracteres): Pedro Pérez
Distancia (hasta 10000 metros a plena carga): 50000
Introdujo: 50000
El FarnaDron no hace portes a mas de 15000 metros.
Distancia (hasta 10000 metros a plena carga): 5423
Angulo (entre 0 y 2000 pi / 1000 radianes): -10
Introdujo: -10
Angulo (entre 0 y 2000 pi / 1000 radianes): 456
Los datos son correctos? (S/N)?

```

Distancia

Ángulo

- **ShowPatient.** Muestra por pantalla los datos de la variable tipo *paciente* de acuerdo a un formato.

```

C:\Users\Ricardo\Documents\C+-\Practicas\FarmaDron\FarmaDron2\bin\Release\FarmaDron2.exe
Los datos son correctos? (S/N)? S
Otro paciente? (S/N)? N
Desea volver al menu FarnaDron? (S/N) S
FarmaDron: Distribucion de Farnacos con Dron
Alta nuevo paciente (Pulsar A)
Ubicar pacientes (Pulsar U)
Nuevo pedido (Pulsar N)
Lista diario de pedidos (Pulsar L)
Calendario mensual de pedidos (Pulsar C)
Salir (Pulsar S)
Teclear una opcion valida (A;U;N;L;C;S)? U
Lista de pacientes y su ubicacion:

```

Ref.	Identificador	Distancia	Angulo
1	Pedro Pérez	5423	456
2	Maria López	8876	1356
3	José Gómez	2789	867
4	Pablo García	1765	1823
5	Pilar González	11345	145

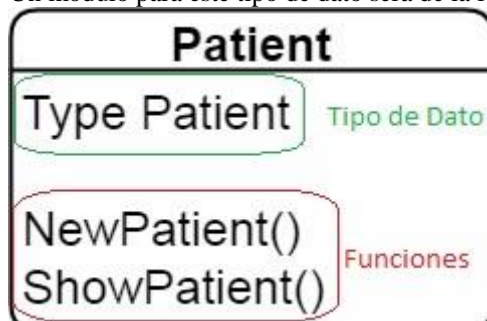
```

Desea volver al menu FarnaDron? (S/N)

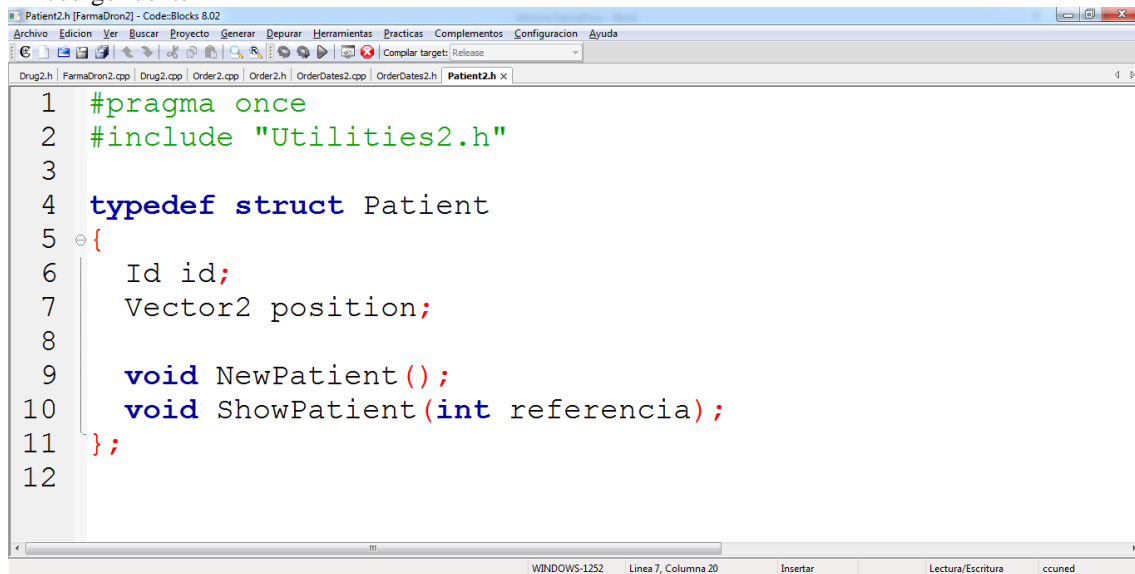
```

Datos de la variable tipo *paciente*

Un módulo para este tipo de dato será de la forma



## En código fuente

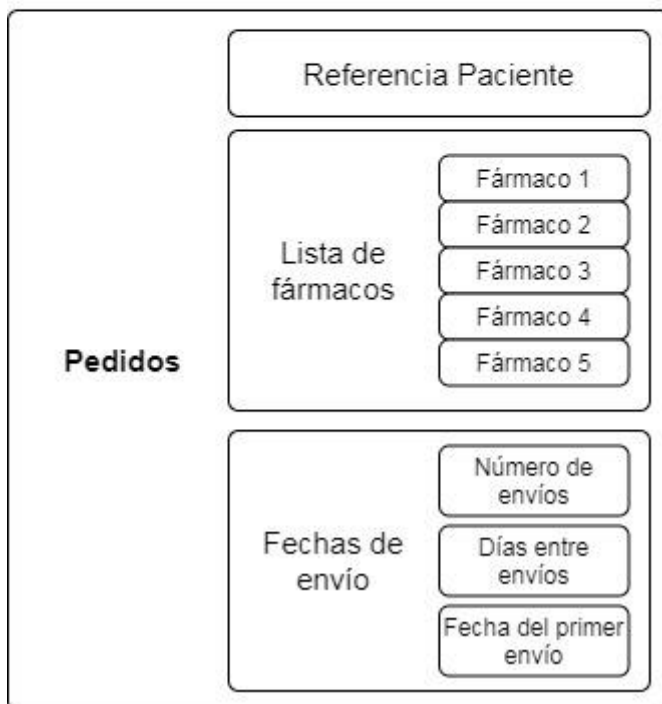
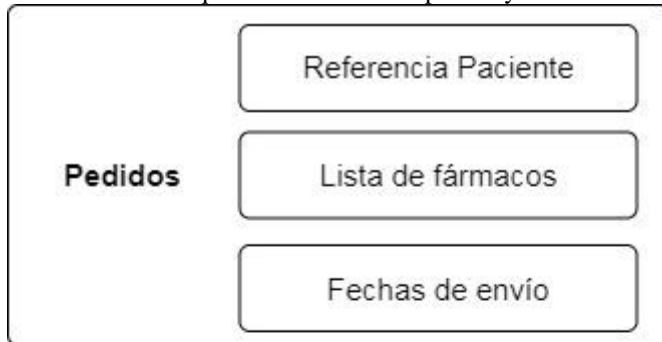


```
1  #pragma once
2  #include "Utilities2.h"
3
4  typedef struct Patient
5  {
6      Id id;
7      Vector2 position;
8
9      void NewPatient();
10     void ShowPatient(int referencia);
11 };
12
```

STATUS: WINDOWS-1252 | Linea 7, Columna 20 | Insertar | Lectura/Escritura | ccuned

## 2.2.- Pedidos.

Un pedido es identificado por los campos que hacen referencia al paciente que se le enviará el pedido, la lista de fármacos que se enviarán en el pedido y las fechas en las que se deben hacer los envíos.



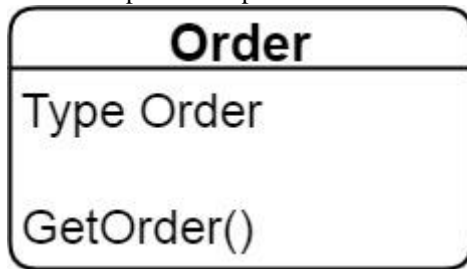
Identificando cada una de las referencias del tipo de dato *Pedido*:

- **Referencia paciente** → Número entero. Resulta un tipo de dato reconocido por C++.
- **Fármaco** → Dato de tipo *Fármaco*.
- **Número de envíos** → Número entero. Tipo de dato reconocido por C++.
- **Días entre los envíos** → Número entero. Tipo de dato reconocido por C++.
- **Fecha del primer envío** → Dato de tipo *Fecha*.

Cualquier variable del tipo *Pedido* que se cree tendrá una función:

- **GetOrder**. La función inicializa la variable del dato tipo *pedido* preguntando por pantalla los datos necesarios hasta haber completado la información del pedido.

El módulo para este tipo de dato será de la forma:



En código fuente:

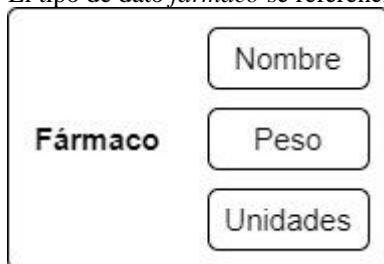
```
Order2.h [FarmaDron2] - Code::Blocks 8.02
Archivo Edición Ver Buscar Proyecto Generar Depurar Herramientas Practicas Complementos Configuración Ayuda
C++11 Compiler target: Release
Drug2.h FarmaDron2.cpp Drug2.cpp Order2.cpp Order2.h x OrderDates2.cpp OrderDates2.h Patient2.h
1 #pragma once
2 #include "OrderDates2.h"
3 #include "AllDrugs2.h"
4
5 typedef struct Order
6 {
7     int refPatient;
8     OrderDates orderDate;
9     DrugList orderDrugs;
10
11     void GetOrder(int reference);
12 };
13
```

En el análisis de los tipos de datos que componen el tipo de dato *paciente*, aparecieron dos tipos de datos que no reconoce el lenguaje C++:

- *Fármaco*.
- *Fecha*.

### 2.2.1.- Fármacos.

El tipo de dato *fármaco* se referencia con los campos: nombre, peso y unidades.



Identificando cada uno de los campos que componen el tipo de dato *fármaco*, se encuentran tipos de datos que sí reconoce el lenguaje C++.

- **Nombre** → Cadena de caracteres.
- **Peso** → Número entero.
- **Unidades** → Número entero.

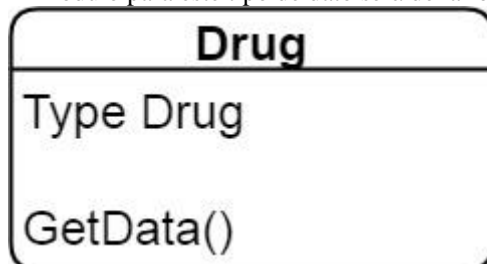
Cualquier variable del tipo *Fármaco* que se cree tendrá la función:

- **GetData.** La función inicializa la variable del dato tipo *fármaco* preguntando por pantalla al usuario los datos que identifican el dato de tipo *fármaco*, discriminando datos que no tengan sentido para la aplicación como pesos del fármaco superiores a la capacidad de transporte del dron.

```
C:\Users\Ricardo\Documents\C++\Practicas\FarmaDron\FarmaDron2\bin\Release\FarmaDron2.exe

Ref. Paciente <entre 1 y 20> 3
Numero de envios? 1
Dia del envio? 23
Mes del envio? 1
Año del envio? 2021
Nombre del farmaco <Entre 1 y 20 caracteres>? Analgésico
Peso farmaco <Menor de 3000 gramos>? 3050
No se transporta peso igual o superior a 3000 gramos.
Peso farmaco <Menor de 3000 gramos>? 1000
Unidades de farmaco? 4
Introdujo: 4 unidades * 1000 peso por unidad, Resulta: 4000 gramos.
El sistema no transporta mas de 3000 gramos.
Peso farmaco <Menor de 3000 gramos>? _
```

El módulo para este tipo de dato será de la forma:



En código fuente:

```
#pragma once
#include "Utilities2.h"

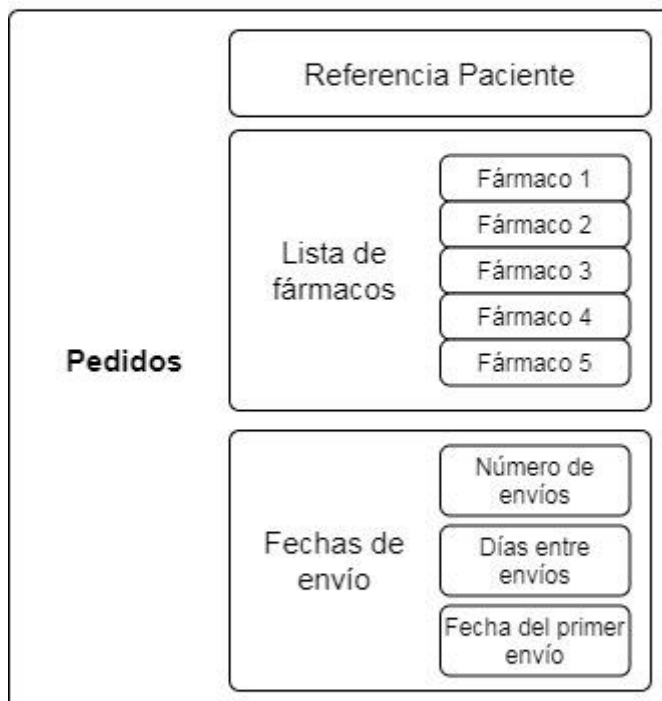
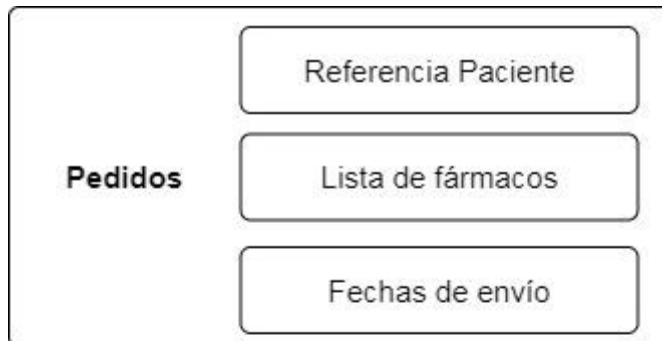
typedef struct Drug {
    Text name;
    int weight;
    int unities;

    void GetData();
};
```

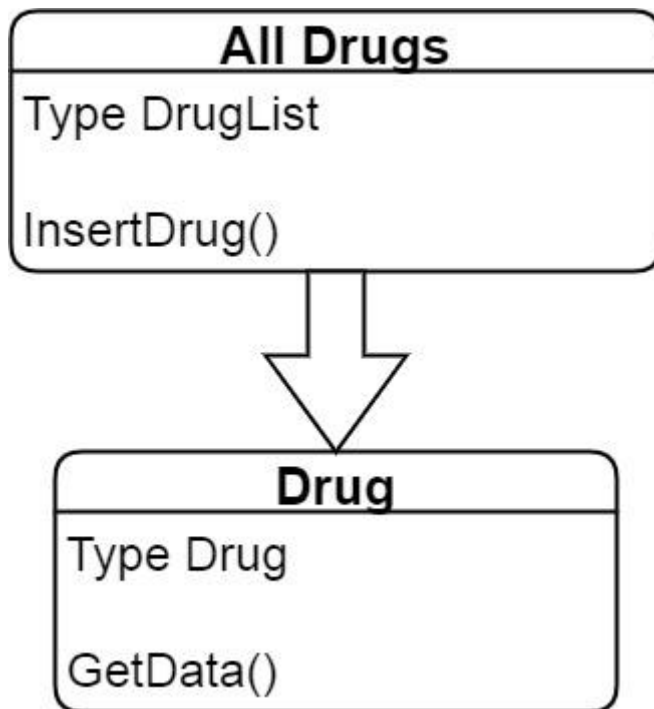


### 2.2.2.- Array de datos tipo *fármaco*.

Examinado el tipo de dato *pedido* la lista de fármacos es un tipo de dato de dato que sí reconoce el lenguaje C++; se trata de un array de datos tipo *fármaco*.



Este tipo de datos, el array de datos tipo *fármaco* se define en un módulo de la forma

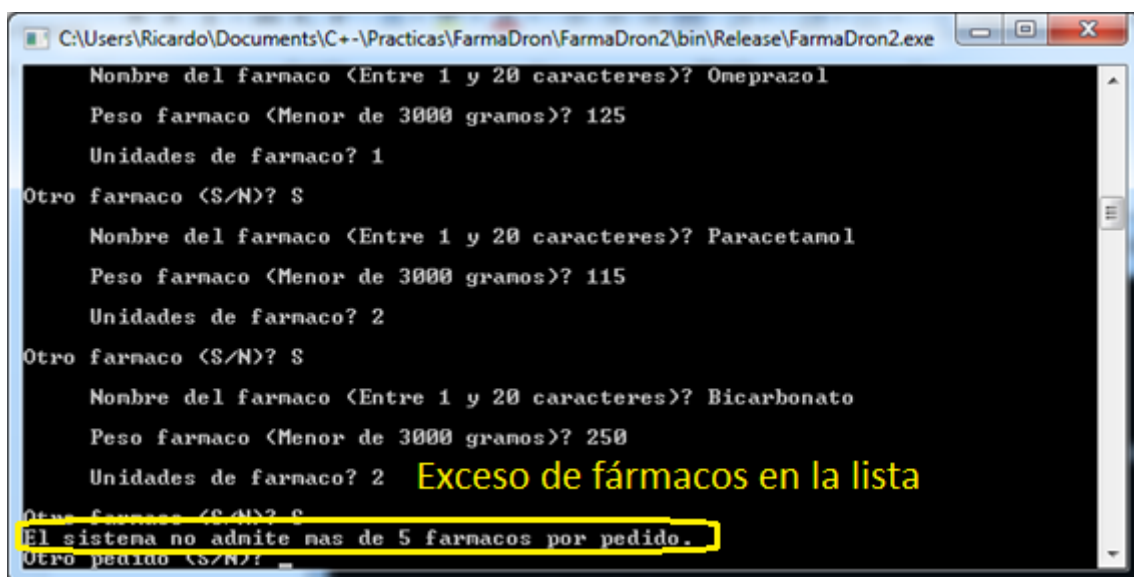
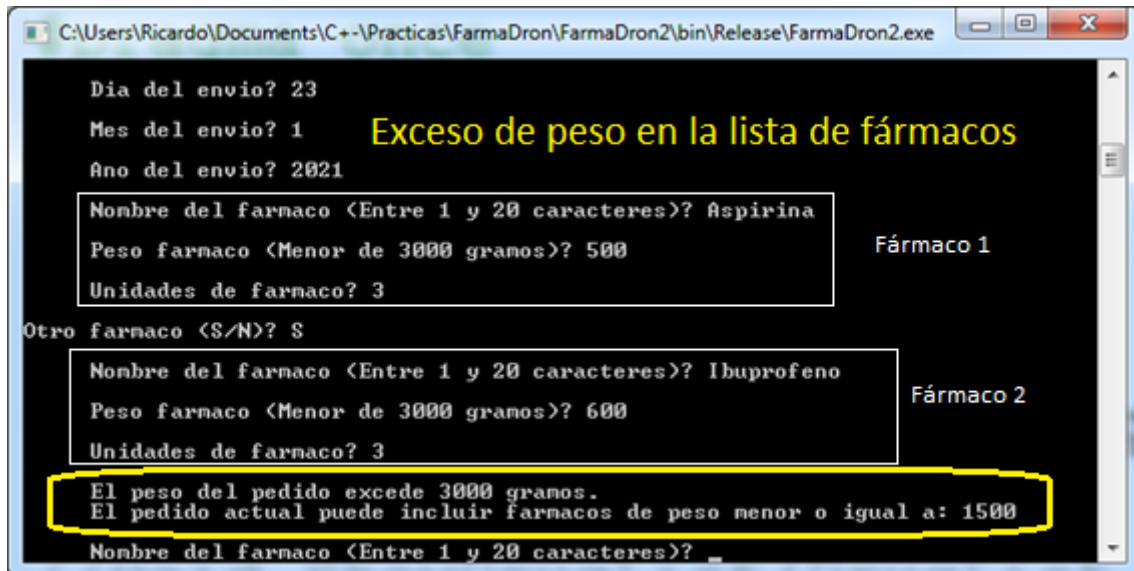


En código fuente

```
*AllDrugs2.h [FarmaDron2] - Code::Blocks 8.02
Archivo Edición Ver Buscar Proyecto Generar Depurar Herramientas Prácticas Complementos Configuración Ayuda
Compliar target: Release
Drug2.h | FarmaDron2.cpp | Drug2.cpp | Order2.cpp | Order2.h | OrderDates2.cpp | OrderDates2.h | Patient2.h | AllDrugs.cpp | *AllDrugs2.h x
1  #pragma once
2  #include "Drug2.h"
3
4  typedef Drug DrugList[5];
5
6  void InsertDrug(DrugList drugList);
7
8
9

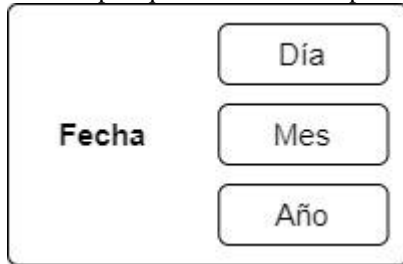
C:\Users\Ricardo\Documents\C++\Prácticas\FarmaDron\FarmaDron2\AllDrugs2.h  WINDOWS-1252  Línea 8, Columna 1  Insertar  Modificado  Lectura/Escritura  ccuned
```

La función que se declara en este módulo, **InsertDrug** registra los fármacos del pedido evitando que el peso total de los fármacos del pedido supere el peso que puede transportar el dron y advirtiendo al usuario cuando el número de fármacos es superior a cinco, máxima capacidad fijada en la aplicación.



### 2.2.3.- Fecha.

Los campos que identifican el tipo de dato *fecha* son: día, mes y año.



Identificando cada uno de los campos que identifican el tipo de dato *fecha*:

- **Día** → Número entero. Tipo de dato reconocido por C++.
- **Mes** → Dato de tipo enumerado.

Captura de pantalla del editor de código Code::Blocks 8.02. Se muestra el archivo `Date2.h` con la siguiente definición de enumeración:

```
4 typedef enum Month
5 {
6     january,
7     february,
8     march,
9     april,
10    may,
11    june,
12    july,
13    august,
14    september,
15    october,
16    november,
17    december
18 };
19
```

La barra de estado indica: C:\Users\Ricardo\Documents\C++\Practicas\FarmaDron\FarmaDron2\Date2.h, WINDOWS-1252, Línea 10, Columna 7.

- **Año** → Número entero. Tipo de dato reconocido por C++.

Cualquier variable del tipo *Fecha* que se cree tendrá las funciones:

- **ValidateDate.** Chequea que los datos con los que se inicializa la variable tipo *fecha* tengan sentido en el formato de una fecha.
- **NewDate.** La función inicializa la variable del dato tipo *fecha* preguntando por pantalla los datos de cada uno de los campos al usuario, discriminando fechas que no tengan sentido.
  1. **SendDayMonthError.** Advierte al usuario mediante un mensaje por pantalla de que la fecha introducida no es correcta.

```
Otro pedido (S/N)? S
Pedido 2

Ref. Paciente (entre 1 y 20) 1
Numero de envios? 1
Dia del envio? 40
Mes del envio? 1
Año del envio? 2021
Error. Introdujo dia: 40 y mes: 1.
No se corresponde el numero de días con el mes introducido
Dia del envio? 20
Mes del envio? 30
Año del envio? 2021
Error. Introdujo el mes: 30
Dia del envio?
```

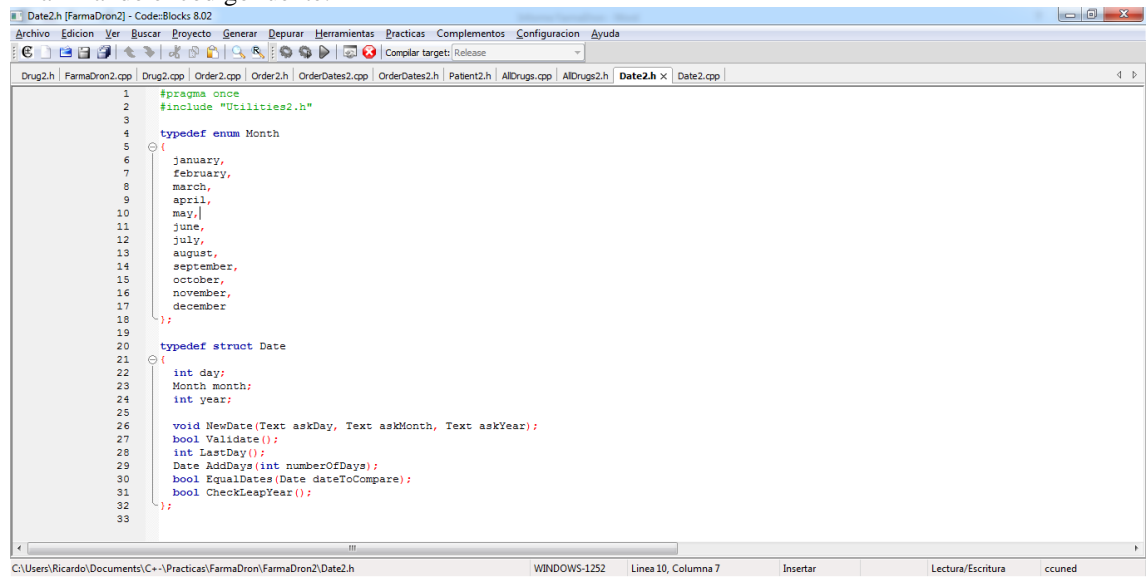
- **LastDay** Calcula el último día del mes de la fecha que llama la función.
- **AddDays**. La función añade el número de días que se le pasan por parámetro a la fecha que contiene la variable devolviendo la fecha que resulta de añadir a la fecha inicial los días del parámetro.
- **EqualDates**. La función valora dos fechas decidiendo si son la misma.
- **CheckLeapYear**. La función valora si el año de la fecha es bisiesto.

El módulo para este tipo de dato será de la forma:



La función recuadrada *–SendDayMonthError–* no es un subprograma que permite definir operaciones sobre datos tipo fecha, es una función que facilita la lectura e interpretación del código fuente en el fichero Date2.cpp

## Examinando el código fuente:



```
1  #pragma once
2  #include "Utilities2.h"
3
4  typedef enum Month
5  {
6      january,
7      february,
8      march,
9      april,
10     may,
11     june,
12     july,
13     august,
14     september,
15     october,
16     november,
17     december
18 };
19
20 typedef struct Date
21 {
22     int day;
23     Month month;
24     int year;
25
26     void NewDate(Text askDay, Text askMonth, Text askYear);
27     bool Validate();
28     int LastDay();
29     Date AddDays(int numberOfDays);
30     bool EqualDates(Date dateToCompare);
31     bool CheckLeapYear();
32 };
33
```

Windows status bar: C:\Users\Ricardo\Documents\C++\Practicas\FarmaDron\FarmaDron2\Date2.h | WINDOWS-1252 | Linea 10, Columna 7 | Insertar | Lectura/Escritura | ccuned

**2.2.4.- Fechas de envío.**

Examinado el tipo de dato *Pedido* el tipo de dato *fechas de envío* es el que ordena las distintas fechas de un pedido, estando estructurado de la forma

Pedidos

Referencia Paciente

Lista de fármacos

Fechas de envío

Pedidos

Referencia Paciente

Lista de fármacos

Fármaco 1

Fármaco 2

Fármaco 3

Fármaco 4

Fármaco 5

Fechas de envío

Número de envíos

Días entre envíos

Fecha del primer envío

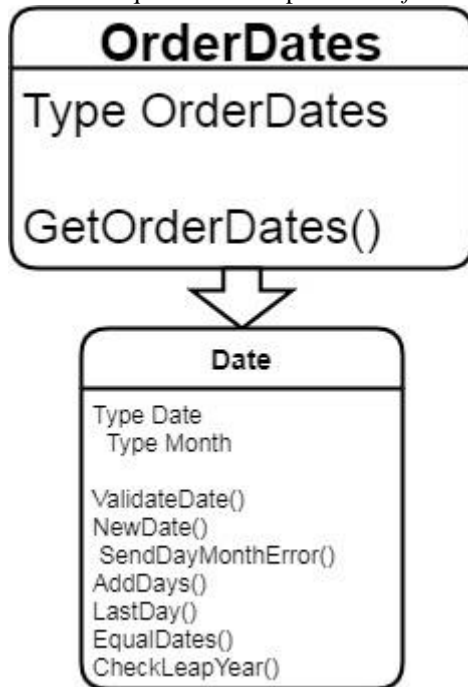
Fecha

Día

Mes

Año

El módulo que define al tipo de dato *fechas de envío* es de la forma



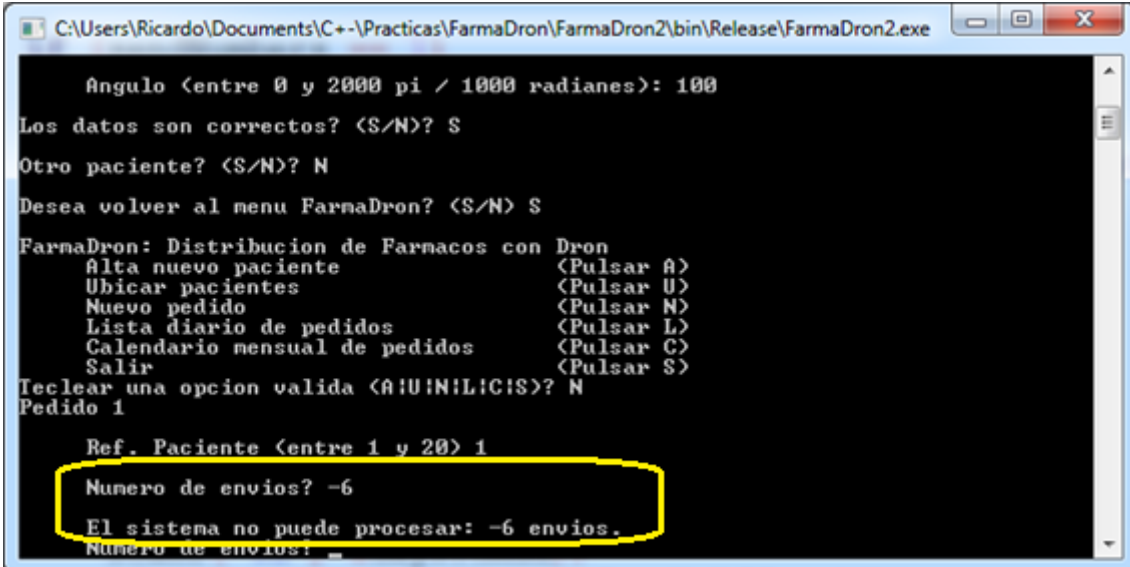
Examinando el código fuente:

```
1 #pragma once
2 #include "Date2.h"
3
4 typedef struct OrderDates
5 {
6     int sendNumbers;
7     int daysToAdd;
8     Date firstSendingDay;
9
10    void GetOrderDates ();
11 };
12
```

The screenshot shows a code editor window titled "OrderDates2.h [FarmaDron2] - Code::Blocks 8.02". The editor displays the source code for the **OrderDates** struct. The code includes a **#pragma once** directive, an **#include "Date2.h"** statement, and a **typedef struct OrderDates** definition. The struct contains three members: **int sendNumbers**, **int daysToAdd**, and **Date firstSendingDay**. It also includes a **void GetOrderDates ()** method. The code is enclosed in a **{ ... }** block and ends with a semicolon. The editor's status bar at the bottom shows the file path, window title, and line/column information.



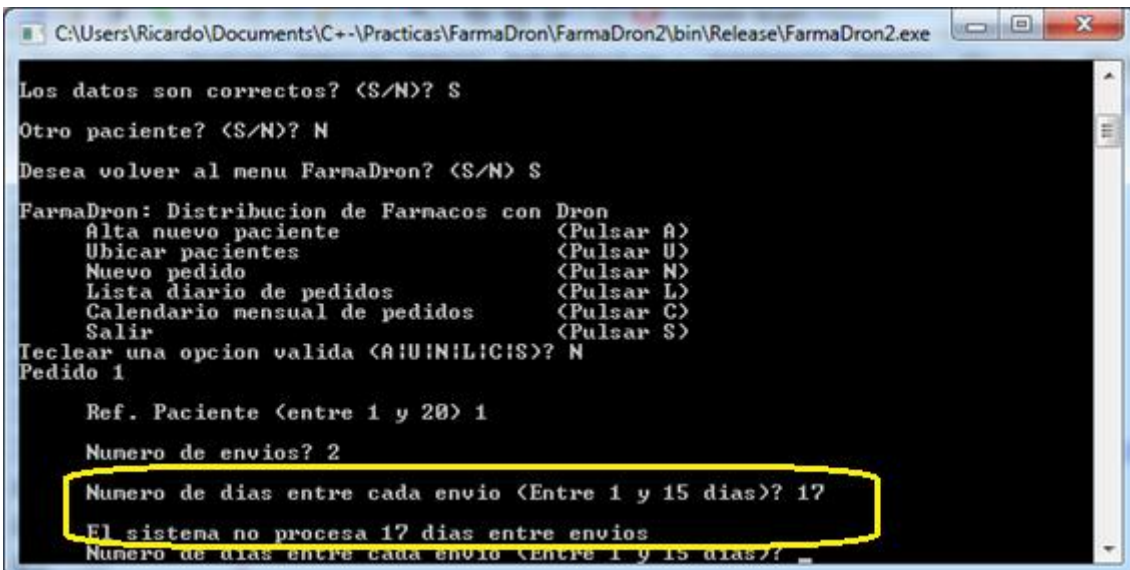
La función **GetOrderDates** inicializa las variables del tipo de dato *fechas de envío* discriminando los valores que tengan sentido en el contexto de la aplicación para los campos del número de envíos y que los días entre los envíos no sean superiores a 15.



```
C:\Users\Ricardo\Documents\C++\Practicas\FarnaDron\FarnaDron2\bin\Release\FarnaDron2.exe

Angulo <entre 0 y 2000 pi / 1000 radianes>: 100
Los datos son correctos? <S/N>? S
Otro paciente? <S/N>? N
Desea volver al menu FarnaDron? <S/N> S
FarnaDron: Distribucion de Farnacos con Dron
Alta nuevo paciente          <Pulsar A>
Ubicar pacientes             <Pulsar U>
Nuevo pedido                 <Pulsar N>
Lista diario de pedidos      <Pulsar L>
Calendario mensual de pedidos <Pulsar C>
Salir                        <Pulsar S>
Teclear una opcion valida <A!U!N!L!C!S!>? N
Pedido 1

Ref. Paciente <entre 1 y 20> 1
Numero de envios? -6
El sistema no puede procesar: -6 envios.
Numero de envios? _
```

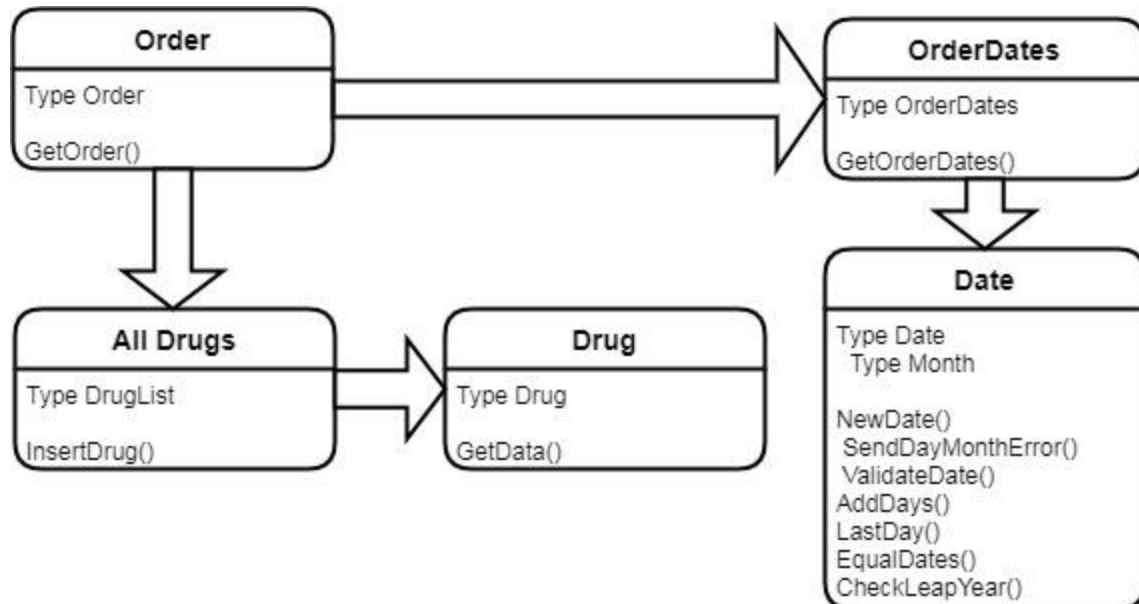


```
C:\Users\Ricardo\Documents\C++\Practicas\FarnaDron\FarnaDron2\bin\Release\FarnaDron2.exe

Los datos son correctos? <S/N>? S
Otro paciente? <S/N>? N
Desea volver al menu FarnaDron? <S/N> S
FarnaDron: Distribucion de Farnacos con Dron
Alta nuevo paciente          <Pulsar A>
Ubicar pacientes             <Pulsar U>
Nuevo pedido                 <Pulsar N>
Lista diario de pedidos      <Pulsar L>
Calendario mensual de pedidos <Pulsar C>
Salir                        <Pulsar S>
Teclear una opcion valida <A!U!N!L!C!S!>? N
Pedido 1

Ref. Paciente <entre 1 y 20> 1
Numero de envios? 2
Numero de dias entre cada envio <Entre 1 y 15 dias>? 17
El sistema no procesa 17 dias entre envios
Numero de dias entre cada envio <Entre 1 y 15 dias>? _
```

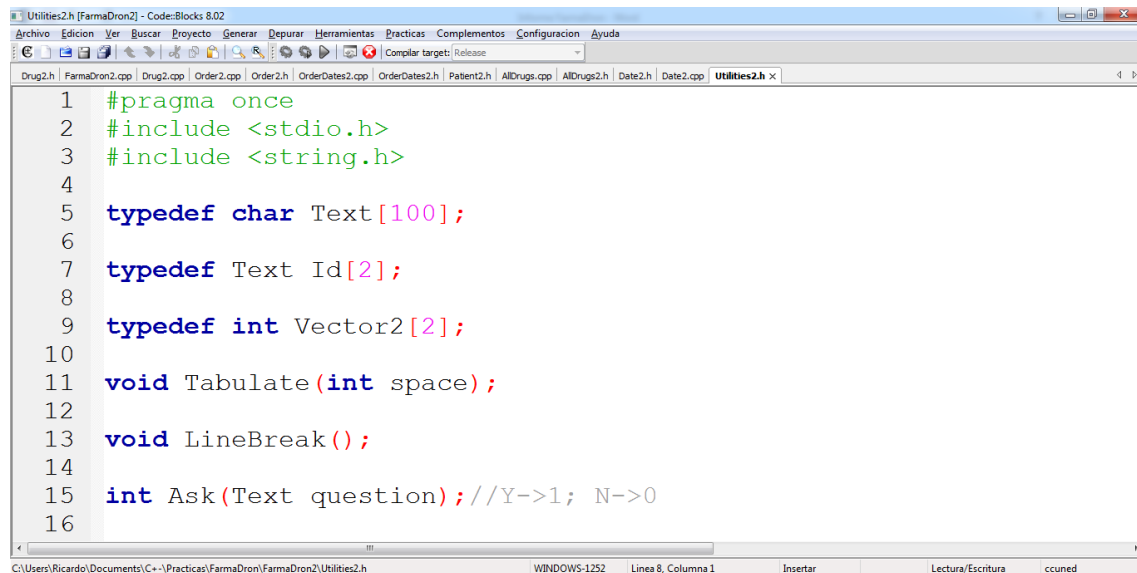
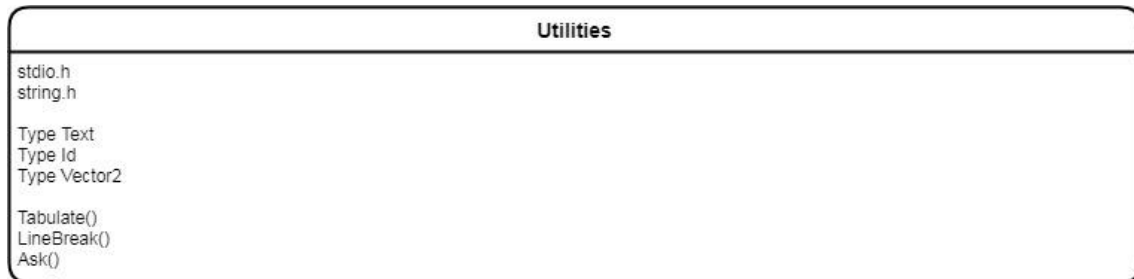
En este punto, el análisis detallado de los tipos de datos y funciones que se encuentran en el bloque dedicado de los datos de tipo *pedido* del bloque de datos es completo y se puede tener una visión general de los módulos que forma en bloque de los datos tipo *pedido* y como están conectados entre ellos.



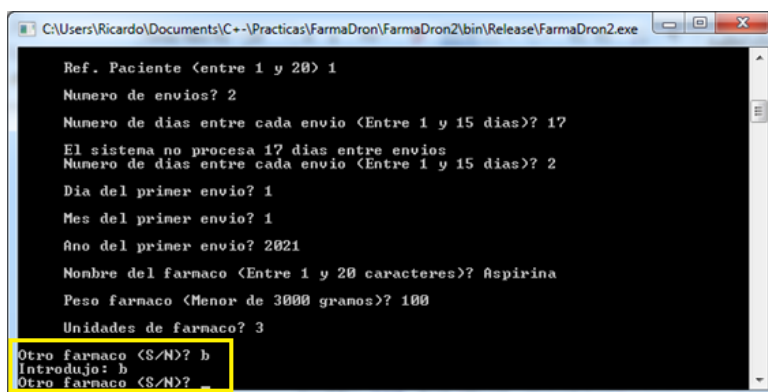
Como se vio a lo largo del análisis del bloque, en esta parte de la aplicación, no sólo se crea la estructura del tipo de dato *pedido*, sino que además se cuida de que los datos que registren las variables tenga sentido en el contexto de la aplicación.

## 2.3.- Servicios

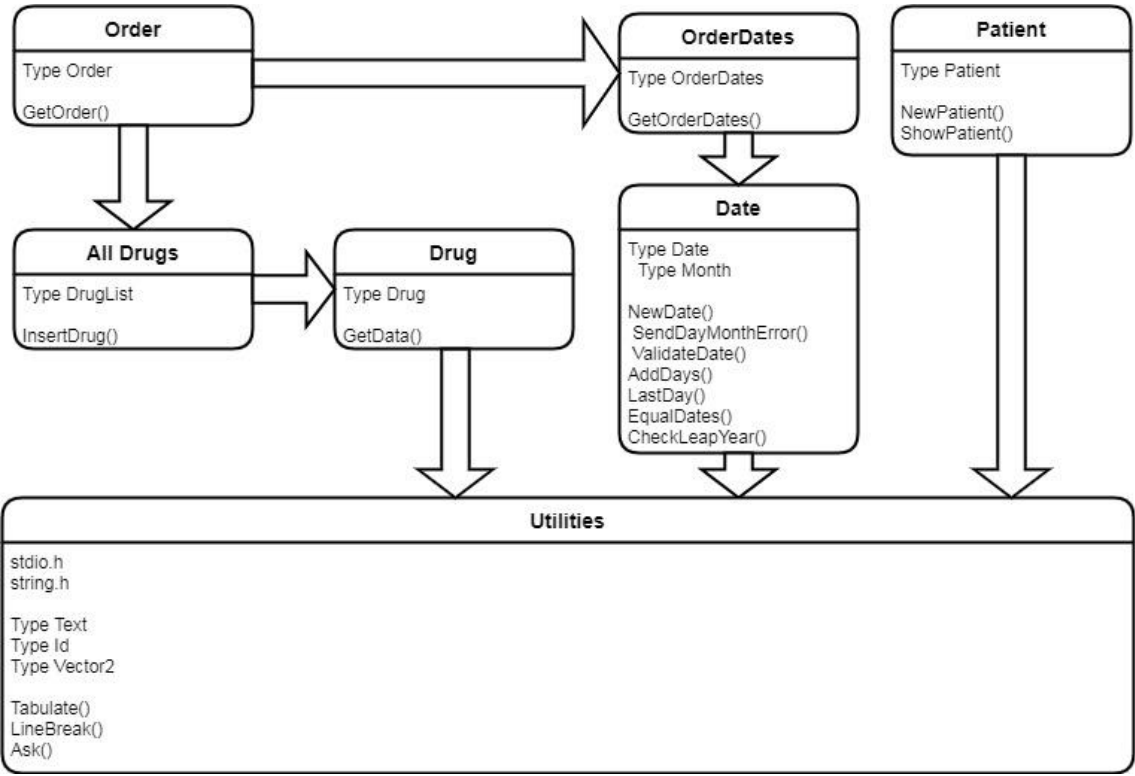
Algunos datos y funciones, demasiado básicos como para tener interés en el contexto de la aplicación, así como las importaciones de la librerías que permiten tratar con cadenas de caracteres y pedir y mostrar datos por pantalla se agrupan en un módulo de servicios al que recurrirán los módulos interesados.



- **Tabulate.** Función que sirve para dejar el número de espacios en blanco que se pasa por parámetro a la hora de escribir por pantalla.
- **LineBreak.** Salto de línea escribiendo por pantalla.
- **Ask.** Escribirá por pantalla la pregunta que se le pase en el texto por parámetro. Genera un uno si la respuesta del usuario es sí y un cero si se respondió no. Discrimina respuestas que salgan de las opciones sí o no.



Con la presentación del módulo servicios, los módulos del bloque de *Datos* de la aplicación quedarían de la forma.



### 3.- Segundo Bloque: Gestión de los datos.

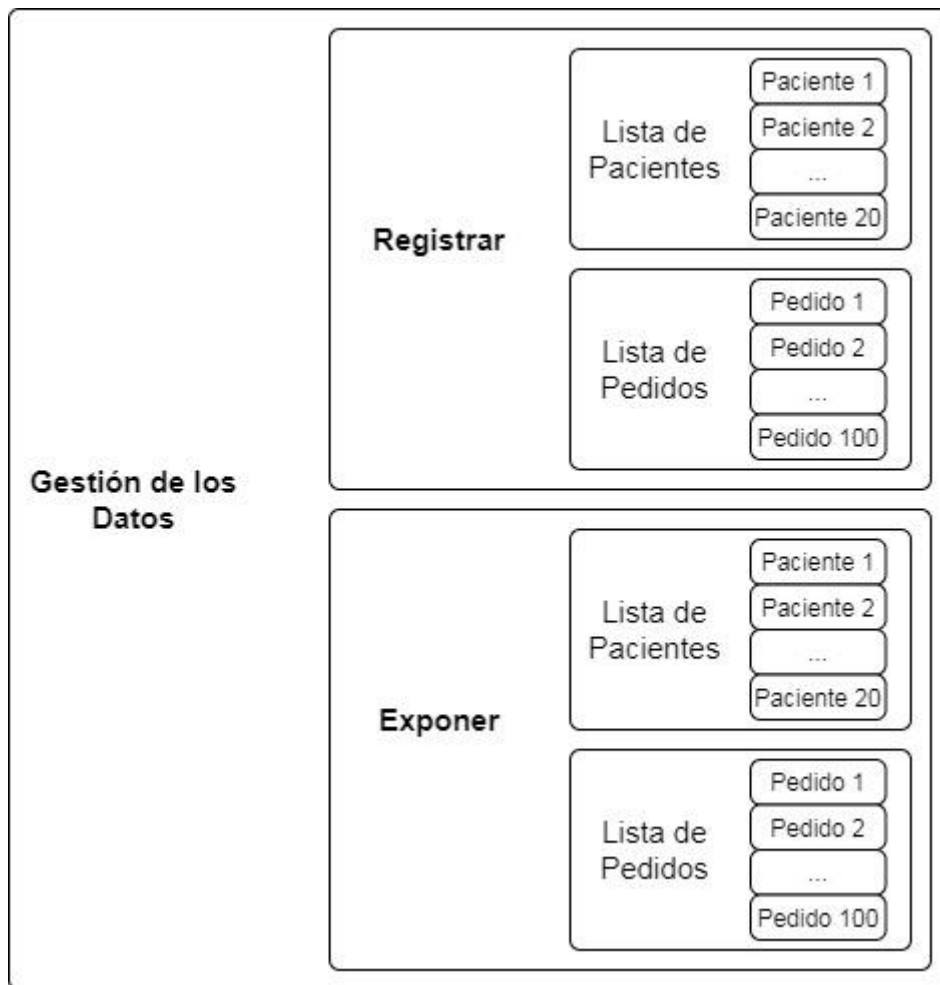
Las funciones de la aplicación se pueden separar en dos categorías: registrar datos y exponen los datos registrados; atendiendo a los tipos de datos que se definieron en el bloque dedicado a los datos, estas dos funciones se pueden clasificar según el tipo de dato que registran o exponen.



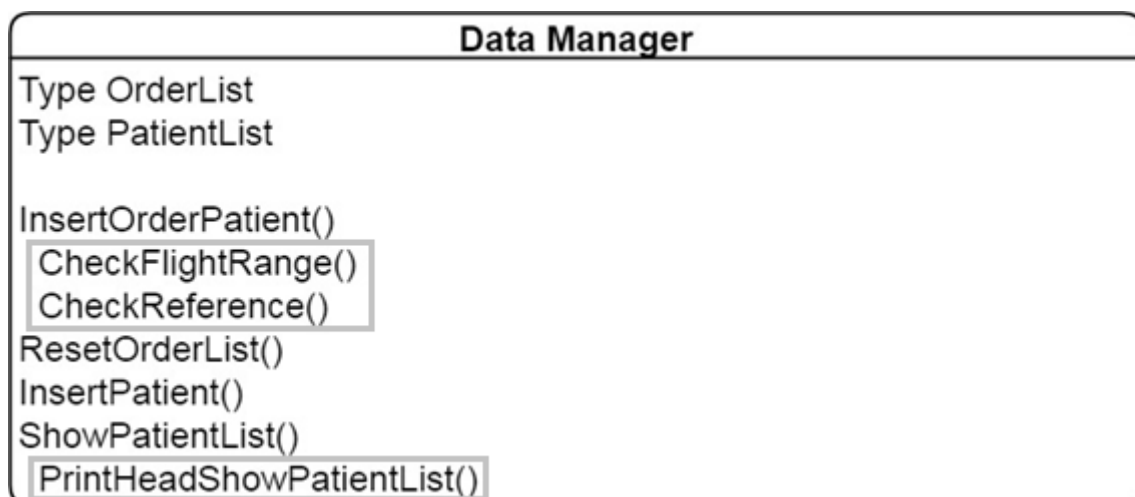
En el bloque de datos se crea el tipo de dato, la estructura que permite a la aplicación, reconocer y trabajar con pacientes y pedidos.

Para registrar un paciente o un pedido, será suficiente con crear una variable del tipo *paciente* o del tipo *pedido* e inicializarla con la información que se desea registrar.

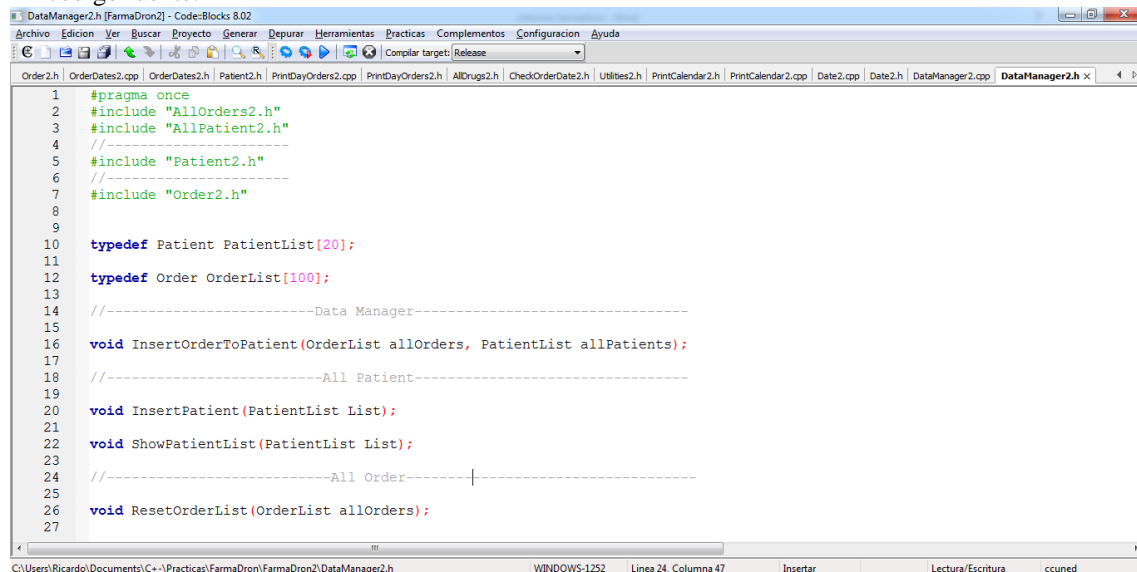
En la aplicación se trabaja con un número fijo de pacientes y de pedidos con lo que, así como se hizo con los fármacos en el tipo de dato *pedido*, los pacientes y pedidos se guardarán en listas ordenadas.



El módulo que define los tipos de datos ordenados de pacientes y pedidos es de la forma



En código fuente:



```
1  #pragma once
2  #include "AllOrders2.h"
3  #include "AllPatient2.h"
4  //-----
5  #include "Patient2.h"
6  //-----
7  #include "Order2.h"
8
9
10 typedef Patient PatientList[20];
11
12 typedef Order OrderList[100];
13
14 //-----Data Manager-----
15
16 void InsertOrderToPatient(OrderList allOrders, PatientList allPatients);
17
18 //-----All Patient-----
19
20 void InsertPatient(PatientList List);
21
22 void ShowPatientList(PatientList List);
23
24 //-----All Order-----
25
26 void ResetOrderList(OrderList allOrders);
27
```

- **ResetOrderList.** Inicializa todas las variables a valores que no tienen sentido en el contexto de la aplicación. Equivale a un reseteo de la lista de pedidos, cuando la aplicación encuentra en uno de los pedidos valores iguales a los del reseteo, tratará la variable como si estuviera vacía.

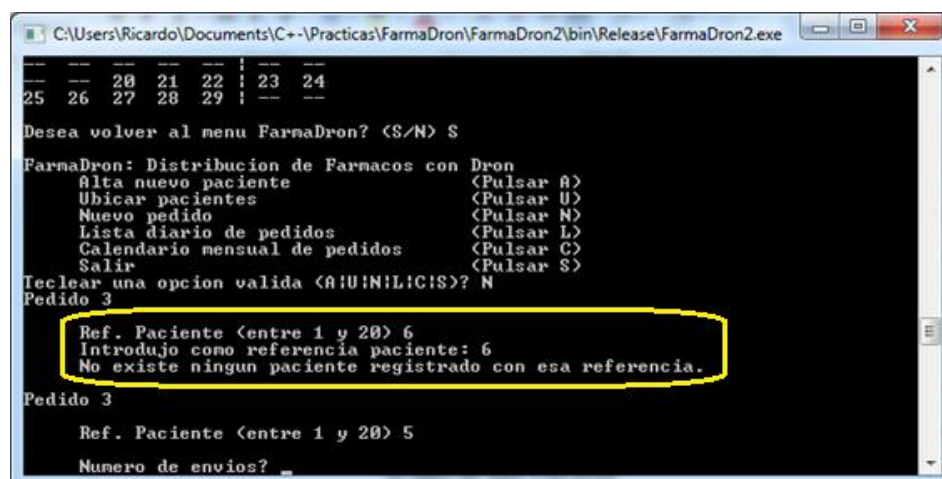
### 3.1.- Registro de los pacientes.

- **InsertPatient.** La función recorre la lista ordenada de pacientes inicializando las variables que no contienen datos. Para inicializar una variable es suficiente con las funcione que se declararon en el dato de tipo *paciente*.

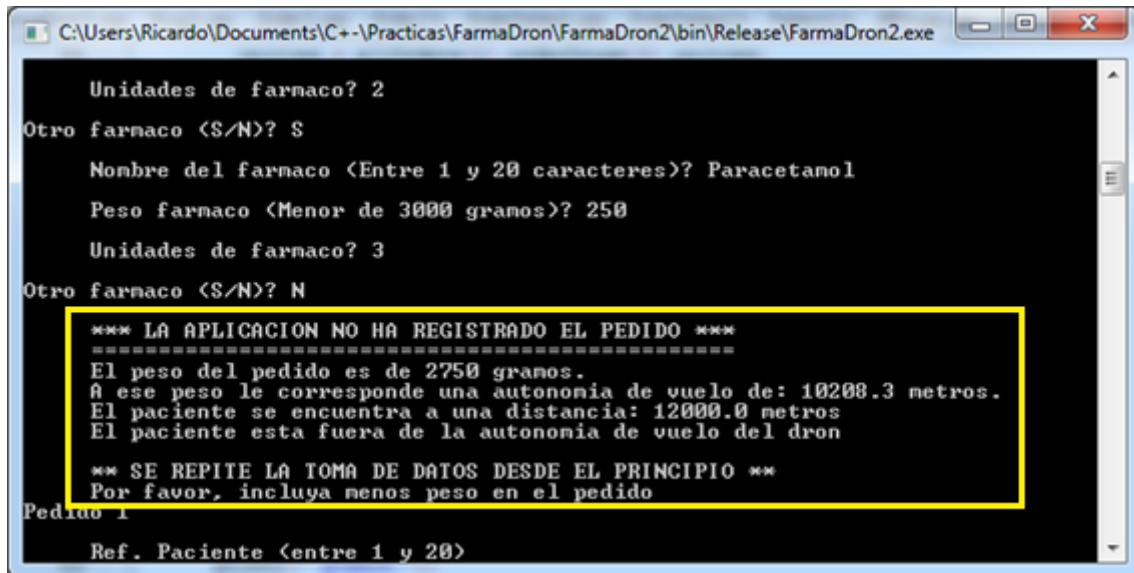
### 3.2.- Registro de los pedidos

- **InsertOrderToPatient.** A la hora de registra los pedidos, algunos campos del dato tipo *Paciente* y del dato tipo *Pedido* tienen relación. Está función comprueba que las relaciones entre los distintos campos son correctas.

1. **CheckReference.** Comprueba la referencia de paciente al que se envía el pedido contiene datos y no se está enviando el pedido a pacientes inexistentes.



2. **CheckFlightRange.** Las distancias a las que se encuentran algunos pacientes pueden estar dentro del radio de acción del dron, pero no del radio de acción del dron para cualquier peso que tenga el pedido. Esta función comprueba siguiendo una sencilla relación lineal ( $autonomía = (25000 - 5/3 \times peso\ total\ del\ pedido) / 2$ ) entre peso del pedido y autonomía del dron que los datos sean correctos.



```
C:\Users\Ricardo\Documents\C+-\Practicas\FarmaDron\FarmaDron2\bin\Release\FarmaDron2.exe

Unidades de farmaco? 2
Otro farmaco (S/N)? S
Nombre del farmaco (Entre 1 y 20 caracteres)? Paracetamol
Peso farmaco (Menor de 3000 gramos)? 250
Unidades de farmaco? 3
Otro farmaco (S/N)? N

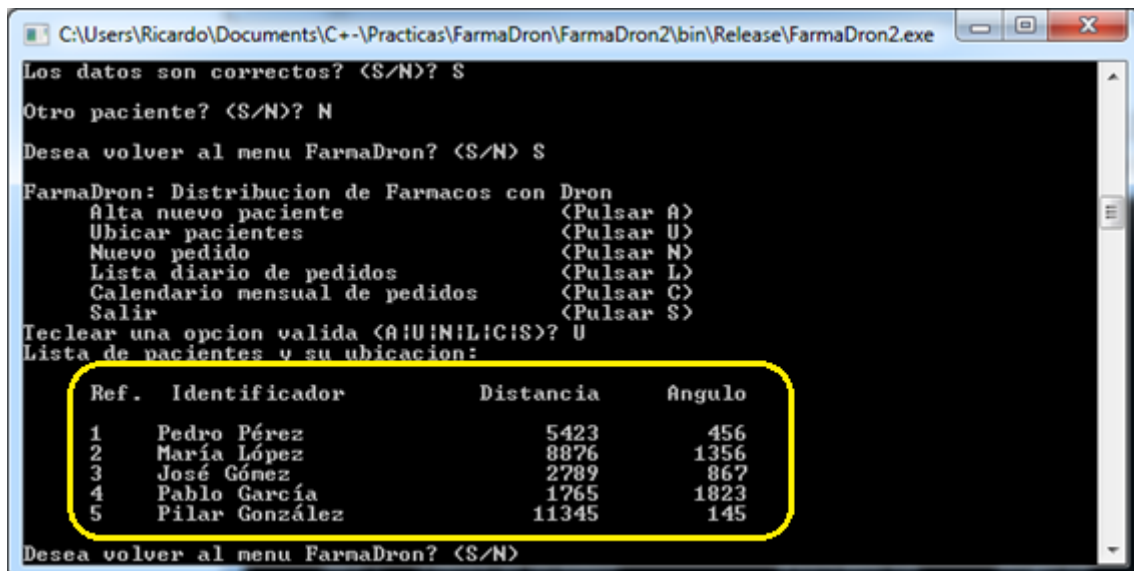
*** LA APLICACION NO HA REGISTRADO EL PEDIDO ***
=====
El peso del pedido es de 2750 gramos.
A ese peso le corresponde una autonomia de vuelo de: 10208.3 metros.
El paciente se encuentra a una distancia: 12000.0 metros
El paciente esta fuera de la autonomia de vuelo del dron

** SE REPITE LA TOMA DE DATOS DESDE EL PRINCIPIO **
Por favor, incluya menos peso en el pedido

Pedido 1
Ref. Paciente (entre 1 y 20)
```

### 3.2.- Exponer pacientes.

- **ShowPatientList.** Esta función expone todos los pacientes que están registrados en la aplicación.
  1. **PrintHeadShowPatientList.** Imprime un encabezamiento que da formato a la exposición de los datos.



```
C:\Users\Ricardo\Documents\C+-\Practicas\FarmaDron\FarmaDron2\bin\Release\FarmaDron2.exe

Los datos son correctos? (S/N)? S
Otro paciente? (S/N)? N
Desea volver al menu FarmaDron? (S/N) S

FarmaDron: Distribucion de Farmacos con Dron
Alta nuevo paciente (Pulsar A)
Ubicar pacientes (Pulsar U)
Nuevo pedido (Pulsar N)
Lista diario de pedidos (Pulsar L)
Calendario mensual de pedidos (Pulsar C)
Salir (Pulsar S)
Teclear una opcion valida (A;U;N;L;C;S)? U
Lista de pacientes y su ubicacion:

Ref.  Identificador      Distancia      Angulo
1     Pedro Pérez         5423           456
2     María López          8876           1356
3     José Gómez            2789           867
4     Pablo García           1765           1823
5     Pilar González         11345          145

Desea volver al menu FarmaDron? (S/N)
```



### 3.3.- Exponer pedidos.

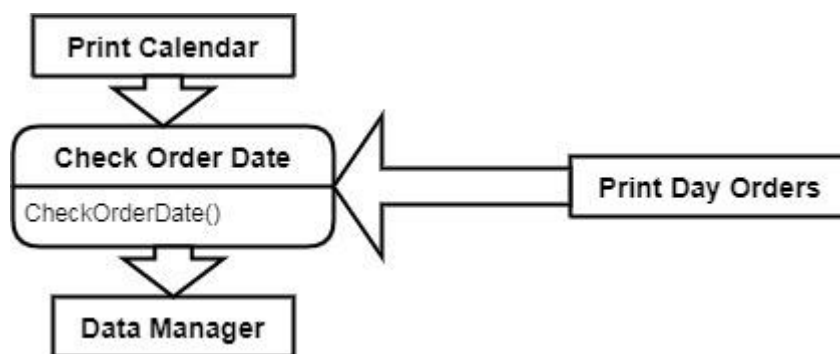
Los datos de los pedidos no se exponen con el mismo formato que se registran sino en función de uno de sus campos, la fecha.

Esta gestión se compone así de dos comandos distintos e independientes:

- Mostrar la lista de pedidos de un día. **Print Day Orders.**
- Mostrar los días del mes en los que hay pedido. **Print Calendar.**

Que estos comandos expongan los datos del pedido depende de que el pedido se envíe al paciente en la fecha que el comando consulta.

La relación de los módulos es:



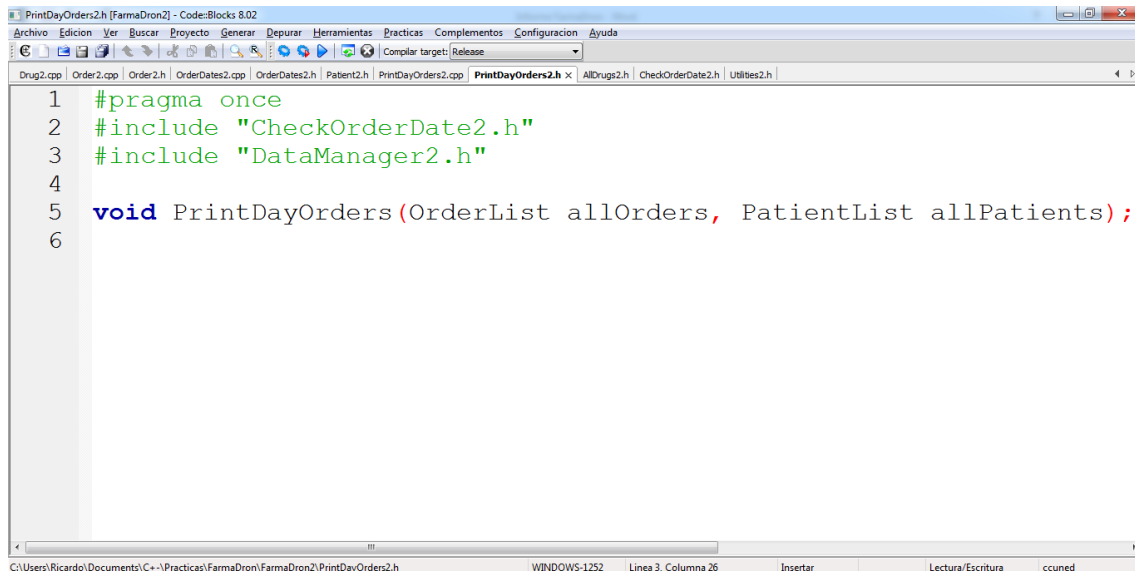
```
1 #pragma once
2 #include "DataManager2.h"
3
4 bool CheckOrderDate(Date aDate, Order anOrder);
5
```

- **CheckOrderDate.** La función evalúa un pedido y una fecha e indica si el pedido será enviado al paciente en la fecha de la consulta. En el contexto de la aplicación, es una función que será consultada por las dos gestiones que exponen los pedidos.

### 3.3.1.- Mostrar la lista de pedidos de un día.

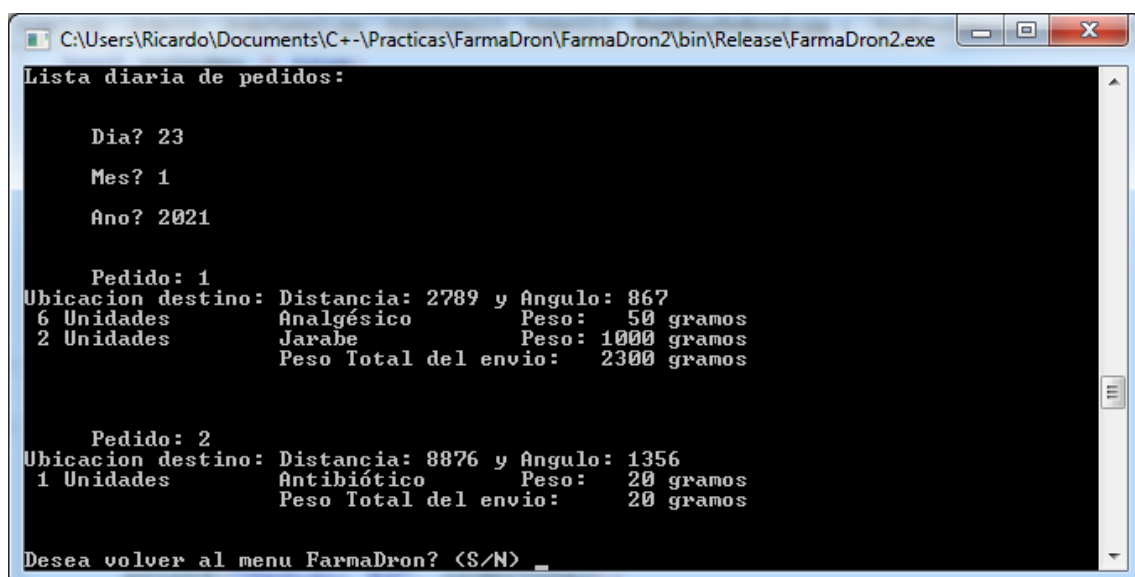
## Print Day Orders

### PrintDayOrder()



```
1 #pragma once
2 #include "CheckOrderDate2.h"
3 #include "DataManager2.h"
4
5 void PrintDayOrders(OrderList allOrders, PatientList allPatients);
6
```

- **PrintDayOrder.** La función pide por pantalla una fecha al usuario y recorre la lista de pedidos buscando la fecha entre ellos. Si algún pedido debe ser enviado en la fecha de la consulta, expone los datos del pedido y el paciente al que serán enviados los medicamentos con un determinado formato.



```
Lista diaria de pedidos:

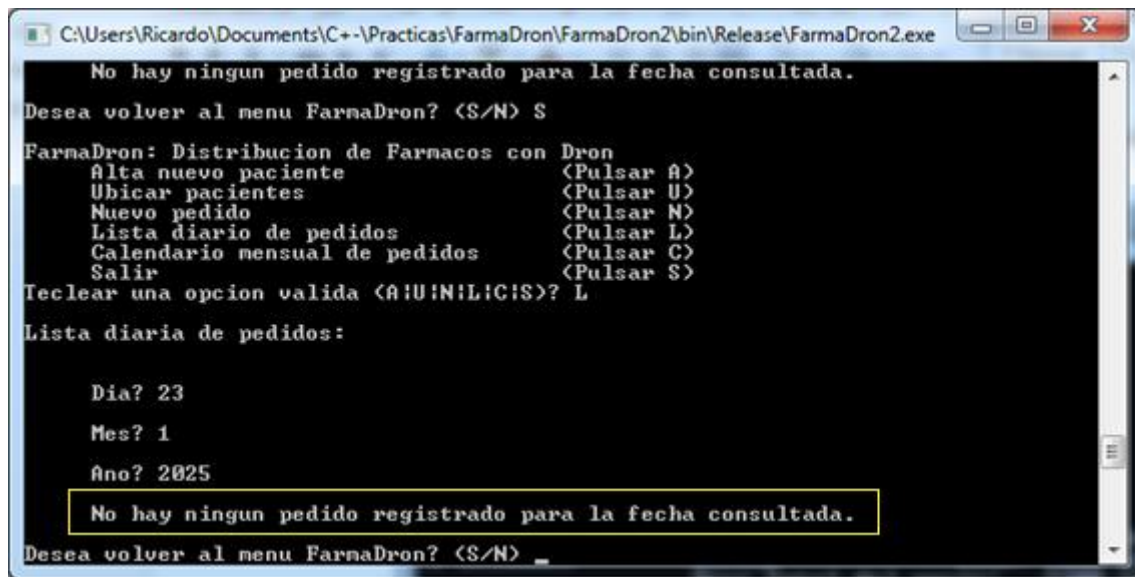
Dia? 23
Mes? 1
Año? 2021

Pedido: 1
Ubicacion destino: Distancia: 2789 y Angulo: 867
6 Unidades Analgésico Peso: 50 gramos
2 Unidades Jarabe Peso: 1000 gramos
Peso Total del envío: 2300 gramos

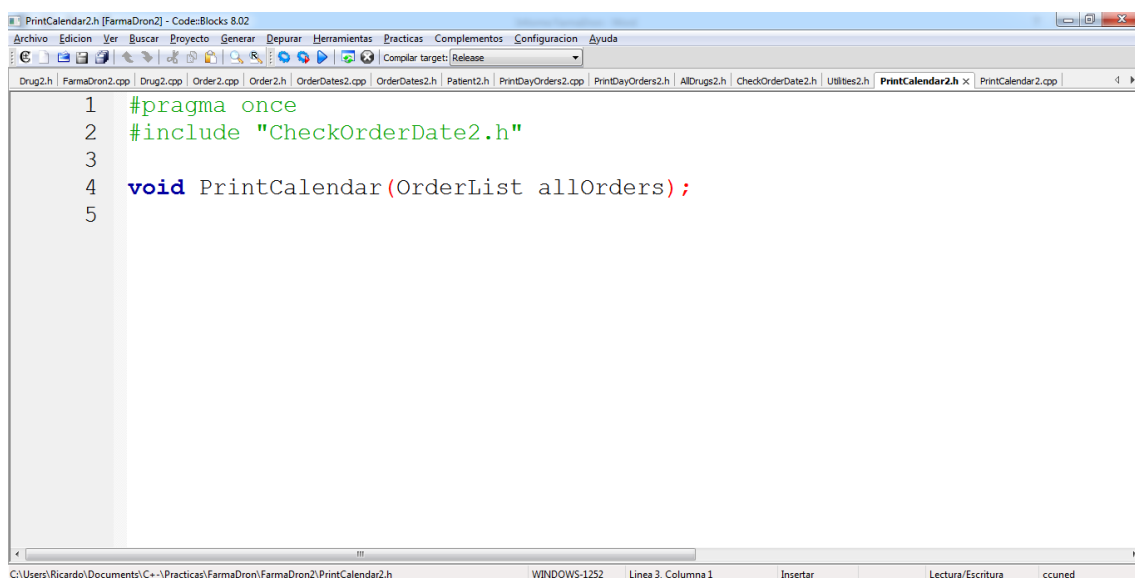
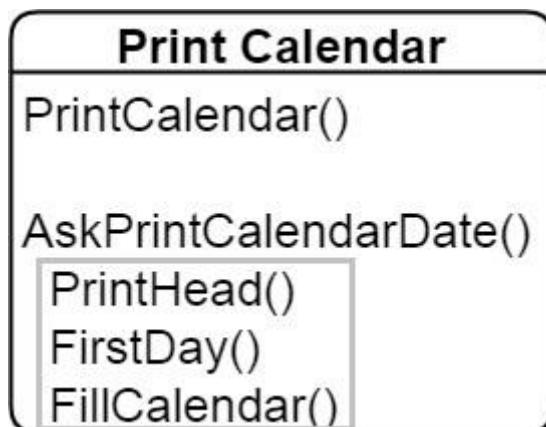
Pedido: 2
Ubicacion destino: Distancia: 8876 y Angulo: 1356
1 Unidades Antibiótico Peso: 20 gramos
Peso Total del envío: 20 gramos

Desea volver al menu FarmaDron? <S/N> _
```

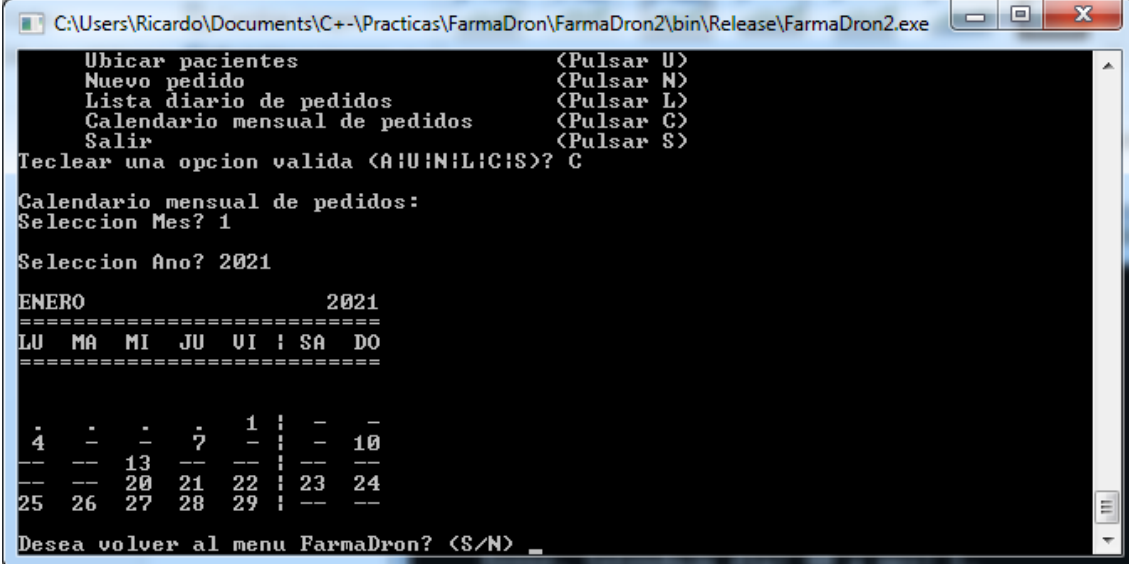
En caso de que en la fecha consultada no haya envíos programados, la aplicación avisará al usuario



### 3.3.2.- Mostrar los días del mes en los que hay pedido.



- **PrintCalendar.** Se trata de un módulo que imprime un mes con un formato de calendario con la salvedad de que en lugar de escribir siempre el número del día, lo hará sólo si tras consultar la función **CheckOrderDate** sabe que ese día hay programado el envío de un pedido.



```

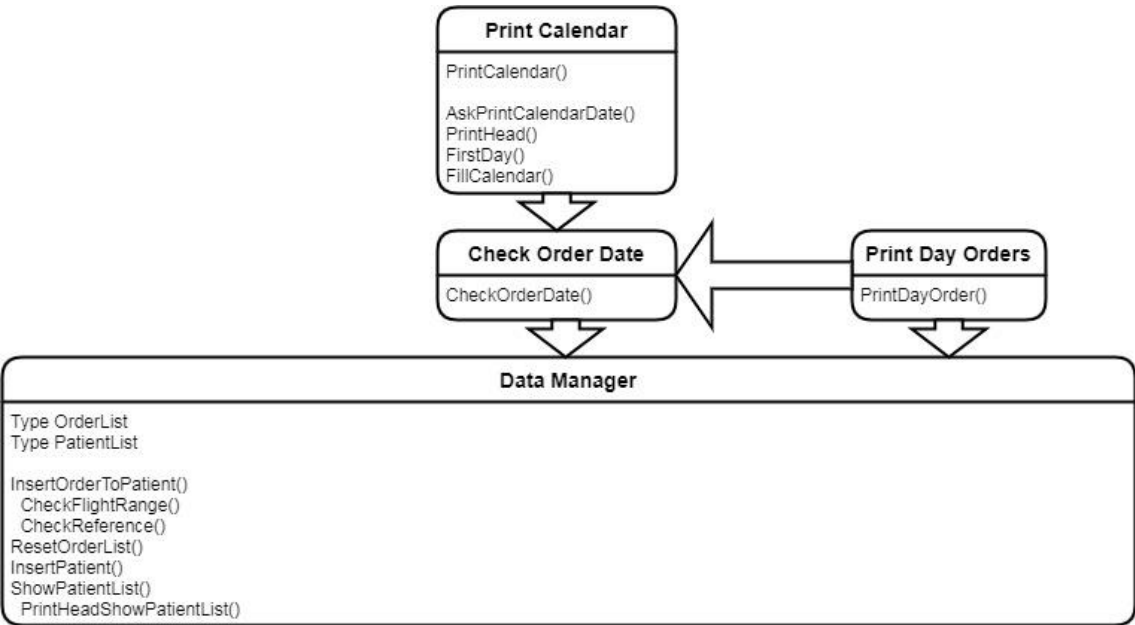
C:\Users\Ricardo\Documents\C+-\Practicas\FarmaDron\FarmaDron2\bin\Release\FarmaDron2.exe
Ubicar pacientes                <Pulsar U>
Nuevo pedido                   <Pulsar N>
Lista diario de pedidos        <Pulsar L>
Calendario mensual de pedidos <Pulsar C>
Salir                          <Pulsar S>
Teclear una opcion valida <A|U|N|L|C|S>? C

Calendario mensual de pedidos:
Seleccion Mes? 1
Seleccion Ano? 2021

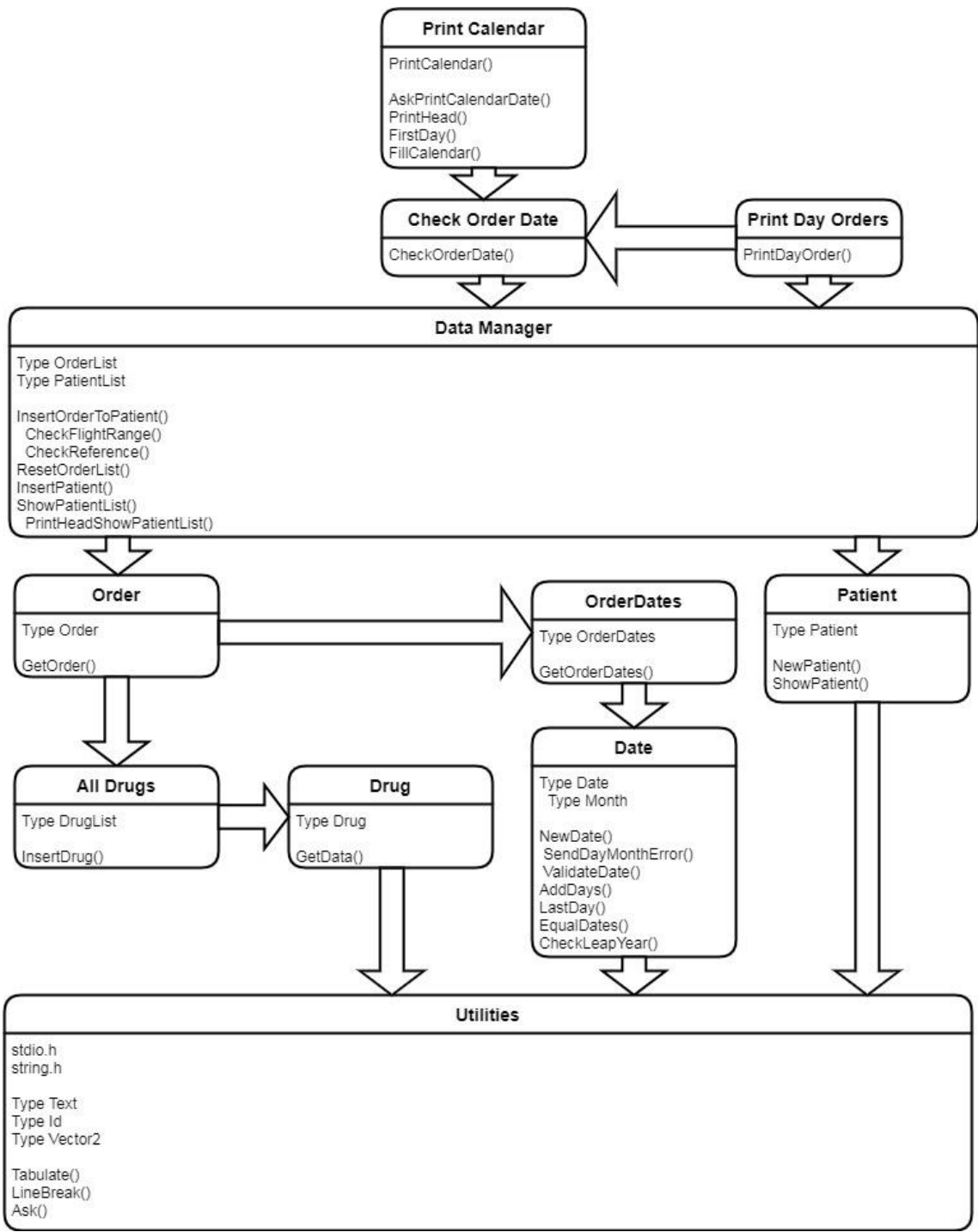
ENERO                          2021
=====
LU MA MI JU VI SA DO
=====
  4  -  -  7  1  -  -
-- -- 13 -- -- 10
-- -- 20 21 22 23 24
25 26 27 28 29 - -
Desea volver al menu FarmaDron? <S/N> _

```

De esta forma, los módulos que componen el bloque de gestión de los datos quedan conectados

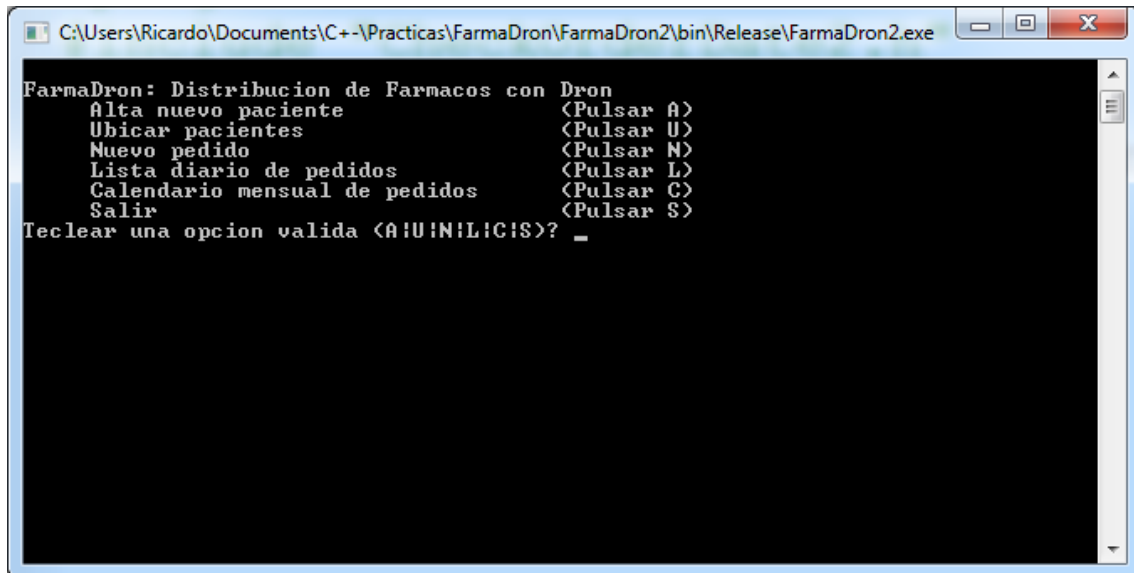


La conexión entre los módulos del bloque de datos y del bloque de gestión es



## 4.- Interfaz.

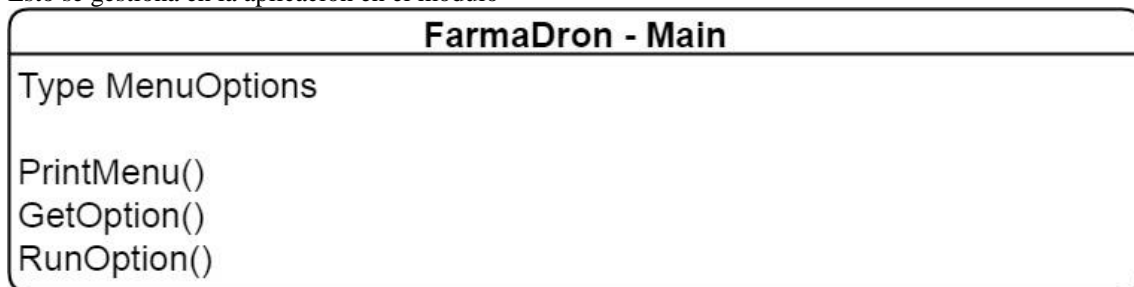
La aplicación ofrece al usuario las opciones de las que dispone a través de un menú y recibe las instrucciones del usuario a través del teclado.



```
C:\Users\Ricardo\Documents\C++\Practicas\FarmaDron\FarmaDron2\bin\Release\FarmaDron2.exe

FarmaDron: Distribucion de Farmacos con Dron
Alta nuevo paciente          <Pulsar A>
Ubicar pacientes             <Pulsar U>
Nuevo pedido                 <Pulsar N>
Lista diario de pedidos      <Pulsar L>
Calendario mensual de pedidos <Pulsar C>
Salir                        <Pulsar S>
Teclear una opcion valida <A!U!N!L!C!S!>? _
```

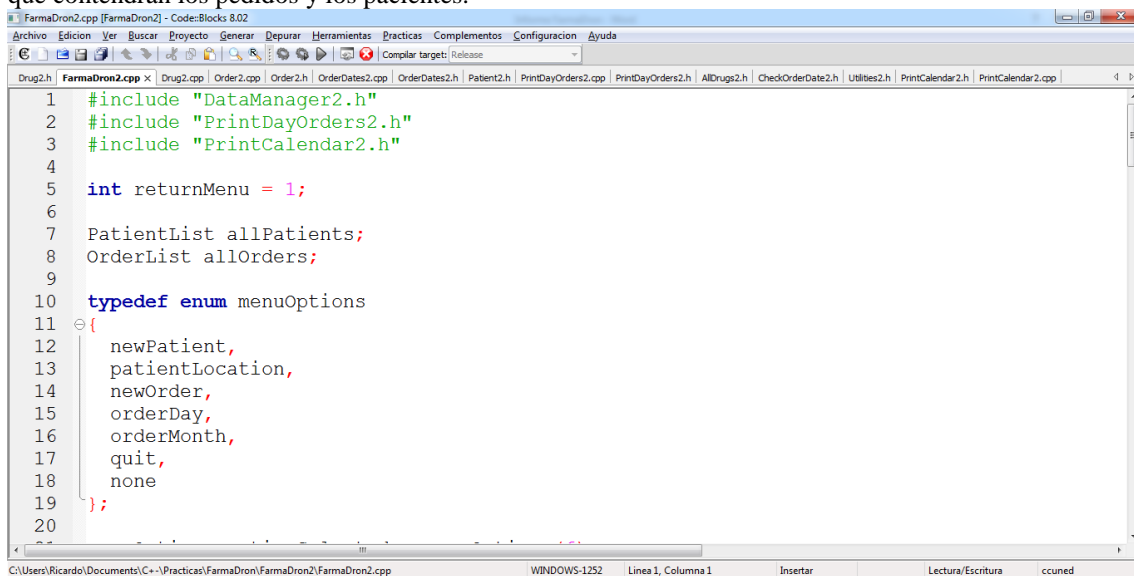
Esto se gestiona en la aplicación en el módulo



```

FarmaDron - Main
Type MenuOptions
PrintMenu()
GetOption()
RunOption()
```

En este módulo se declaran las funciones que ofrece el menú como tipos de datos enumerados y las variables que contendrán los pedidos y los pacientes.



```
FarmaDron2.cpp [FarmaDron2] - Code::Blocks 8.02
Archivo Edición Ver Buscar Proyecto Generar Depurar Herramientas Practicas Complementos Configuración Ayuda
C:\Users\Ricardo\Documents\C++\Practicas\FarmaDron\FarmaDron2\FarmaDron2.cpp
1 #include "DataManager2.h"
2 #include "PrintDayOrders2.h"
3 #include "PrintCalendar2.h"
4
5 int returnMenu = 1;
6
7 PatientList allPatients;
8 OrderList allOrders;
9
10 typedef enum menuOptions
11 {
12     newPatient,
13     patientLocation,
14     newOrder,
15     orderDay,
16     orderMonth,
17     quit,
18     none
19 };
20
```

- **PrintMenu.** Imprime el menú de la aplicación de acuerdo al formato pedido.
- **GetOption.** Almacena la opción seleccionada en una variable. Si no se selecciona una opción válida advierte al usuario.

The screenshot shows a Windows application window titled "C:\Users\Ricardo\Documents\C++\Practicas\FarmaDron\FarmaDron2\bin\Release\FarmaDron2.exe". The application displays a menu titled "FarmaDron: Distribucion de Farnacos con Dron" with the following options and their corresponding key presses:

Alta nuevo paciente	<Pulsar A>
Ubicar pacientes	<Pulsar U>
Nuevo pedido	<Pulsar N>
Lista diario de pedidos	<Pulsar L>
Calendario mensual de pedidos	<Pulsar C>
Salir	<Pulsar S>

Below the menu, the user is prompted: "Teclear una opcion valida <A!U!N!L!C!S>? X". The user has entered 'X', and the application displays the message: "No elegio ninguna opcion del menu." followed by "Desea volver al menu FarmaDron? <S/N>".

- **RunOption.** Según la opción seleccionada, llamará a la función que la aplicación necesite en cada caso.

El código fuente de la función principal de la aplicación es

The screenshot shows the source code of the FarmaDron2 application in Code::Blocks 8.02. The code is in C++ and is located in the file "FarmaDron2.cpp". The main function is defined as follows:

```

112 }
113
114 int main()
115 {
116     ResetOrderList(allOrders);
117     while(returnMenu==1)
118     {
119         optionSelected = GetOption();
120         RunOption(optionSelected, allPatients, allOrders);
121         returnMenu = Ask("Desea volver al menu FarmaDron? (S/N) ");
122     }
123
124     LineBreak();
125     printf("La aplicacion FarmaDron se ha cerrado.");
126     LineBreak();
127 }
128

```

The code is written in C++ and uses standard library functions like `ResetOrderList`, `GetOption`, `RunOption`, and `Ask`. The `main` function is enclosed in a `while` loop that continues as long as `returnMenu` is equal to 1. The `Ask` function is used to prompt the user for input, and the `printf` function is used to display a message when the application closes.



La conexión de los distintos módulos de la aplicación es

