Curso cero de programación

Guillem Borrell Nogueras

8 de septiembre de 2007

1. Introducción

Según Donald E. Knuth, la programación como comunicación con una máquina es un arte. Esta opinión aparentemente exagerada es completamente cierta. No lo es por el hecho que los que programan puedan ser considerados artistas sino porque la programación es una práctica muy parecida a la creación de una obra de arte. Basta con realizar una simple comparación. ¿Qué es lo necesario para pintar un retrato? Pues a parte del artista; un modelo; pinturas, lienzo y pinceles y la capacidad para hacerlo. Resumienedo: algoritmos, herramientas y técnica. Con la ausencia de cualquiera de las anteriores el cuadro sería cualquier cosa menos una obra de arte.

La realización de un programa es parecida; también requere estos tres ingredientes, capacidad de crear el algoritmo, unas herramientas de desarrollo adecuadas y el buen uso de los lenguajes de programación. La enseñanza de la informática suele obviar uno de los tres ingredientes esenciales, las herramientas. La consecuencia es que quien se inicia en un curso de programación necesita horas y horas de lucha con el ordenador completamente innecesarias. Basta con dar a conocer unos pocos conceptos que por desgracia suelen obviarse.

2. ¿Qué es programar?

Programar es, como su propio nombre indica, crear un programa. Todo sería trivial si supieramos con precisión qué es un programa. Windows ha acercado la informática al usuario novel, sin embargo ha escondido conceptos esenciales detrás de ventanas de colores. Programar consta de dos pasos esenciales una vez el código está escrito: compilar y enlazar; en inglés, build and link. Compila el compilador y enlaza el linker.

2.1. ¿Qué es compilar?

Compilar no es generar un ejecutable, un programa. Este es uno de los errores más comunes en los cursos de introducción a la programación. Compilar es crear código *objeto* a partir de código *fuente*. También es simplificar decir que la compilación consta de un único paso pero no necesitamos saber más por ahora.

El compilador no es capaz de transformar cualquier tipo de archivo de código, debe ser una *unidad de programa*. He aquí uno de los conceptos clave de la programación clásica. Una unidad de programa es un código que tiene un

principo y un final claro. En Fortran, el primero de todos los lenguajes de programación de alto nivel, una unidad de programa es un *program*, un *function* o un *subroutine*[1]. *Estas son las unidades de programa que entiende el compilador independientemente, es por ello que cada una de ellas se suele escribir en un archivo y se compila a parte.* Este proceso es independiente de qué se esté usando para programar. Da igual que estemos delante de una interfaz gráfica como el Visual Studio o Eclipse o accediendo al compilador por una consola UNIX [2]. Por mucha ventana que tengamos delante el compilador no es más que una pequeña aplicación que recibe un archivo y devuelve otro. Cuando en una interfaz gráfica seleccionamos la opción *build* estamos generando los archivos de código objeto correspondientes.

2.2. Compilar una unidad de programa

Para entenderlo mejor vamos a compilar una unidad de programa, un *subroutine*. Este archivo va a implementar la ecuación diferencial de un tiro parabólico en dos dimensiones:

$$\dot{u} = -cu\sqrt{u^2+v^2} \quad (1)$$

$$\dot{v} = -g - cv\sqrt{u^2+v^2} \quad (2)$$

Obviamente, para encontrar la trayectoria tendremos en cuenta las dos ecuaciones adicionales $\dot{x} = u$ y $\dot{y} = v$.

Para resolver el problema de la forma

$$\dot{\vec{x}} = F(\vec{x}, t) \quad (3)$$

necesitaremos convertir las ecuaciones anteriores en algo de la forma $F(\vec{x}, t)$ y escribirlo en Fortran 95. *Recordemos que ahora no nos interesa el algoritmo, sólo entender qué hace el compilador con la función que escribamos*

```
subroutine tiro_parabolico(neq,t,y,ydot)
!!  tiro_parabolico.f90
!!  Ecuacion del tiro parabolico con rozamiento
!!  en dos dimensiones
!!  Se hacen cosas raras con las variables para
!!  adaptarse al solver slsode

  integer,intent(in) :: neq
  real,intent(in) :: t
  real,dimension(4),intent(in) :: y
  real,dimension(4),intent(out) :: ydot

!!  constantes del problema
!!    aceleracion de la gravedad
  real,parameter :: g=9.8
!!    coeficiente de rozamiento
  real,parameter :: c=1
```

[1] Dejaremos las palabras *programa*, *subrutina* y *función* para las estructuras no ligadas a un lenguaje de programación en concreto

[2] Nunca hay que olvidar que Windows no es el único sistema operativo, ni siquiera el mejor

```
  ydot(1)=y(3)
  ydot(2)=y(4)
  ydot(3)=-c*y(3)*sqrt(y(3)**2+y(4)**2)
  ydot(4)=-g-c*y(4)*sqrt(y(3)**2+y(4)**2)

end subroutine tiro_parabolico
```

¿Qué tenemos que hacer para compilar esta subrutina? ¿Cómo podemos compilar una subrutina sin haber definido un programa con anterioridad? ¿Cómo podemos hacerlo sin haber empezado un proyecto?

Cada interfaz gráfica y cada compilador lo hace a su manera. Como ejemplo vamos a utilizar el compilador del proyecto GNU, gfortran[3] y el Salford fortran 95 compiler Personal Edition[4]. El primero es un típico compilador UNIX sin interfaz gráfica mientras que el segundo basa todo su funcionamiento en el IDE[5] Plato3. Ambos son gratuitos y muy fáciles de instalar. Aunque los ejemplos se basan en el uso de estos compiladores en concreto todos funcionan de un modo parecido, basta con consultar la documentación.

Para empezar abriremos una consola con: Inicio ->Accesorios ->Símbolo del sistema. Se abrirá el terminal de Windows, eso a lo que los usuarios noveles tienen un miedo irracional. Los comandos son bastante sencillos y siempre es útil saber moverse por ahí. Para aprender a moverse por los directorios basta con este tutorial

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

c:\Documents and Settings\Guillem Borrell\Escritorio> dir
 El volumen de la unidad C no tiene etiqueta.
 El numero de serie del volumen es: 548A-799A

 Directorio de c:\Documents and Settings\Guillem Borrell\Escritorio

13/05/2005  23:03    <DIR>          .
13/05/2005  23:03    <DIR>          ..
29/09/2005  09:03    <DIR>          eclipse
24/07/2005  13:09    <DIR>          texmakerwin32-1.2.1
09/12/2005  12:19               707 SciTE.lnk
11/09/2005  20:26             1.754 peret.lnk
19/03/2006  17:27             1.465 Simbolo del sistema.lnk
               3 archivos          3.926 bytes
               4 dirs  19.424.706.560 bytes libres

c:\Documents and Settings\Guillem Borrell\Escritorio>
```

dir: Lista los archivos y directorios del directorio actual

```
c:\Documents and Settings\Guillem Borrell\Escritorio> cd..
```

[3] `http://gcc.gnu.org/wiki/GFortran`
[4] `http://www.silverfrost.com/11/ftn95/overview.asp`
[5] Integrated Development Environment

```
C:\Documents and Settings\Guillem Borrell> dir

 El volumen de la unidad C no tiene etiqueta.
 El numero de serie del volumen es: 548A-799A

 Directorio de C:\Documents and Settings\Guillem Borrell

13/05/2005  23:03    <DIR>          .
13/05/2005  23:03    <DIR>          ..
13/05/2005  22:21    <DIR>          Menu Inicio
19/03/2006  17:11    <DIR>          Mis documentos
19/03/2006  17:11    <DIR>          Favoritos
15/05/2005  20:56    <DIR>          Escritorio

 ...
C:\Documents and Settings\Guillem Borrell> cd Mis Documentos
C:\Documents and Settings\Guillem Borrell\Mis documentos> dir

 El volumen de la unidad C no tiene etiqueta.
 El numero de serie del volumen es: 548A-799A

 Directorio de C:\Documents and Settings\Guillem Borrell\Mis documentos

13/05/2005  23:03    <DIR>          .
13/05/2005  23:03    <DIR>          ..
19/03/2006  17:11    <DIR>          Mis imagenes
19/03/2006  17:11    <DIR>          Mi musica
26/05/2005  21:32               433 Frame1.py

 ...
```

`cd Mis Documentos`**:** Entra en el directorio Mis Documentos argumento. Un truco intersante es escribir parte del nombre del directorio y apretar la tecla de tabular; automáticamente Windows va a intentar completar el nombre para que no sea necesario introducirlo entero.

`cd..`**:** Baja un nivel en el árbol de directorios

Ahora que ya sabemos movernos por los directorios iremos en el que se encuentre el archivo `tiro_parabolico.f90` del que hemos hablado antes. En mi caso es el directorio `Mis Documentos`. Ahora tecleamos lo siguiente (para abreviar la consola se expresará con el símbolo >)

```
> ftn95
[FTN95/Win32 Ver. 4.8.0 Copyright (C) Salford Software Ltd 1993-2005]
*** No source file specified
*** Usage: ftn95 <file-name> [/option [/option ...]]
*** Or:    ftn95 /? for more information on command-line options.
> gfortran
gfortran: no input files
```

`ftn95`**:** Invoca el compilador Salford Fortran 95

`gfortran`**:** Invoca el compilador GNU Fortran 95

Ahora ya somos capaces de movernos por los directorios y de ejecutar el compilador vamos a convertir nuestro archivo de código fuente en un archivo objeto:

```
>  ftn95 tiro_parabolico.f90
[FTN95/Win32 Ver. 4.8.0 Copyright (C) Salford Software Ltd 1993-2005]
0008)   integer,intent(in) :: neq
WARNING - The argument NEQ has not been used
0009)   real,intent(in) :: t
WARNING - The argument T has not been used
    NO ERRORS, 2 WARNINGS  [<TIRO_PARABOLICO> FTN95/Win32 v4.8.0]
```

Esta es precisamente el output que los IDEs nos ponen en la ventanita de la parte inferior. Nos avisa de que hay dos variables que no hemos utilizado pero que ha sido capaz de crear el archivo objeto. En el caso del compilador de Salford se llamará `tiro_parabolico.obj`. El compilador `gfortran` es un compilador tipo UNIX con lo que la llamada va a ser ligeramente distinta:

```
> gfortran -c tiro_parabolico.f90
```

La opción `-c` pide al compilador que sólo compile, que no intente generar un ejecutable. Vemos además que no nos ha soltado ningún mensaje de aviso aunque deberia haberlo hecho. Si deseamos que el compilador trabaje un poco más con los avisos pasaremos una opción adicional:

```
> gfortran -c -Wall tiro_parabolico.f90
tiro_parabolico.f90: In function 'tiro_parabolico':
tiro_parabolico.f90:1: warning: unused variable 'neq'
tiro_parabolico.f90:1: warning: unused variable 't'
```

En ambos casos el compilador GNU Fortran 95 habrá generado un archivo de código objeto llamado `tiro_parabolico.o`

Nada nos impide poner varias funciones en un mismo archivo de código fuente, tampoco está prohibido poner todo el código de un programa en un único archivo; pero es un buen ejercicio de higiene mental separar todas las unidades de programa para que sean procesadas por separado. Una razón muy importante es porque así sabemos sólo con el nombre del archivo qué unidad de programa contiene.

2.3. Manejar archivos objeto

Los archivos de código compilado o archivos objeto son útiles de modo completamente independiente. Gracias a la existencia de archivos objeto existen las bibliotecas, en inglés libraries. A todo el mundo le suenan esos archivos con la extensión `.dll`. Estas librarías no serían posibles si los procesos de compilación y enlazado no fueran independientes.

Supongamos que nos dicen que no tenemos que escribir la subrutina de integración de la ecuación diferencial, que utilicemos las subrutinas LSODE. Son integradores de paso variable escritos por los desarrolladores del Lawrence Livermore National Laboratory del Departamento de Energía de los Estados Unidos. Además nos dicen que no tenemos que compilarlo, que alguien lo ha hecho previamente con mucho cuidado para que tenga un rendimiento máximo. ¿Qué nos

dan entonces? no nos dan una subrutina llamada `slsode.f` sino archivo llamad `slsode.obj` por si utilizamos el compilador de Salford y otro llamado `slsode.o` por si utilizamos el de GNU. Deben proporcionarnos dos archivos objeto distintos porque los dos compiladores no son compatibles. Podría darse el caso que lo fueran con lo que bastaría con un único archivo objeto[6].

Además nos pasan un manual de cómo utilizar la subrutina y escribimos finalmente el programa que resuelve el problema en `poblematiro.f90`

```
program problematiro

  implicit none

  external tiro_parabolico

!!  Parametros de configuracion de lsode

  integer, parameter :: neq=4
  integer, parameter :: iopt=0
  integer, parameter :: itol=1
  integer, parameter :: itask=1
  integer, dimension(20) :: iwork
  integer :: liw=20
  integer, dimension(20+16*neq) :: rwork
  integer :: lrw=20+16*neq

  integer :: istate=1
  integer :: mf=10 ! Adams nonstiff method

!!  Configuracion del problema
  integer,parameter :: nitmax=1000

!!  Errores
  real :: rtol=1.e-6
  real :: atol=1.e-6
  real, dimension(4) :: y
  real :: t,tout,dt=.1

  integer :: i

  y(1)=0.
  y(2)=0.
  y(3)=10.
  y(4)=10.

  write(*,*) 'coord. x - ','coord. y - ','veloc. u - ','veloc. v - ','tiempo'
  do i=1,nitmax
     tout=t+dt
```

[6] Existen una clase especial de librerías llamadas librerías estáticas. No son más que archivos objeto empaquetados. Las librerías dinámicas tienen un poco más de arte porque no se acoplan al ejecutable en el enlazado sino en tiempo de ejecución

```
     call slsode(tiro_parabolico,neq,y,t,tout,itol,rtol,atol,itask,&
          &      istate,iopt,rwork,lrw,iwork,liw,'dummy',mf)
     write(*,*) y(1),y(2),y(3),y(4),t
!!     Condicion de parada es la llegada al suelo
     if (y(2)<0) exit
  end do

end program problematiro
```

Una vez hayamos convertido esta unidad de programa tipo *program* en otro archivo de código objeto ya tenemos todo lo necesario para crear el programa que resuelve el tiro parabólico

2.4. Enlazado

Una vez se han creado los archivos objeto de `slsode`, `tiro_parabolico` y `problematiro` tenemos que enlazarlos con un *linker* para crear el ejecutable.

2.4.1. GNU Fortran 95

Para realizar todo el proceso de compilación y enlazado con el compilador GNU Fortran abrimos una consola y nos situamos donde están todos los archivos de código fuente y escribimos los siguientes comandos:

```
> gfortran -c tiro_parabolico.f90 slsode.f problematiro.f90
 In file slsode.f:3593

      IF (INCX .EQ. INCY) IF (INCX-1) 5,20,60
                                                                      1
Warning: Obsolete: arithmetic IF statement at (1)
 In file slsode.f:3683

      IF (INCX .EQ. INCY) IF (INCX-1) 5,20,60
                                                                      1
Warning: Obsolete: arithmetic IF statement at (1)

> gfortran tiro_parabolico.o slsode.o problematiro.o
> a.exe

 coord. x - coord. y - veloc. u - veloc. v - tiempo
  0.9931067      0.9443293       9.863879       8.890494      0.1000000
   1.973107       1.778831       9.737710       7.802993      0.2000000
   2.940949       2.505577       9.620543       6.734952      0.3000000
   3.897485       3.126395       9.511436       5.684084      0.4000000
   4.843474       3.642899       9.409443       4.648350      0.5000000
   5.779579       4.056512       9.313605       3.625965      0.6000000
   6.706368       4.368490       9.222939       2.615413      0.7000000
   7.624307       4.579950       9.136432       1.615462      0.8000001
   8.533762       4.691906       9.053059      0.6251828      0.9000001
   9.434990       4.705294       8.971790     -0.3560317       1.000000
   10.32815       4.620996       8.891625      -1.328465       1.100000
```

```
   11.21331       4.439896       8.811617      -2.292088       1.200000
   12.09044       4.162890       8.730906      -3.246585       1.300000
   12.95944       3.790915       8.648735      -4.191391       1.400000
   13.82011       3.324977       8.564475      -5.125751       1.500000
   14.67224       2.766164       8.477622      -6.048762       1.600000
   15.51554       2.115657       8.387791      -6.959434       1.700000
   16.34969       1.374742       8.294710      -7.856720       1.800000
   17.17437      0.5448111       8.198214      -8.739567       1.900000
   17.98923     -0.3726401       8.098217      -9.606936       2.000000
```

2.4.2. Salford FTN95

Para conseguir exactamente el mismo propósto con el compilador de Salford Software:

```
> ftn95 tiro_parabolico.f90
...
> ftn95 slsode.f
...
> ftn95 problematiro.f95
...
> slink tiro_parabolico.obj slsode.obj problematiro.obj -out:a.exe
...
> a.exe
 coord. x - coord. y - veloc. u - veloc. v - tiempo
     0.993107        0.944329         9.86388         8.89049        0.100000
      1.97311         1.77883         9.73771         7.80299        0.200000
      2.94095         2.50558         9.62054         6.73495        0.300000
      3.89749         3.12639         9.51144         5.68408        0.400000
      4.84347         3.64290         9.40944         4.64835        0.500000
      5.77958         4.05651         9.31361         3.62596        0.600000
      6.70637         4.36849         9.22294         2.61541        0.700000
      7.62431         4.57995         9.13643         1.61546        0.800000
      8.53376         4.69191         9.05306        0.625182        0.900000
      9.43499         4.70529         8.97179       -0.356033         1.00000
      10.3282         4.62099         8.89163        -1.32847         1.10000
      11.2133         4.43989         8.81162        -2.29209         1.20000
      12.0904         4.16289         8.73091        -3.24659         1.30000
      12.9594         3.79091         8.64874        -4.19139         1.40000
      13.8201         3.32498         8.56448        -5.12575         1.50000
      14.6722         2.76616         8.47762        -6.04877         1.60000
      15.5155         2.11566         8.38779        -6.95944         1.70000
      16.3497         1.37474         8.29471        -7.85673         1.80000
      17.1744        0.544809         8.19821        -8.73957         1.90000
      17.9892       -0.372643         8.09822        -9.60694         2.00000
```

Como podemos apreciar, la herramienta de Salford diferencia entre el compilador y el enlazador; a diferencia del compilador GNU. También es apreciable que no nos podemos fiar del formato libre, cada compilador tiene el suyo.

3. La buena programación

Programar bien es complicado y requiere mucha experiencia. Lo que sí es siempre cierto es que los IDEs suponen un freno a las capacidades de un programador. Una interfaz gráfico es una herramienta pensada para ser sencilla y rápida, pero no para ser potente.

Utilizar un editor como emacs, VIM o Nedit tiene una curva de aprendizaje salvajemente amplia pero no por ello es menos interesante dominar por lo menos uno de ellos. Del mismo modo comunicarse con el compilador y el enlazador desde el intérprete de comandos, ya sea el de Windows como el de UNIX/Linux no parece la forma más adecuada para un principiante pero da una medida real del control que se puede llegar a tener.

4. Módulos

Los módulos son una pequeña invención de Fortran 90. Son una extensión a las librerías estáticas para hacer que los programas sean más versátiles, pequeños y polivalentes. Sin poder definir verdaderos módulos es imposible practicar la programación modular. Pero son una herramienta para un paradigma de programación. No es unaobligación utilizarlos.

Un módulo es como una librería pero con código fuente en vez de código objeto. Esto otorga más potencia al desarrollador y permite que los ejecutables sean más rápidos porque compila todo el programa con las mismas opciones y el mismo compilador. Una vez sabemos que todas las funciones y subrutinas funcionan como es debido las encapsulamos en un módulo.

Esto ayuda a tener una idea intuitiva de por qué un módulo debe ser un archivo a parte. Su propio nombre lo dice, un módulo es algo independiente que se enlaza como si fuera algo adicional.

5. Una sugerenncia

Conseguir el mismo ejecutable mediante un IDE como Visual Studio o plato3