Introducción a la programación en Pascal

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad Rey Juan Carlos

gsyc-profes (arroba) gsyc.urjc.es

Septiembre de 2018

Contenidos

Programación de Ordenadores

- Programar es darle instrucciones a un ordenador
- Un ordenador recibe datos y genera nuevos datos según las instrucciones del programa
- El programa también son datos, se puede cambiar tan fácilmente como el resto de datos, lo que permite que el ordenador sea de uso general

Hardware y Software

- Hardware: parte tangible.
- Software: intangible.
  - Programas de usuario
  - Sistema Operativo (*Operating System*) : programa intermedio entre el hardware y los programas de usuario. Acrónimo: SO, OS.
    Ejemplos: Microsoft Windows, Mac OS, Linux, Android, iOS, ...

Ejecutar un programa: pedirle al ordenador (al sistema operativo) que siga sus instrucciones.

Componentes de un ordenador

- CPU. *Central Processing Unit*
- Memoria principal
- Memoria secundaria
- Dispositivos de entrada/salida

CPU

Acrónimo de *Central Processing Unit*

- En español normalmente decimos CPU, pero también se usa *unidad central de procesamiento*, *unidad de procesamiento central*
- Wikipedia lo define como *Hardware dentro de un ordenador u otros dispositivos programables, que interpreta las instrucciones de un programa informático mediante la realización de las operaciones básicas aritméticas, lógicas y de entrada/salida del sistema*

Atención: a veces se le da un significado distinto: se llama CPU a la *caja* principal del ordenador (la *torre*), que contiene memoria, disco y lo que aquí llamamos CPU

- En los ordenadores actuales, las CPU más frecuentes son las basadas en la arquitectura X86_64: los Intel i3, i5, i7, y sus equivalentes AMD
  - Los hay más sencillos como Intel Pentium, y más potentes como Intel i9
- Los teléfonos móviles modernos son verdaderos ordenadores. Suelen usar una arquitectura diferente, ARM. Son más sencillos, baratos, consumen menos y se calientan menos
  - Distintos fabricantes: Qualcomm, Nvidia, Samsung, Huawei,...

Memoria Principal

- Normalmente se le llama *memoria*, a secas. También memoria primaria o memoria interna
- Programa y datos están en memoria principal
- La memoria principal es volátil (se borra al apagar), por tanto es necesario almacenar en memoria secundaria.
- Puede ser de varios tipos: RAM, ROM...

Memoria Secundaria

- Normalmente se le llama simplemente *disco*
- Puede ser un disco duro mecánico tradicional, (HDD, *hard disk drive*), un disco de estado sólido (SDD, *Solid State Drive*), un pendrive, etc. En épocas anteriores lo habitual fue el cdrom, los floppy disk, diversos tipos de cintas magnéticas, o, en los orígenes, tarjetas de cartulina perforadas
- La memoria secundaria es mucho más lenta que la memoria principal. Pero tiene mucha mayor capacidad, es más barata y sobre todo, no es volátil (si manejamos los dispositivos debidamente)
- La CPU no trabaja directamente sobre memoria secundaria [1], programas y datos de entrada se leen desde disco hasta memoria. Los datos de salida se escriben desde memoria hasta disco

[1] salvo en alguna excepción

- La información en el disco se organiza en ficheros y directorios
  - *fichero* y *directorio* (file / directory) son los términos informáticos tradicionales. En Microsoft Windows y otros SO se usan las palabras *documento* y *carpeta*
- Cada vez se usan más los discos de red y los sistemas *en la nube*: son discos como los demás, pero que
  - No están físicamente cerca de la CPU, sino que se accede a ellos a través de una red informática, típicamente Internet
  - Con frecuencia los administra una entidad distinta

Dispositivos de Entrada Salida

Aquel tipo de dispositivo periférico de un computador capaz de interactuar con los elementos externos a ese sistema de forma bidireccional, es decir, que permite tanto que sean ingresada información desde un sistema externo, como ser emitir información a partir de ese sistema [2] (Wikipedia)

- Con frecuencia se usan las siglas E/S (entrada salida) o I/O (input output)
- Ejemplos: Consola (pantalla + teclado), ratón, impresora, pantalla táctil, escáner, lector braille, tarjeta de red, módem...

[2] La memoria secundaria realmente también es un dispositivo E/S, pero como es tan importante, en nuestra clasificación le dedicamos una categoría completa

Unidades de almacenamiento

En un ordenador, todos los datos (incluidos los programas) se almacenan como números en sistema binario

- Los números binarios no usan los dígitos del 0 al 9, sino el 0 y el 1
- Esto es muy conveniente: por ejemplo el 1 se puede representar por 5 voltios y el 0 por 0 voltios.
  - O mejor aún: un voltaje por debajo de 2.5 V es un 0, un voltaje superior a 2.5 V es un 1
- Otro ejemplo: un agujero microscópico en un cdrom (*pit*) representa un 1. La ausencia de agujero (*land*) representa un 0

Unidades de información

Unidades de información *tradicionales*

```
1 bit
8 bits = 1 byte
1024 bytes = 1 kilobyte (Kb)
1024 kilobytes = 1 megabyte (MB)
1024 megabytes = 1 gigabyte (GB)
1024 gigabytes = 1 terabyte (TB)
1024 terabytes = 1 petabyte (PB)

... Exabyte, Zettabyte, Yottabyte
```

Este sistema tiene un problema:
Usa potencias de 2 ($2^{10} = 1024$)
A pesar de que esos prefijos están reservados para las potencias de 10 ($10^3 = 1000$)

Anexo: Unidades ISO/IEC 80000

Desde el año 1998 diversos organismos internacionales establecen que

- Los nombres anteriores (Kb, Mb, etc) deben basarse en potencias de 10
- Los nombres correctos basados en potencias de 2 son kibibyte (KiB), mebibyte (MiB), gibibyte (GiB), tebibyte (TiB), pebibyte (PiB), exbibyte (EiB), zebibyte (Zib) , yobibyte (YiB)

Sin embargo, esta norma no se emplea mucho. Sigue siendo más frecuente la notación tradicional.

- Cuando veamos que por ejemplo un disco tiene 140 Mb, no podemos estar seguros de si son 140 * 1024 o 140 * 1000

Pasos en la programación:

1. Definir el problema. También llamado especificar. Se pueden usar
   - Métodos formales. Estuvieron de moda un tiempo pero raramente se usan
   - Descripciones informales, detalladas, en lenguaje natural (español, inglés...)
2. Diseñar un algoritmo
   Es un plan detallado de cómo será el programa. Normalmente se usa un lenguaje denominado *pseudocódigo*, a mitad de camino entre lenguaje natural y un lenguaje de programación
3. Implementar
   Es la programación en sentido estricto. Se codifican las instrucciones en un *lenguaje de programación*, en nuestro caso, Pascal
4. Probar
5. Corregir

- En las metodologías de programación tradicionales (años 1970, 1980), estos 5 pasos (definir, diseñar, implementar, probar, corregir) se consideraban bien definidos y aislados. Se suponía que se debía ejecutar cada paso una vez, en cascada uno detrás de otro
- Desde los años 1990, lo habitual es usar diversas metodologías *ágiles*, donde estos 5 pasos se repiten una y otra vez, generando prototipos cada vez más completos

Compilación

- Para escribir un programa (en Pascal o en cualquier otro lenguaje de programación) usamos un editor de texto (no un procesador de texto)
  Ejemplo: Atom, Geany
- El programa escrito no se puede ejecutar directamente, es necesario compilarlo y enlazarlo
  En nuestro caso usaremos *free pascal compiler*, tecleando desde un terminal
  `fpc -gl nombre_del_programa.pas`
- Con esto obtenemos un *ejecutable*, listo para ser ejecutado por el sistema operativo.

Ejemplo: holamundo.pas

```
program holamundo;
begin
    writeln('Hola, mundo')
end.
```

Compilación:

```
Free Pascal Compiler version 3.3.1 [2018/09/14] for x86_64
Copyright (c) 1993-2018 by Florian Klaempfl and others
Target OS: Linux for x86-64
Compiling holamundo.pas
Linking holamundo
7 lines compiled, 0.3 sec
```

Ejecución:

```
koji@mazinger:~/pascal$ ./holamundo
Hola, mundo
```

Tipos de error en un programa

- Error de compilación
  Son los más sencillos de detectar, nos los indicará el compilador antes de ejecutar nada. P.e. errores de sintaxis, errores con los tipos de datos...
- Errores en tiempo de ejecución
  Un poco más complicados de detectar, solo se producen si el programa ejecuta cierta instrucción con ciertas condiciones. P.e. una división entre cero, un fichero que no existe...
- Error lógicos.
  También llamados *bugs* (bichos). Son los más difíciles de detectar y corregir. El programa hace algo que no genera errores, pero que no es lo que deseamos que haga
- Defectos en la claridad del código
  El programa puede que no produzca errores actualmente, pero su falta de calidad conlleva errores potenciales

Si hay un error de compilación, el compilador nos indica la fila y la columna

```
program holamundo_erroneo;
begin
    writeln['Hola, mundo']     //  ¡Mal!
        { Esto es un error, el argumento está entre corchetes
          cuando debería estar entre paréntesis}
end.
```

Compilación

```
koji@mazinger:~/pascal$ fpc -gl holamundo_mal.p
Free Pascal Compiler version 3.3.1 [2018/09/14] for x86_64
Copyright (c) 1993-2018 by Florian Klaempfl and others
Target OS: Linux for x86-64
Compiling holamundo_mal.p
holamundo_mal.p(3,13) Error: Illegal qualifier
holamundo_mal.p(8) Fatal: There were 1 errors compiling module, stopping
Fatal: Compilation aborted
Error: /usr/bin/ppcx64 returned an error exitcode
```

En la línea 3, columna 13, tenemos un error de tipo *Illegal qualifier*

Defectos en la claridad del código

Es imprescindible que el código sea claro, con un diseño adecuado y que cumpla los convenios establecidos

- Con frecuencia el principiante, que acaba de hacer un programa de unas docenas de líneas, piensa *lo fundamental es que funcione. Que esté bonito es secundario y subjetivo*
- Esto es completamente erróneo.
  - En la vida profesional no hay programas de unas docenas de líneas. Los programas tienen entre miles y millones de líneas
  - Los programas de tamaño real, mal escritos, que funcionan correctamente, no existen

Diseño de algoritmos

Para diseñar un algoritmo deberíamos emplear solo tres tipos de construcción

- Secuencia de acciones
- Selección de acciones
- Iteración de acciones

Construcciones adicionales no son ni necesarias ni recomendables

Lenguaje Pascal

- Creado por Niklaus Wirth a finales de los años 1960
- Lenguaje sencillo pero completo, muy adecuado para formar a un programador y que adquiera buenas prácticas
- En los años 1980 y 1990 fue muy popular en la industria, hoy es raro usarlo fuera de la ensañanza, aunque sigue siendo perfectamente válido para desarrollar (en Windows, macOS, Linux, iOS, Android...)
- Con los años han aparecido diferentes *dialectos* con pequeñas variantes, aquí usaremos Free Pascal

Estructura de un programa en Pascal

Como referencia, indicamos aquí la estructura de un programa en Pascal:
Un programa se compone de

- Cabecera (`program nombre`)
- Un bloque
  - Parte declarativa: declaración de constantes, variables, funciones y procedimientos
  - Parte de las sentencias: `begin` lista_de_sentencias `end`
- Un punto

lista_de_sentencias

- Secuencia de sentencias, separadas por *punto y coma*

Las sentencias pueden ser

- Simples
  - Asignación
  - Llamada a procedimiento
  - Raise
- Estructuradas
  - Condicional (case, if)
  - Bucle: for, repeat, while
  - with
  - try

Como ejemplo y como anticipo de lo que trataremos en los próximos 4 temas, mostramos el siguiente programa

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program nota_fpi;

    {Constantes visible en todo el programa (y subprogramas)}
const
    Compensable: real = 4.0;

    { Peso relativo de cada apartado de la evaluación, en tanto por uno}
    Peso_entrega: real = 0.25;    { Entrega prácticas }
    Peso_teorico: real = 0.35;    { Examen teórico }
    Peso_practico: real = 0.40;   { Examen práctico }

function hace_media(entrega, teorico, practico: real): boolean;
begin
    result :=  (entrega >= Compensable) and (teorico >= Compensable)
            and (practico >= Compensable)
end;

function media(entrega, teorico, practico: real): real;
begin
    result := entrega * Peso_entrega +
              teorico * Peso_teorico +
              practico * Peso_practico
end;
```

```
    {Constantes del programa principal}
const
    Entrega_jperez : real = 3.5;
    Teorico_jperez: real = 6.2;
    Practico_jperez: real = 7.4;

begin
    write('Nota final: ');
    if (hace_media( Entrega_jperez, Teorico_jperez, Practico_jperez)) then
        writeln( media(Entrega_jperez, Teorico_jperez, Practico_jperez):0:2)
    else
        writeln( 'No apto' )
end.
```

Tabulación

El texto de los programas no está alineado a la izquierda, como el lenguaje natural, sino que se añaden espacios. Esto se denomina sangrado o tabulación

- Una tabulación correcta es imprescindible para la claridad de un programa
- No hay una forma única de tabular, hay distintos criterios dependiendo del gusto del programador
  - Si el proyecto lo empezamos nosotros, podemos tabular según nuestras preferencias. Pero siempre de forma consistente
  - Para un proyecto preexistente, debemos respetar el criterio establecido

Hay dos formas de insertar espacios

- Mediante el carácter *tab*
  Es un carácter especial que significa *añadir cierto espacio horizontal*. No especifica cuánto. Depende del editor, puede estar configurado por omisión para ocupar el equivalente a 4 espacios, a 8 espacios o cualquier otro valor, es configurable
- Mediante espacios

Ejemplo:

```
const
    a:real=3;
```

A la izquierda de la *a* puedo haber insertado o bien 1 tabulador o bien 4 espacios. En una versión impresa como esta transparencia, es imposible distinguirlo
Supongamos que teníamos el editor configurado con el tabulador a 4 espacios y que hayamos insertado un tabulador. Si en otro momento editamos con un editor configurado a 8 espacios, veremos algo así

```
const
        a:real=3;
```

Esto no es un problema

Según nuestras preferencias, podemos tabular

- Con tabuladores
- Con 4 espacios
- Con 8 espacios
- Con otro número de espacios (aunque es poco frecuente)

Pero es imprescindible mantener el mismo criterio dentro del mismo programa o proyecto
Si en algún momento necesito cambiar el criterio en todo el proyecto, es fácil hacerlo automáticamente

- En esta asignatura, consideramos que cada fichero es un proyecto independiente, por tanto puedes cambiar de criterio entre programa y programa (por si quieres experimentar o cambia tu gusto)
- En entornos *reales*, esto no sería aceptable

El problema de no respetar el mismo criterio es el siguiente.
Supongamos que tengo el editor con un tabulador a 4 espacios.

```
const
    a:real=3;
    b:integer=0;
```

Supongamos que para la primera declaración usé tabuladores y para la segunda,espacios
Si en otro momento otra persona o yo mismo trabajo con un editor con el tabulador a 8 espacios, veré lo siguiente

```
const
        a:real=3;
    b:integer=0;
```

¡Esto arruina por completo la claridad del programa!

Por tanto

- Es imprescindible configurar el editor para que muestre los caracteres invisibles. Así, representará tabuladores y espacios con algún símbolo especial (normalmente flechas o puntos)
- Es imprescindible prestar atención para mantener el criterio de tabulación establecido

En esta asignatura,

- Un programa con tabulación inconsistente tendrá mala nota o suspenderá, dependiendo de la gravedad del defecto
- Un programa que mezcle continuamente tabulaciones con espacios, estará suspenso (porque potencialmente, toda la tabulación será incorrecta)

Entorno de prácticas

En la asignatura usaremos el compilador Free Pascal

- Es libre y gratuito, está disponible para Microsoft Windows, macOS y Linux
- En casa puedes trabajar en cualquiera de estos tres sistemas operativos, pero en clase (y en el examen) es necesario que sepas usarlo en Linux, desde el terminal de texto

También necesitarás un editor de texto.

- Puedes usar el que prefieras, aquí recomendamos atom (https://atom.io) o geany (https://www.geany.og) Ambos son libres y gratuitos y están disponibles para Microsoft Windows, macOS y Linux

Para compilar los programas y para ejecutarlos, usaremos un terminal
En Ubuntu Linux 18.04 hay tres formas de lanzar un terminal, todas resultan equivalentes

- Pulsar las teclas
  `Ctrl Alt t`
- Hacer clic sobre el icono del terminal en el *dock* (la barra de iconos en la parte izquierda del escritorio)
- Pulsar la tecla *super* (la tecla *windows*) y escribir *terminal*

Escritorios virtuales

Los escritorios virtuales son una herramienta que permite trabajar de forma muy cómoda, simula que tenemos ventanas sobre varios escritorios al mismo tiempo (varias pantallas), y cambiar rápidamente de uno a otro
Es como tener varias pantalla virtuales

- En Ubuntu Linux se les llama *workspaces*
  Atajos de teclado:
  - Opción 1
    ctrl alt flecha_arriba, ctrl alt flecha_abajo
  - Opción 2
    super RePag, super AvPag[3]
- En macOS este sistema se denomina *Mission control*
- En Windows 10 es el *Task View* quien se encarga de gestionar distintos *Virtual Desktops*

[3] Recuerda que super es la tecla windows

Sugerencia: trabaja siempre con varios escritorios virtuales

- Uno con el editor de texto con el código fuente
- Otro con un terminal para compilar y ejecutar
- Opcionalmente, otro con el gestor de ficheros (en Ubuntu se llama Nautilus)
- Opcionalmente, otro con el navegador web

Piensa en la disposición que te resulte más cómoda y procura usar siempre la misma

Uso básico del terminal

Para moverte por el sistema de ficheros necesitarás manejar estas órdenas básicas de la shell

- `ls`
  Listado de ficheros y directorios
- `ls -l`
  Listado detallado de ficheros y directorios. Reconocerás los ficheros porque el primer carácter de la línea es una letra *d*
- `cd nombre_directorio`
  Entras en un subdirectorio del directorio actual
- `cd ..`
  Subir al directorio padre del directorio actual
- `./nombre_programa`
  Ejecutar un programa del directorio actual. Por ejemplo `./holamundo`

Compilar desde el terminal

Una vez situados en el directorio donde esté el código fuente del programa en Pascal que queramos compilar, escribimos
`fpc -gl nombre_fichero.pas`

- Las opciones `-gl` no son imprescindibles, pero sí son convenientes para que el compilador muestre los mensajes de error con más claridad
- En este curso, los ficheros con el código fuente tendrán la extensión `.pas` (que es la más habitual). Otras extensiones posibles son `.pp` y son `.p`
- El compilador generará un fichero `.o` con el código objeto. Podemos ignorarlo y/o borrarlo
- El fichero ejecutable tendrá el mismo nombre que el fichero con el código fuente, pero sin extensión

Instalación en tu ordenador

- Windows

  Bájate y ejecuta el instalador desde la página de free pascal

  https://www.freepascal.org/download.var

- Mac OS

  1. Instala *Xcode* desde la app store
  2. Instala *Xcode Command Line Tools*, ejecutando en un terminal

     ```
     xcode-select --install
     ```

  3. Bájate y ejecuta el instalador desde la página de free pascal

     https://www.freepascal.org/download.var

- Ubuntu Linux, Linux Mint

  Ejecuta en un terminal

  ```
  sudo apt-get update
  sudo apt-get upgrade
  sudo apt-get install fpc
  ```

Compilación en Windows

Es imprescindible que sepas trabajar en Linux, porque es lo que usamos en el laboratorio (y por tanto en el examen). El funcionamiento en macOS es igual
Si quieres programar en Windows, los cambios son pocos

- Linux
  Lanza un *terminal* y ejecuta allí las órdenes
  Windows
  Lanza *cmd* y ejecuta allí las órdenes
- Linux
  Usa `cd`, `cd ..` y `ls -l`
  Windows
  Usa `cd`, `cd ..` y `dir`

- Linux
  El fichero ejecutable no lleva extensión
  Windows
  El fichero ejecutable lleva extensión `.exe`
- Linux
  La separación entre directorios se indica con una barra. P.e.
  `~/fpi/practica01/holamundo.pas`
  Windows
  La separación entre directorios se indica con una barra invertida. P.e.
  `escritorio\fpi\practica01\holamundo.pas`

Configuración de Atom

Para que atom de color al fuente siguiendo la sintaxis de Pascal, ejecuta dede un terminal

- `apm install language-pascal`

Para que atom muestre los caracteres invisibles, de forma que puedas distinguir los tabuladores de los espacios

- Pulsa la opción *edit* del menú de atom, luego vete a *preferences*, luego a *editor*, busca la opción *show invisibles* y actívala
- En lo sucesivo, instrucciones como esta las indicaremos con una sintaxis mucho más compacta:
  Activa
  `edit|preferences|editor|show invisibles`

Configuración de Geany

Para que Geany muestre los caracteres invisibles:

- `ver|mostrar espacios en blanco`