Introducción a la programación en el ámbito de diversas ingenierías

Ester Bernadó, Josep Maria Garrell, Manuel Román, Maria Salamó, Joan Camps, Jaume Abella

Departament d'Informàtica
Enginyeria i Arquitectura La Salle
Universitat Ramon Llull
Passeig Bonanova, 8. 08022 Barcelona

{esterb,josepmg,mroman,mariasal,joanc,jaumea}@salleURL.edu

Resumen

La asignatura de programación constituye una base fundamental para las diversas carreras de ingeniería. Los posibles enfoques que se pueden utilizar, tanto de contenidos como de método docente, son muy diversos. En este artículo presentamos la solución adoptada en Enginyeria i Arquitectura la Salle (Universitat Ramon Llull).

En este contexto la asignatura es común a los diversos planes de estudios que se imparten: Ingeniería de Informática, Ingeniería de Telecomunicaciones y Graduado en Multimedia. Por este motivo la asignatura es producto de un compromiso entre las diversas necesidades de cada carrera. La solución que aquí planteamos se ha venido utilizando en los últimos años con resultados plenamente satisfactorios.

1 Introducción

La asignatura de Programación (Introducción a la Programación) del centro *Enginyeria i Arquitectura La Salle*, de la Universitat Ramon Llull, es una asignatura troncal que se imparte en el primer curso de Ingeniería Técnica de Informática, Ingeniería Técnica de Telecomunicaciones y Multimedia.

El hecho que la asignatura sea común a diversas carreras, conlleva la necesidad de hacer coincidir objetivos dispares. Por este motivo el temario y el método docente son el producto de un compromiso, y se organizan para cubrir las diversas necesidades de los distintos planes de estudio.

Por ejemplo, en la carrera de Informática de Sistemas, la programación se complementa con otras asignaturas de cursos posteriores [1]:

- Estructuras de datos. Se profundiza en las estructuras de datos lineales, funcionales, árboles y grafos.
- Programación II. La asignatura consta de tres partes: el desarrollo y la verificación formal de programas, esquemas de algoritmos y una introducción a la programación paralela.

En cambio, en Ingeniería Técnica de Telecomunicaciones, ésta es la única asignatura donde se imparten fundamentos teóricos de programación. Otras asignaturas de cursos posteriores, como por ejemplo, *Diseño y Programación de microprocesadores* (de tercer curso) usan conceptos de programación a nivel aplicativo. Es decir, se realizan prácticas donde se precisa la implementación en un lenguaje de programación, partiendo de la base proporcionada en *Programación*.

Este hecho da lugar a un compromiso entre la base teórica o introductoria de la Ingeniería Técnica de Informática, y la parte de carácter más aplicativo y autocontenido de la Ingeniería Técnica de Telecomunicaciones. El temario que se expone en la sección 4 es la solución adoptada ante dicho compromiso.

El calendario académico de *Enginyeria i Arquitectura La Salle* determina el carácter anual de las asignaturas. Por lo tanto, la asignatura tiene una duración anual, con nueve créditos repartidos entre teoría y prácticas.

El presente trabajo se estructura de la siguiente manera. En la sección 2, se presentan los objetivos de la asignatura. En la sección 3, método docente, se expone la problemática docente que surge en esta introducción a la programación y la justificación del temario, que se halla detallado en la sección 4. Seguidamente, se describen las prácticas de la asignatura y la bibliografía y los materiales docentes que se recomiendan al alumno. Finalmente, las conclusiones de nuestra experiencia y las referencias bibliográficas.

2 Objetivos

El objetivo global de la asignatura de Programación es introducir al alumno en el diseño y programación de algoritmos. En este proceso, se da especial énfasis en el planteamiento global del problema, la fase de diseño y la implementación.

Dada la gran diversidad de lenguajes de programación, metodologías de diseño y técnicas de programación, se ha creído conveniente marcar unos objetivos de contenido, que permitan al alumno tener una visión global de la programación, desligada de herramientas específicas.

Además de los objetivos referentes al contenido de la asignatura, se plantean también una serie de objetivos didácticos, que permitan al alumno adquirir una metodología de trabajo.

2.1 Objetivos de Contenido

En este subapartado se pretenden introducir todos aquellos conceptos de contenido que se consideran fundamentales dentro de la asignatura de programación. Los objetivos que se exponen hacen referencia al paradigma de programación escogido, el lenguaje utilizado en clase para introducir los diferentes conceptos, la introducción a los tipos abstractos de datos y a la orientación a objetos.

De los diferentes paradigmas de lenguajes de programación (imperativos, funcionales, lógicos,etc.), la asignatura se centra en dos de ellos: el imperativo y el de orientación a objetos. De hecho, se enfatiza el modelo imperativo, ya que didácticamente se considera más sencillo como punto de partida para alumnos que se inician en la programación. El lenguaje imperativo utilizado es el lenguaje C.

Para la introducción de los conceptos básicos de programación, se utiliza pseudocódigo. De esta forma, se pretende independizar el aprendizaje de los conceptos o esquemas generales de la programación, de los detalles de un lenguaje de programación.

Una vez se han explicado los conceptos básicos de la programación imperativa, y teniendo en cuenta que el alumno tendrá que enfrentarse a aplicaciones de una complejidad considerable, es necesario introducir el diseño modular como una base imprescindible para tratarla.

De forma complementaria al diseño modular, y para facilitar al alumno la posterior comprensión de la programación orientada a objetos, se introduce el concepto de Tipo Abstracto de Datos (TAD). El hecho de utilizar TADs facilita la posterior introducción de estructuras de datos como entidades encapsuladas y reutilizables. Es muy importante hacer comprender al alumno la estrecha relación que existe entre el diseño modular y los TADs.

La orientación a objetos se ha convertido durante los últimos años en una herramienta muy útil para tratar la complejidad de las aplicaciones. Por este motivo se considera importante introducir este paradigma a los alumnos, de manera que conozcan conceptos tales como encapsulación, abstracción, herencia, polimorfismo, etc.

2.2 Otros objetivos

Otros objetivos, de carácter más metodológico, hacen referencia a la futura incorporación de los alumnos en el mundo laboral. Desde la asignatura, se intentan potenciar ciertas características que consideramos importantes en el perfil de un futuro analista/programador: el trabajo en equipo y la iniciativa o capacidad de autoaprendizaje. Ambas características se potencian principalmente en las sesiones de laboratorio.

3 Método docente

En esta sección se desarrolla la metodología docente empleada en la asignatura. Se presenta la justificación del temario, dentro del marco de los objetivos fijados. Así mismo, planteamos algunos puntos problemáticos o susceptibles de discusión de dicho temario, observados a partir de nuestra experiencia docente.

3.1 ¿Por qué pseudocódigo?

Uno de los objetivos de la asignatura es lograr que el alumno asimile las técnicas y métodos básicos de la programación imperativa, sin estar ligado a ningún lenguaje de programación concreto. La idea es aprender a diseñar algoritmos usando los esquemas básicos de la programación imperativa, independientemente de los detalles específicos de un lenguaje de programación [4], [3].

Por este motivo, como ya se ha comentado anteriormente, la introducción a los esquemas de la programación se realiza en pseudocódigo. Cabe destacar las siguientes ventajas:

- En el planteamiento de los algoritmos, no se presta tanta atención a los detalles de sintaxis y/o lexicográficos, como conllevaría un lenguaje de programación concreto.
- Las estructuras básicas del pseudocódigo son válidas para cualquier lenguaje imperativo. Por lo tanto, la traducción de pseudocódigo a un lenguaje de programación no supone una gran dificultad.
- El pseudocódigo permite expresar los algoritmos a distintos niveles de detalle.

3.2 ¿Qué lenguaje de programación?

El lenguaje de programación que usamos es el C, por su gran versatilidad en muchos campos distintos de la ingeniería y por su creciente aplicabilidad en la industria.

Anteriormente, se utilizaba el lenguaje Pascal en la primera parte del curso y, en el último trimestre, se introducía el lenguaje C. Pascal es un lenguaje más pedagógico y de hecho, el pseudocódigo que utilizamos es más similar a éste. Últimamente este enfoque se ha canviado por los siguientes motivos:

- En primer lugar, el alumno no tenía tiempo suficiente de asimilar el lenguaje C. Aunque ésto no supone un problema para Ingeniería de Informática, puesto que la asignatura viene seguida por otras asignaturas de segundo curso, para Ingeniería de Telecomuniciones no hay otras asignaturas en el plan de estudios que permitan complementarla. Por lo tanto, es conveniente dar una base sólida de un lenguaje como C.
- En segundo lugar, y de nuevo debido a que la asignatura debe cubrir distintos planes de estudios, se introdujo el paradigma de orientación a objetos. Y para dar una versión aplicativa de estas técnicas, también se introduce un lenguaje orientado a objetos. Si en la primera parte del curso se estudia C, continuar con el lenguaje C++ como ejemplo de un lenguaje Orientado a Objetos, es más asequible para el alumno que empezar por un lenguaje totalmente nuevo. De hecho, puede considerarse el lenguaje C como un subconjunto del lenguaje C++ [10].

No obstante, el aprendizaje de C como primer lenguaje de programación presenta más dificultad que por ejemplo Pascal. El lenguaje C no es tan pedagógico. Por citar un ejemplo concreto, el paso de parámetros por referencia se realiza mediante punteros [5]. En consecuencia, para poder implementar el concepto de paso por referencia, de fácil comprensión en pseudocódigo, se necesita conocer el tratamiento de punteros en C, y éste punto del temario no se imparte hasta el segundo trimestre. Este problema surge, sobretodo, en la realización de prácticas durante el primer trimestre. Así pues, la coordinación de las prácticas con el temario es un punto crítico que se tratará más adelante.

Otros problemas que surgen con el aprendizaje por parte del alumno son:

- Al principio, confunden el pseudocódigo con el lenguaje C. Como es inevitable, hay diferencias en la sintaxis y ciertos aspectos concretos, por ejemplo en el tratamiento de ficheros, memoria dinámica, etcétera. El alumno tiene cierta dificultad en distinguir los conceptos y la forma concreta en que un lenguaje de programación determinado los implementa.
- Una vez que el alumno asimila un lenguaje de programación, descarta el pseudocódigo, tendiendo a implementar directamente en lenguaje C.

Estos aspectos tratan de resolverse usando siempre el pseudocódigo para plantear los algoritmos y dejando el lenguaje de programación para la implementación de las prácticas.

3.3 ¿Cómo enfocar el paradigma de la Orientación a Objetos?

Nuestro temario empieza con los fundamentos más básicos de la programación imperativa en pseudocódigo, usando algoritmos muy sencillos que progresivamente van aumentando el grado de dificultad. Al principio, los programas se estructuran en procedimientos (y funciones). Después, se introduce el concepto de programación modular y el concepto de TAD. Finalmente, se amplía el concepto de TAD con el de Objeto, mediante el paradigma orientado a objetos. Se intenta que la Orientación a Objetos no signifique un "empezar de nuevo", sino que se pueda aprovechar muchos conceptos usados en la estructuración en TADs. De esta forma, el alumno no nota un cambio total en el enfoque de la programación.

Un aspecto a discutir es sobre el aprendizaje de la Orientación a Objetos desde el principio de la asignatura. A continuación, presentamos las dos posibilidades:

- Nuestro enfoque es el más tradicional: empezando por los conceptos de programación estructurada, se va preparando el camino para la Orientación a Objetos. Este temario se ha venido aplicando en los últimos años y se ha comprobado que el alumno asimila el paradigma de orientación a objetos con facilidad.
- En cambio, otras metodologías defienden empezar directamente por el paradigma Orientado a Objetos [6] [2]. Los principales motivos consisten en enseñar desde el principio un paradigma y no hacer una transición de uno a otro. Otros defienden que el diseño se asimila mucho mejor si es orientado a objetos desde el principio. Desde este punto de vista, quizás sería más productivo empezar por una Orientación a Objetos desde el principio.

En nuestro contexto no se ha descartado la segunda alternativa, aunque nos gustaría dejar el debato abierto a discusión.

3.4 Estructuración de las sesiones

Las sesiones se estructuran en: teoría y prácticas. En las primeras, se imparte el temario y se realizan ejercicios complementarios. Las prácticas se realizan en el laboratorio y se basan en el diseño e implementación de aplicaciones, con el objetivo de ayudar al alumno en la asimilación de los conceptos vistos en clase. Asimismo, se aprovechan dichas sesiones para acostumbrar al alumno en la metodología de trabajo descrita en los objetivos.

4 Temario

1. **Introducción**

   En primer lugar, se introduce al alumno en el mundo de los ordenadores, describiendo que es un ordenador, cómo funciona y las distintas configuraciones posibles. Además, se introducen algunos conceptos básicos de álgebra de Boole, necesarios para las condiciones lógicas de los algoritmos, aunque este tema ya se desarrolla posteriormente y en profundidad en otras asignaturas.

2. **Conceptos fundamentales y pseudocódigo**

   Después de la parte introductoria, el alumno se inicia en los conceptos fundamentales de programación a través de una notación en pseudocódigo:

   2.1 Concepto de programa y propiedades: finitud y eficiencia.

   2.2 Expresiones. Operadores, variables, constantes y funciones.

   2.3 Sentencias: asignación, condicional, iterativa, alternativa, entrada/salida.

   2.4 Tipos de datos elementales: entero, real, caracter, booleano y enumerado.

   2.5 Tipos de datos estructurados: arrays, cadenas de caracteres y registros.

   2.6 Definición de tipos.

   2.7 Procedimientos y funciones.

   2.8 Variables globales y locales.

   2.9 Paso de parámetros por valor y por referencia.

   2.10 Modularidad y diseño del software.

   A continuación, y con el objetivo de facilitar la asimilación del pseudocódigo, se realizan algunos algoritmos clásicos, como por ejemplo los algoritmos de ordenación de arrays.

3. **Tratamiento de ficheros**

   En este tema se introduce el almacenamiento permanente de la información mediante el uso de ficheros. Se da una visión global de los tipos de ficheros y las operaciones de tratamiento de éstos.

4. **Memoria dinámica**

   En este punto se presentan las características de la memoria estática y dinámica, justificándose el por qué de la memoria dinámica. El tema incluye el concepto de memoria dinámica, la utilización de punteros y ejercicios aplicativos. Es uno de los temas de más dificultad de asimilación por parte del alumno.

   Estos conceptos son importantes para la comprensión de los siguientes temas. Por ejemplo, algunas estructuras de datos del próximo tema se implementan mediante el uso de memoria dinámica. Asimismo, el concepto de puntero es de vital importancia para la comprensión de algunos aspectos del lenguaje C.

5. **Concepto de TAD y estructuras de datos**

   Hasta el presente punto, el alumno ha tratado ya con aplicaciones de una cierta complejidad, pero mediante el diseño y programación modular. Se da un pequeño giro a este enfoque y se introduce el concepto de Tipo Abstracto de Datos, ilustrado con el concepto gráfico de *una caja negra*. La explicación se centra en dos puntos de vista: por una parte, la implementación interna (representación+codificación) del TAD y por otra, la utilización del mismo.

   Este concepto se aplica progresivamente en ejemplos sencillos y en las nuevas estructuras de datos que se introducen en este tema: las pilas, colas y listas. Estas estructuras son muy adecuadas para ilustrar o complementar el concepto de TAD. Por una parte, desde el punto de vista externo, es fácil identificar un conjunto de operaciones, las cuales describen su comportamiento. Por otra parte, internamente, pueden implementarse de múltiples formas. Típicamente, usamos representaciones con memoria estática y dinámica.

6. **Introducción al paradigma Orientado a Objetos y C++**

   Este tema consta de dos partes diferenciadas. La primera, es una introducción al paradigma de orientación a objectos. Se presenta el nuevo enfoque de la programación, basado en objetos, y sus características principales: la encapsulación y abstracción (ya vistos anteriormente) y la herencia y polimorfismo. También se da énfasis en el diseño de las aplicaciones utilizando este modelo de programación y sus ventajas respecto a la programación modular.

   Finalizada la introducción a este paradigma, se explica el lenguaje C++, como ejemplo de un lenguaje orientado a objetos. El objetivo no es tan solo aprender a programar en C++, sino entender cómo aplicar el paradigma orientado a objetos, mediante un lenguaje.

   Cabe destacar que no se puede profundizar en todos los detalles de C++, debido a las limitaciones de tiempo.

7. **Lenguaje C**

   Este tema no se imparte a final de curso, como podría parecer de la lectura de este temario, sino que se trata simultáneamente a otros temas. Los puntos más destacados del tema son los siguientes:

   7.1 Introducción

   7.2 Tipos de datos elementales y estructurados.

   7.3 Operadores: aritméticos, lógicos,...

   7.4 Sentencias.

   7.5 Estructura de un programa en C.

   7.6 Declaración de funciones y prototipos.

   7.7 Paso de parámetros.

   7.8 Entrada/Salida.

   7.9 Arrays y punteros.

   7.10 Tratamiento de strings.

   7.11 Memoria dinámica.

   7.12 Argumentos de la línia de comandos.

   7.13 Ficheros.

5 Prácticas de laboratorio

El objetivo de las prácticas de laboratorio es que el alumno desarrolle un conjunto de aplicaciones para complementar los conocimientos de las sesiones teóricas. Simultáneamente, el alumno apendre a familiarizarse con los recursos y herramientas básicas, como por ejemplo el sistema operativo y el compilador.

5.1 Entorno de trabajo

Las sesiones de laboratorio están formadas por grupos de 2 personas, para potenciar el trabajo en equipo y están asistidos por un monitor de prácticas. Los alumnos realizan las prácticas con terminales de texto, conectados a servidores *Unix*.

5.2 Contenido de las prácticas

Las sesiones prácticas se estructuran de la siguiente forma:

1. **Introducción al sistema operativo Unix**

   Las primeras clases de laboratorio son de introducción al sistema operativo Unix y herramientas básicas, tales como un editor de texto. Estas primeras clases son sesiones lectivas, acompañadas de ejercicios prácticos y se realizan en el laboratorio.

2. **Introducción al lenguaje C y al compilador**

   Los primeros contactos con el lenguaje C se realizan mediante programas muy sencillos. Se quiere familiarizar al alumno en los tipos de datos básicos, las sentencias de control y la entrada/salida.

   Estos pequeños ejercicios sirven también como introducción al compilador (gcc): cómo compilar, cómo ejecutar un programa, cómo interpretar los mensajes del compilador y cómo corregir los errores tanto de compilación, como de ejecución.

3. **Práctica sobre programación estructurada, en lenguaje C**

   Cuando el alumno ya está familiarizado con el entorno se va desarrollando un conjunto de prácticas en lenguaje C. Estas prácticas, se realizan gradualmente y de forma coordinada con el temario de las clases lectivas. Se estructuran de la siguiente manera:

   (a) Tipos de datos simples y estructurados: arrays y registros. Entrada y salida.

   (b) Aplicación de un algoritmo de ordenación en un array de datos.

   (c) Tratamiento de ficheros en C.

   (d) Implementación de una estructura de datos, usando el concepto de TAD. En concreto, se suele implementar una lista ordenada y se requiere usar memoria dinámica.

   A medida que la complejidad de la práctica aumenta, el alumno aprende a estructurar su código fuente en ficheros y a compilarlos de forma eficiente con ayuda de un fichero *makefile*.

4. **Introducción a la Orientación a Objetos y lenguaje C++**

   Esta última práctica se realiza a final de curso y por limitación de tiempo, no se pueden aplicar todos los conceptos tratados en clase sobre la Orientación a Objetos.

   Por tanto, la práctica no es más que la implementación de una pequeña aplicación usando los conceptos de abstracción y encapsulación, y algunas características particulares del lenguaje C++.

6 Bibliografia y otros materiales docentes

La bibliografia básica recomendada es:

- **Algoritmos fundamentales y pseudocódigo**

  Francesc Escudero y Josep M. Garrell, "Fonaments de Programació", Editorial Bruño/EUETT, 1993

  Este libro sirve de guía básica al alumno sobre los conceptos explicados en clase, excepto el tema de orientación a objetos. El libro contiene el mismo enfoque que las clases teóricas impartidas. De hecho, es un

libro "hecho a medida"para la asignatura de Programación.

- **Consulta sobre lenguaje C**

  Brian W.Kernighan and Dennis M.Ritchie, "The C Programming Language", Prentice Hall, 1988

- **Lenguaje C++**

  Bjarne Stroustrup, "The C++ Programming Language", Addison-Wesley, 1991

  Este libro es muy especializado y contiene algunos aspectos de C++ demasiado complejos respecto al nivel que se imparte en nuestra asignatura. No obstante, puede ser útil para realizar consultas puntuales.

- **Publicaciones internas**

  Además de estos libros, básicamente utilizados a nivel de consulta o refuerzo de conceptos, el alumno dispone de publicaciones internas de ejercicios [7] [8] [9].

  Los ejercicios están separados por temas y se ordenan de forma ascendente en dificultad. Cada publicación tiene un conjunto de ejercicios resueltos. También, se añaden algunos exámenes resueltos de años anteriores, los cuales son muy útiles al alumno para la preparación de la asignatura.

7 Conclusiones y temas de debate

Este trabajo concluye con un breve resumen de lo expuesto anteriormente y plantea un conjunto de temas abiertos y de posible debate.

En primer lugar se han identificado los requerimentos de contorno que, sobre la asignatura, ejercen los distitos planes de estudios en la que ésta se imparte. Se ha dado especial énfasis en buscar una solución que garantice los requisitos mínimos y que satisfaga, en la medida de lo posible, las necesidades particulares de cada plan de estudios. Se han descrito los objetivos de la asignatura y el método docente utilizado para cubrirlos, así como el temario y las prácticas. Finalmente se ha descrito la bibliografía y el material docente utilizado.

Para finalizar este trabajo nos gustaría plantear un conjunto de temas de debate que consideramos de vital importancia en el planteamiento de una asignatura que trate una introducción a la programación. Algunos de estos temas se han tratado a lo largo de este artículo. Otros, en cambio, no se han comentado anteriormente por el contexto de la asignatura, pero deben analizarse en el marco general de una introducción a la programación. Veámoslos:

- ¿Debe utilizarse un paradigma de orientación a objetos? En caso afirmativo, ¿qué peso debe tener en el temario? ¿Se debe utilizar únicamente? O, ¿se debe introducir a continuación de un paradigma imperativo tradicional?

- Sea cual sea el paradigma utilizado, ¿cuáles son los lenguajes de programación más adecuados? Quizás el más pedagógico no es el que posteriormente se utilizaría en un entorno laboral.

- ¿Cómo deben organizarse la sesiones de laboratorio? ¿Qué tipo de prácticas deben plantearse?

- ¿Debe introducirse el concepto de métodos formales? O, ¿debe esperarse a que el alumno ya tenga cierta experiencia en programación? ¿Puede analizarse alguna solución intermedia?

- Y ya finalmente, desde el punto de vista pedagógico ¿cómo se puede *recuperar* al alumno que *ha perdido el tren* de las clases?

Referencias

[1] *Guia Acadèmica i d'Activitats.* Enginyeria La Salle, Universitat Ramon Llull, 1997-98.

[2] Timothy Budd. *Introducción a la Programación Orientada a Objetos.* Addison-Wesley Iberoamericana, 1994.

[3] Jorge Castro, Felipe Cucker, Xavier Messeguer, Albert Rubio, Lluis Solano, and Borja Valles. *Curs de programació.* McGraw-Hill, 1992.

[4] Francesc Escudero Costa and Josep Maria Garrell i Guiu. *Fonaments de Programació.* Bruo/EUETT, 1993.

[5] Brian W. Kernighan and Dennis M Ritchie. *The C programming language.* Prentice Hall, 1988.

[6] Jo Ellen Perry and Harold D. Levin. *An Introduction to Object-Oriented Design in C++.* Addison-Wesley, 1996.

[7] Maria Salamó, Ester Bernadó, Joan Camps, Manuel Román, and Jaume Abella. *Exercicis d'algorismes fonamentals.* Departament d'Informàtica, Enginyeria La Salle, Universitat Ramon Llull, 1997-98.

[8] Maria Salamó, Ester Bernadó, Joan Camps, Manuel Román, and Jaume Abella. *Exercicis d'estructures de dades i TADs.* Departament d'Informàtica, Enginyeria La Salle, Universitat Ramon Llull, 1997-98.

[9] Maria Salamó, Ester Bernadó, Joan Camps, Manuel Román, and Jaume Abella. *Exercicis de llenguatge C i C++.* Departament d'Informàtica, Enginyeria La Salle, Universitat Ramon Llull, 1997-98.

[10] Bjarne Stroustrup. *The C++ Programming Language.* Addison-Wesley, 1995.