Enseñanza de la programación

Ariel Ferreira Szpiniak
Universidad Nacional de Río Cuarto, Departamento de Computación,
Río Cuarto, Argentina, 5800
aferreira@exa.unrc.edu.ar

Guillermo A. Rojo
Universidad Nacional de Río Cuarto, Departamento de Computación,
Río Cuarto, Argentina, 5800
grojo@exa.unrc.edu.ar

Resumen

Las computadoras y la forma de programarlas han evolucionado de una forma vertiginosa con el paso del tiempo. Estos avances impactaron en el ámbito educativo. Un punto de inflexión lo marca la aparición PASCAL, un lenguaje de programación para la enseñanza de técnicas de programación estructurada que se convirtió en un estándar de facto en el mundo de la programación. Han pasado más de 35 años desde ese entonces y sin embargo la programación estructurada sigue teniendo enorme importancia en el campo de la enseñanza. Aunque han habido muchos cambios y avances desde ese entonces, coexisten varios enfoques y tendencias pero sin consenso sobre cual es la mejor manera de enseñar los conceptos básicos de la programación. En este trabajo se analizan los temas que se trabajan en la asignatura de primer año de las Carreras de Computación de la Universidad Nacional de Río Cuarto, el enfoque adoptado desde el año 2004 para la enseñanza, la organización temática y su fundamentación. Por último se presentan los resultados obtenidos en la aplicación del nuevo enfoque que fueron recogidos el marco de los Proyectos de Innovación e Investigación para el Mejoramiento de la Enseñanza de Grado, durante 2004 y 2005.

Palabras claves: Programación, Algoritmos, Enseñanza, Aprendizaje, Enfoque.

Introducción

Las computadoras y la forma de programarlas han evolucionado de una forma vertiginosa con el paso del tiempo. La primera generación de computadoras surgió allá por la década del ’50 con la aparición de la UNIVAC y la ENIAC, de la mano de los tubos de vacío, donde la única manera de programarlas era mediante el lenguaje máquina, basado únicamente en números binarios. La aparición casi 10 años después de la segunda generación, gracias a la invención del transistor, también se nutrió de grandes avances en el terreno de la programación de esas computadoras, primero mediante los lenguajes ensambladores y muy poco tiempo después con los denominados lenguajes de alto nivel como FORTRAN (1964) y COBOL (1959). La tercera generación de computadoras toma relevancia cuando surgen los chips de circuitos integrados y los discos magnéticos, pero ello solo aporta mejoras en el terreno del hardware, aumentando considerablemente la velocidad, confiabilidad y capacidad de almacenamiento de las computadoras. La llegada de la cuarta generación, en la década del ’70, sería más una evolución que una revolución, al pasar del chip especializado para uso en la memoria y procesos lógicos, al procesador de propósito general en un chip o microprocesador. Sin embargo, desde el punto de vista de la programación sería muy importante puesto que en 1968 apareció una versión preliminar del lenguaje PASCAL y el primer compilador totalmente completo estuvo listo a finales de 1970. El lenguaje de programación Pascal es un lenguaje de alto nivel y de propósito general desarrollado

por Wirth [1]. Aunque el propósito de Wirth fue crear un lenguaje para la enseñanza de técnicas de programación, a medida que pasaron los años Pascal se fue convirtiendo en un estándar en el mundo de la programación. El éxito de Pascal radicó en el hecho de que fue concebido bajo una nueva metodología de programación denominada *programación estructurada*. Ello permitió expresar principios de programación y de diseño de solución de problemas en forma abstracta y estructurada [2].

Esta metodología permitió solucionar grandes problemas planteados en la década del `70 que se conocieron como *"la crisis del software"*. Este término fue acuñado cuando la industria del software ya había producido los suficientes programas para darse cuenta de que habían cosas que no estaban nada bien: llevaba mucho tiempo terminar los programas, el coste era muy elevado y tampoco se podían encontrar todos los errores antes de entregar el programa al usuario final [3]. Luego vendrían la Ingeniería de Software, nuevas técnicas y paradigmas de programación orientados a mejorar cada vez más la construcción de programas, sin embargo la metodología de la programación estructurada seguiría vigente hasta nuestros días.

Enseñanza de la programación

Desde el punto de vista de la enseñanza de la programación también hubieron muchos cambios a lo largo del tiempo, coexistiendo varios enfoques y tendencias. Aún hoy se puede verificar que no hay un consenso en los métodos a utilizar [4] y [5]. Algunas de las razones son que no existe un único método para la resolución de algoritmos así como tampoco un enfoque didáctico para materias introductorias que se haya impuesto por sobre otros o demostrado una indiscutible efectividad. Observamos que hay métodos de enseñanza que se fundamentan a partir de un paradigma de programación en particular como los son el paradigma funcional, el imperativo o el imperativo con el aporte de la orientación a objetos [6], [7], [8] y [9]. Además, dentro de un paradigma determinado se visualizan varios enfoques para enseñar: algunos enseñan a programar en un lenguaje de programación particular, utilizando su sintaxis y su semántica, y otros emplean un lenguaje algorítmico lo bastante general cómo para permitir su traducción posterior a cualquier lenguaje de programación. El primero de estos enfoques tiene el inconveniente de ligar los conceptos básicos de la programación a un lenguaje determinado, el cual tiene sus propias características y especificidades, perdiendo de vista los conceptos generales. Esto lleva a que existan varios cursos de enseñanza de programación que no son otra cosa mas que la enseñanza de los conceptos básicos pero en otros tantos lenguajes, con otras características y con otras especificidades. Así podemos encontrar en una currícula cursos de Pascal, C, FORTRAN, Ada, Clipper, C++, Java, Delphi, VisualBasic, etc. Dentro del segundo enfoque pueden observarse distintas tendencias en cuando al grado de formalidad y rigurosidad en el proceso de desarrollo de los algoritmos. Unos utilizan técnicas informales mientras que otros se apoyan en algún cálculo o formalismo lógico que posibilite verificación o la derivación formal de programas.

En el caso de nuestra asignatura, podemos encuadrar la enseñanza de la programación dentro del último enfoque presentado anteriormente, adoptando una tendencia semi-formal, es decir, dentro del paradigma imperativo con uso de un lenguaje algorítmico estructurado de carácter general que contiene los tres tipos de estructuras básicas (secuencial, condicional e iterativa), tipos simples y estructurados, más algunos elementos para manejar abstracciones (acciones, funciones y módulos) y donde se utilizan especificaciones semi-formales como parte de la etapa de análisis. Sin embargo, creemos importante destacar que arribamos a esta idea luego de transitar por otros caminos, aunque siempre dentro del mismo enfoque. Este proceso fue iniciado a fines de la década del '90 [10] y [11] donde se introdujo con mucho énfasis la derivación de las acciones del algoritmo a partir de pre y pos condiciones formales del problema a resolver. La idea fue tomada de Gries [12], este autor, empleando un lenguaje imperativo reducido denominado small, desarrolla un método para derivar el programa partiendo de especificaciones lógicas. Este procedimiento fue

llevado a nuestra asignatura. Para ello el planteo del problema requiere como primer paso la construcción de predicados lógicos mediante una notación adecuada, esto implicó la introducción temprana de conceptos de lógica. Cómo resultado de esta metodología se obtuvieron magros resultados académicos en relación a la calidad de los aprendizajes de algorítmica y programación [5] que condujeron a un rediseño de las estrategias metodológicas dando lugar al enfoque actual, el cual hasta el momento arroja datos alentadores.

La enseñanza de los principios básicos de algoritmos y lenguajes se lleva a cabo en la asignatura Introducción a la Algorítmica y Programación, materia anual de primer año. Ésta tiene como objetivo iniciar al alumno en la resolución de problemas cuya solución se pueda escribir en términos de un algoritmo. Posteriormente este algoritmo es traducido a un lenguaje de programación y ejecutado en una computadora. Esta materia es una de las bases fundamentales de la currícula de las carreras de Analista en Computación, Licenciatura y Profesorado en Ciencias de la Computación.

La metodología que empleamos consiste en la ejecución de distintas fases que en resumidas cuentas implican la elaboración de un algoritmo que expresa la solución del problema planteado en un lenguaje algorítmico. Una vez obtenido el mismo se traduce a un lenguaje de alto nivel y se implementa en computadora. Sobre ella se realizan los test de prueba. La obtención del algoritmo involucra un proceso de análisis descendente que puede determinar el diseño de subprogramas, módulos, acciones o funciones.

Una de las características que hacen de nuestro enfoque un caso poco habitual es el hincapié depositado en las estructuras iterativas y las buenas prácticas para resolver problemas de involucren tratamiento de secuencias de objetos. En ello trabajamos arduamente en pos de encontrar estrategias didácticas adecuadas que reduzcan la sobrecarga cognitiva a la hora de afrontar los primeros problemas de este tipo y posibiliten comprenderlos fácilmente sin la necesidad de incluir desde el comienzo las estructuras de representación como arreglos, listas enlazadas, etc. En este sentido nos basamos en la que algunos denominan “escuela francesa”, enfoque propuesto por Scholl y Peyrin [8] y que fuera introducido por éste último en nuestro país a fines de la década del ochenta. Dicho enfoque, con algunas mejoras y modificaciones ha demostrado su vigencia a lo largo de los años. Particularmente hemos tenido el agrado de discutir personalmente con Peyrin sobre éste y otros enfoques durante su visita en febrero de 2006 a la 13º Escuela de Verano de Ciencias Informáticas (RIO 2006).

Temas abordados

Antes de analizar los temas que se trabajan en la asignatura *Introducción a la Algorítmica y Programación*, queremos destacar que los estudiantes que ingresan por primera vez a la Universidad deben realizar de manera obligatoria un *Curso de Ingreso*. A continuación presentaremos las características de dicha actividad.

Curso de Ingreso

El curso de ingreso está dividido en tres áreas o “materias”: *Resolución de Problemas*, *Lógica* y *Matemática*. Ésta división fue realizada de manera tal que permitiera al alumno ingresante incorporar y completar aquellos conocimientos básicos necesarios para las asignaturas de primer año (*Cálculo*, *Lógica* e *Introducción a la Algorítmica y Programación)*. Los docentes responsables de cada materia de primer año fueron los principales encargados de seleccionar y ajustar el contenido de cada curso. Uno de los objetivos del ingreso es lograr una nivelación de contenidos brindando las herramientas necesarias para una mejor comprensión de las asignaturas iniciales de la carrera. El cursillo tiene una duración de tres semanas. El material para la materia Resolución de Problemas está conformado por cuatro capítulos: el primero realiza un recorrido histórico acerca de la evolución de la computación, el segundo trata conceptos referentes a los sistemas operativos más

utilizados actualmente (Windows y Linux), el tercer capítulo introduce Internet y sus servicios, finalmente, el capítulo cuarto presenta un Pseudo-Lenguaje de programación, el lenguaje TIMBA. Éste es un lenguaje que a través de un personaje que sólo comprende una cantidad pequeña de órdenes, manipula pilas de cartas permitiendo la construcción de algoritmos utilizando las tres estructuras básicas (secuencial, condicional y iterativa). Los problemas planteados en este capítulo son de una complejidad mínima, los enunciados son del tipo "... tomar una carta de PilaCartas y si es impar colocarla en PilaImpares de lo contrario colocarla en PilaPares ...". El lenguaje TIMBA se utiliza como una estrategia para que el ingresante piense la solución de un problema en forma estructurada, de manera tal que pueda comprender la relación entre un procesador y las órdenes que interpreta. Por razones de tiempo en este capítulo solo se trabajan las estructuras secuenciales y condicionales, sumado a la prueba de escritorio que puede ser realizada con un mazo real de cartas y el propio alumno oficiando de procesador.

Introducción a la Algorítmica y Programación

Los contenidos de aprendizaje abordados en la asignatura son los siguientes:

Unidad 1: Esquema funcional de una computadora: procesador, dispositivos de entrada y de salida, memoria, software y hardware, programas y lenguajes de programación. La problemática del desarrollo de software: características de un buen programa, fracasos notables, la crisis del software, métodos de desarrollo de programas, métodos informales y métodos formales, pasos en la solución de problemas. Definiciones de acción, proceso, algoritmo, programa, información, estado, especificación, léxico. Máquinas abstractas y reales (computadoras).

Unidad 2: Metodología de desarrollo de programas: método descendente para la solución de problemas, descomposición en subproblemas, composición de soluciones parciales. Paradigmas de programación. Notación algorítmica: algoritmo, acciones básicas (nada, asignación, leer, escribir). Objetos (constante y variables), tipos y valores. Tipos de composición: secuencial, condicional e iterativa. Composición secuencial. Tipos, variables y valores. Tipos simples. Análisis por casos y composición condicional o alternativa. La forma "según", "si entonces sino", "si entonces".

Unidad 3: Abstracciones: acciones y funciones. Parametrización: parámetros (formales, actuales), pasajes de parámetros (referencia, valor, resultado). Acciones: noción, acciones parametrizadas, Funciones: noción intuitiva de función, dominio y rango de una función, descripción de funciones por algoritmo (intencional) o por tabulación (extensional), composición de funciones. Efectos colaterales.

Unidad 4: Secuencias: Características de una secuencia. Notación y operaciones sobre secuencias. Primer modelo de acceso secuencial: máquina de tratamiento de secuencias de caracteres con marca final. Composición iterativa: iteración "mientras". Estados intermedios de un ciclo. Invariante de un ciclo. Esquemas de tratamiento y búsqueda secuencial.

Unidad 5: Segundo modelo de acceso secuencial: máquina de tratamiento de secuencias de caracteres de último elemento. Esquemas de tratamiento y búsqueda secuencial en este modelo. Composición iterativa con la forma "repetir-hasta". Composición iterativa con la forma "para".

Unidad 6: Implementación de algoritmos: traducción de notación algorítmica a lenguaje Pascal, sentencias Pascal, composición de sentencias, procedimientos y funciones y tipos. Reglas y convenciones para la escritura de código (notación, identación, comentarios, etc.)

Unidad 7: Estructuras de datos: tipos estructurados. Tipos homogéneos: arreglos unidimensionales y bidimensionales, conjuntos. Tipos heterogéneos: registros.
Almacenamiento persistente: archivos, tipos de archivos. Acceso secuencial. Acceso directo. Gestión de archivos (ABM).

Unidad 8: Abstracción: creación de distintos niveles de abstracción para la solución de problemas complejos. Relación entre máquinas abstractas y Tipos Abstractos de Datos. Ventajas de la utilización de TADs, TADs más usuales: pilas, colas, listas. Implementación de TADs:

almacenamiento contiguo, colas circulares, estructuras encadenadas, estructuras doblemente encadenadas, utilización de un elemento ficticio en la cabeza de la estructura. Implementación de TADs usando Units del lenguaje Turbo Pascal.

Unidad 9: Métodos de búsqueda: algoritmos de búsqueda secuencial y búsqueda dicotómica o binaria. Métodos de ordenamiento: ordenamiento por selección, por intercambio, por inserción.

Unidad 10: Recursividad: Introducción. Concepto. Funciones y acciones recursivas en el paradigma imperativo. Recursión directa e indirecta. Profundidad. Algoritmos recursivos.

Unidad Transversal: Convenciones o estilos de escritura de algoritmos y programas Pascal. Nociones de Pascal.

Enfoque

El orden cronológico de los temas es correspondiente con las unidades siendo la práctica en computadora la única unidad transversal que se introduce "bajo demanda", es decir, los temas abordados en cada una de las unidades arroja algoritmos escritos en lenguaje algorítmico que luego son traducidos al lenguaje Pascal, aplicando las convenciones o estilos de escritura previamente acordados y probados en computadora. Este orden se fundamenta en la idea de introducir paulatinamente cada uno de los conceptos principales y aplicarlos en la solución concreta de un problema hasta llegar a su ejecución y prueba en computadora, de manera tal que se pueda visualizar todo el proceso de solución de forma integral e ir introduciendo nuevos conceptos a medida que se complejiza la solución de los problemas. Es por ello que a partir de la segunda semana de clases los alumnos ya están implementando pequeños programas en Pascal.

La línea de ideas que se sigue se fundamenta en el concepto de algoritmo, la notación algorítmica (pseudo-lenguaje), la composición secuencial de acciones, variables, constantes, tipos simples y el método para solucionar problemas, basado en el método descendente.

La metodología que empleamos se fundamenta en la elaboración de un algoritmo que resuelve el problema planteado, este punto es el paso previo al desarrollo del programa, se resuelve el problema de manera general, usando una notación algorítmica de alto nivel y sin necesidad de preocuparse por los detalles de programación propios de cada lenguaje. De esta manera se consigue independencia respecto de los lenguajes de programación, evitando encasillar al alumno en un lenguaje en particular. Por otra parte es de gran importancia que el alumno reconozca que la solución de un problema es una actividad metódica, y como tal, involucra una serie de etapas. Estas etapas, debidamente documentados, permiten recorrer un camino que lleva a la solución correcta una vez concluida la tarea, a la vez que permitirán realizar revisiones en caso de existir algún error y retomar el trabajo. En nuestro caso las etapas que utilizamos para resolver un problema, a las cuales también hemos llamado fases, son: *análisis, diseño, implementación y prueba*. Durante el análisis se determinan los datos de entrada, de salida, consideraciones adicionales (fórmulas, tratamiento secuencial o no, etc.) y una especificación (pre y poscondición del problema). En la etapa de diseño construimos el algoritmo utilizando el método descendente, es decir que se realiza la partición del problema, la descomposición en subproblemas de menor complejidad, luego el refinamiento de cada uno de ellos, o sea la construcción de un algoritmo que lo resuelva, que podrá convertirse en un módulo, acción o función, y finalmente la composición de los algoritmos obtenidos en uno solo. Una vez obtenido el algoritmo, finalizando la etapa de diseño, se pasa a la etapa de implementación. En la implementación se traduce el algoritmo al lenguaje Pascal y se compila. Finalmente aborda la última etapa del proceso, ejecución y prueba. Es esta etapa se plantean los casos de test para el problema y se llevan a cabo. En caso de detectarse errores se vuelve a la etapa correspondiente, generalmente implementación o diseño si el problema es de carácter estructural.

La organización de la estructura de programas que se van integrando en el proceso de aprendizaje comienza con la composición secuencial. Se elaboran algoritmos cuya solución es

exclusivamente secuencial. En segundo término se introduce el concepto de composición condicional, mediante las formas “según”, “si entonces sino”, “si entonces”. Se hace notar que el “según” en la forma más general, de gran potencia expresiva pero que no está soportada en Pascal. De allí que se analizan reglas de equivalencia para traducir el “segun” a “si entonces sino” como paso previo a la implementación en Pascal. El “según” posibilita encontrar soluciones simples y concisas, permitiendo focalizar la atención en el problema a resolver sin perderse ni marearse con estructuras a “si entonces sino” anidados.

El paso siguiente es introducir los conceptos de abstracción y modularización. Se trabaja en primer lugar con acciones sin parámetros. Luego se incorpora el tema de pasaje de parámetros (entrada, salida, entrada-salida y referencia) y se aplica a las acciones y luego a funciones. Se hace hincapié en la necesidad de parametrizar para conseguir independencia en los módulos y en evitar los efectos colaterales cuado se utilizan funciones. Estos conceptos permiten aplicar el método de resolución de problemas en toda su magnitud, fundamentalmente lo referido a la etapa de diseño (partición, refinamiento y composición).

Una vez concluidos estos temas nos introducimos en el que consideramos un tema central, nos referimos a la composición iterativa. Aquí analizamos los dos tipos de estructuras, aquellas que se utilizan cuando no se puede determinar a priori la cantidad de iteraciones y aquellas donde si se conoce de antemano la cantidad exacta de iteraciones que se van a realizar. En las primeras analizamos el mientras (similar al While de Pascal), el repetir (similar al Repeat de Pascal) y el iterar. Al igual que sucede con el “según” se hace hincapié en que el iterar es la estructura más general, que contiene a las otras dos y se presentan las reglas de equivalencia para traducir unas en otras. Dentro de las segundas presentamos el para, una estructura de características muy similares al for de Pascal pero con la posibilidad de tener como paso cualquier número entero.

Luego de haber presentado todos los conceptos algorítmicos básicos, y antes de introducirnos en temas como estructuras de datos, archivos, TAD’s, etc, centramos la atención en el tema del tratamiento de secuencias. Tal vez éste sea uno de los puntos que distinguen fuertemente nuestro enfoque de otros similares. Partimos de la idea que existe un gran universo de situaciones problemáticas que involucran la manipulación de secuencias de objetos o el tratamiento secuencial de los mismos. Estas situaciones generalmente involucran para su solución la utilización de bases de datos, archivos, cadenas de caracteres, arreglos, listas enlazadas en memoria dinámica, datos ingresados por alguna entrada estándar, entre otros. Además las estructuras iterativas, tales como el *mientras*, *repetir*, *iterar* y *para*, si bien pueden utilizarse para múltiples propósitos, un uso muy común de las mismas es precisamente para el tratamiento de secuencias de objetos ya que en general hay que realizar recorridos sobre las mismas.

Ya que nuestro objetivo es enseñar a tratar secuencias adecuadamente, nos centramos en el estudio del concepto de secuencia y de las "buenas prácticas" en el uso de las estructuras iterativas para su tratamiento [13]. El objetivo principal es minimizar los errores que se comenten cuando usamos iteraciones. Por ello presentamos los esquemas algorítmicos fundamentales propuestos por Peyrin y Sholl. Ellos parten de la idea que las distintas soluciones sobre secuencias tienen un patrón en común, que ellos llaman esquema, y que opera como una plantilla o guía para encontrar un algoritmo que resuelva el problema. Los esquemas fueron ideados para hacer más sencillo el método de *diseño descendente* (en aquellos problemas que involucren recorrido de secuencias y ciclos) y para reducir la cantidad de errores pueden ocurrir al realizar este tipo algoritmos. Los esquemas "imponen" una manera de realizar un primer nivel de partición, el refinamiento y la composición. Es decir que ofrecen un "estandar" o "plantilla" para realizar de manera simple y rápida el *diseño descendente*.

Los esquemas son un método para construir algoritmos que se fundamentan en el hecho de que toda iteración, leída a posteriori, puede ser interpretada como la enumeración de una serie de acciones. En tal enumeración se accede a cada uno de los objetos de la secuencia y, para cada uno

de ellos, se aplica un tratamiento único, dependiendo básicamente del objeto y del problema a resolver. Se basan en dos hipótesis: el acceso a los elementos puede ser descripto en término de las acciones: *inicialización de la adquisición de los elementos* y *obtener siguiente elemento*, y el tratamiento de los elementos viene descripto en términos de las siguientes acciones: *inicialización del tratamiento*, *tratamiento del elemento corriente* y *tratamiento final*. Se reconocen tres tipos de problemas sobre secuencias que abarcan a toda la familia de problemas: recorrido (R), búsqueda (B) y recorrido parcial (RP, recorrido + búsqueda). A su vez, dentro de los problemas de recorrido se distinguen el esquema para el tratamiento integrado de la secuencia vacía y del primer elemento (R1), el esquema para el tratamiento especial de la secuencia vacía (R2) y el esquema para el tratamiento especial del primer elemento (R3).

Esta familia de esquemas se aplica a dos tipos de secuencias: secuencias con marca inicial y secuencias con marca final. Cada tipo de secuencias tiene sus propios esquemas. Se entiende que toda secuencia de objetos se representa de alguna de las dos formas.

A los efectos de poder desarrollar algoritmos que utilicen secuencias, sin necesidad de involucrarnos todavía con las diferentes alternativas de implementación de las mismas como pueden ser los arreglos o las listas enlazadas, se utilizan máquinas abstractas que nos facilitan la representación, es decir el soporte, y la manipulación de las secuencias, el acceso a sus elementos y el recorrido [14]. Estas máquinas abstractas, diseñadas tanto para secuencias con marca inicial como final, están implementadas como UNITS de Pascal y posibilitan a los alumnos manipular secuencias de caracteres y de números enteros sin necesidad de conocer aún la forma de representar secuencias en la computadora. Las máquinas abstractas están definidas a partir de una serie de primitivas similares a la máquina de creación y a la máquina de consulta [3], pero simplificadas e integradas. Hay 4 máquinas, dos para cada modelo secuencial: MaqMarca, MaqNumMarca, MaqUltimo y MaqNumUltimo. Cada máquina posee tres primitivas, en el caso de las máquinas para procesar secuencias de caracteres se denominan: CargarCinta, Arr, Av y, en el caso de las máquinas para procesar secuencias de números enteros se denominas CargarCintaN, ArrN, AvN [14]. Con estos temas finaliza el primer cuatrimestre.

Como cierre los alumnos desarrollan en grupos, de no más de tres integrantes, un proyecto integrador de los temas trabajados, de mediana complejidad y que deben entregar al reinicio de clases en el segundo cuatrimestre.

Una vez realizado el proyecto y afianzados los conceptos de algorítmica, abstracciones, los tres tipos de composición (secuencial, condicional e iterativa) y la manipulación disciplinada de secuencias, nos introducimos en los tipos estructurados (registros), en las estructuras de datos lineales (arreglos, listas) y en archivos. En los temas de arreglos, listas y archivos vemos de que manera se pueden usar para soportar secuencias y como se le pueden aplicar los esquemas trabajados con anterioridad (R1, R2, R3, B y RP). El trabajo con estos temas lleva la mayor parte del segundo cuatrimestre.

Finalizados estos temas se introduce el concepto de TAD's: pilas, colas, listas y diccionarios. Se trabaja primero con la definición de los mismos y luego con la implementación. Primero se provee a los alumnos de UNITS que implementan estos TAD's en Pascal de manera tal que pueden utilizarlos para resolver problemas. Luego se pasa a tratar las diferentes formas de implementar los TAD's, con arreglos y con listas, y los alumnos implementan sus propios tipos abstractos de datos.

Sobre el cierre de la materia se trabajan los temas de búsqueda (secuencial y dicotómica) y ordenamiento de elementos (burbuja, selección e inserción), ambos sobre arreglos de Pascal.

Por último se aborda el tema de recursividad, trabajando con funciones y acciones recursivas en el paradigma imperativo, sobre arreglos y listas.

Dada la extensa lista de temas que se abarcan hemos renunciado a incluir en este curso algunos otros temas que pueden encontrarse en otras currículas similares de primer año de carreras

de Licenciatura o Ingeniería como puede ser: árboles, tablas de hashing, grafos, programación orientada a objetos o programación orientada a eventos.

Formas metodológicas de enseñanza y aprendizaje

En la asignatura se trabaja con clases teóricas y clases prácticas. Para las clases prácticas los alumnos son distribuidos en comisiones de 30 alumnos. Las clases teóricas son semanales y de tres (3) horas de duración por semana. Las clases prácticas también son semanales y de cuatro (4) horas de duración por semana. En ambos casos se cuenta con clases de consulta de dos (2) horas de duración por semana y clases de implementación/repaso de dos (2) horas de duración por semana. Para la actividad práctica se dispone de cuatro (4) horas por semana reservadas exclusivamente para el acceso al laboratorio de informática fuera de los horarios de clase.

Además se utiliza un aula virtual disponible en la web para brindar información y materiales a los alumnos y e-mail para la comunicación diaria entre el equipo docente.

Resultados

El enfoque presentado anteriormente se viene aplicando desde 2004 y obedece la necesidad que se planteó en ese momento de realizar cambios pedagógicos, metodológicos, didácticos y organizacionales producto de los magros resultados obtenidos por los alumnos en cuanto a la calidad de sus aprendizajes, la alta deserción y el bajo rendimiento académico [5], [9]. Así fue que, aprovechando que las Secretarías de Ciencia y Técnica y Académica de la Universidad Nacional de Río Cuarto impulsaron los Proyectos de Innovación e Investigación para el Mejoramiento de la Enseñanza de Grado (PIIMEG), presentamos esta propuesta y la desarrollamos en el marco de un proyecto que posibilitó realizar, durante 2004 y 2005, un análisis minucioso de los aciertos y errores del mismo, proponer mejoras y volver a analizar los resultados en un proceso de investigación-innovación.-acción. Tanto el proyecto de 2004 denominado *Proyecto para la Mejora de la Enseñanza y el Aprendizaje de la Algorítmica y la Programación* como el proyecto de 2005 *Aprender a programar disciplinadamente: nuevos desafíos* fueron aprobados para su ejecución y obtuvieron en sus respectivos informes finales la aprobación con la escala de muy logrado (máxima calificación posible) en todos los criterios evaluados.

En líneas generales podemos decir que se evidencian mejoras apreciables en los últimos dos años respecto al porcentaje de alumnos que regularizan la materia, 31% en 2004 y 30% en 2005, contra el 20% en 2001, 21% en 2002 y 21% en 2003. No obstante cómo se puede leer estos números indican que todavía hay mucho esfuerzo por realizar. Con relación a estos porcentajes cabe consignar que un porcentaje del orden del 30% de la inscripción (34% en 2004 y 32% en 2005) corresponde a alumnos que abandonan la materia sin agotar todas las instancias de evaluación disponibles. Por otra parte se realizan encuestas a los alumnos ingresantes y a los recursantes, a través de ellas pudimos verificar una apreciación positiva de aquellos con respecto a los cambios de enfoque en la materia como así el grado de satisfacción que ellos manifiestan tanto en la organización de la materia, cómo en los temas trabajados, en las clases teóricas y prácticas, y en la práctica en computadora.

Aunque hay muchos resultados parciales de los que podemos estar dando cuenta entendemos que un aspecto a resaltar a sido la estrategia de convertir la enseñanza de la materia en una investigación activa para la cátedra, dándonos así la posibilidad de mirar nuestras prácticas cotidianas con mirada crítica y a la vez tomar nota de ésta actividad lo que nos ha permitido realizar un seguimiento sistemático en el tiempo el que sirvió para reconocer las fallas y los aciertos.

Conclusión

Hemos expresado en este trabajo cuales han sido los caminos recorridos en la enseñanza de la algorítmica en nuestro caso, y dónde estamos hoy. Nuestra intención es contar nuestra

experiencia que vemos cómo perfectible pero por sobre todas las cosas pensamos que es una experiencia acumulada en el tiempo y aquilatada por la permanente intención de la superación. La enseñanza de la algorítmica es todavía una materia que se sigue gestando en la informática y aunque sus contenidos mínimos están más o menos acordados, sus métodos todavía están en vías de consolidación.

No escapa a nuestros análisis que estamos insertos en una realidad social y educativa que acusa la crisis de su propia identidad y que en cierta medida la deserción de alumnos que se producen en el comienzo de cada año está estrechamente a esa realidad.

Entendemos que una revisión crítica de nuestras actividades en el marco de un proyecto de investigación evaluativa, crea el espacio necesario para documentar las acciones de la cátedra, para entender y reconocer los aciertos y los errores, y es la base necesaria para introducir innovaciones a la enseñanza.

Referencias

[1] Wirth, N. Algoritmos + Estructuras de datos = Programas. Ed. del Castillo S.A., 1980.

[2] López, L. Programación Estructurada en Turbo Pascal 7. ISBN 970-15-0075-X. Alfaomega. 2005.

[3] Pressman, R. Ingeniería del Software. Un Enfoque Práctico. España: McGraw-Hill. 1993.

[4] Dasso, A. Teaching Programming. Primeras Jornadas de Educación en Informática y TICS en Argentina. Bahía Blanca. pp 183-187. 2005.

[5] Ferreira Szpiniak, A. and Rojo, G. Cambios metodológico-didácticos y evaluación del impacto de los mismos en un curso introductorio a los conceptos de algorítmica y programación. Primeras Jornadas de Educación en Informática y TICS en Argentina. Bahía Blanca. pp 210-216. 2005.

[6] Ferreira Szpiniak, A. and Medel, R. and Luna, C. Our Experience Teaching Functional Programming at University of Río Cuarto (Argentina). SIGCSE Bulletin of ACM, 1998, Vol, 30, pp. 28-30. ACM Press. New York, NY, USA ISSN:0097-8418. 1998.

[7] Ferreira Szpiniak, A. and Medel, R. and Luna, C. Una propuesta de Integración de nociones Lógico-Matemáticas en la enseñanza de la Programación. III Congreso Argentino de Ciencias de la Computación. CACIC'97. Vol, 2, pp. 881-892. La Plata. 1997. ISBN 950-34-102-X. 1997.

[8] Scholl, P.C. y Peyrin J.P. Esquemas Algorítmicos Fundamentales. Secuencias e Iteración. Ed. Masson, 1991.

[9] Ferreira Szpiniak, A. and Rojo, G. El desafío de favorecer los aprendizajes de los alumnos en conceptos básicos de algorítmica y programación. X Congreso Argentino de Ciencias de la Computación. CACIC'2004. Buenos Aires. 2004.

[10] Rosso, A y Guazzone, J. Errores que se cometen al resolver un problema con estructuras de repetición. I Congreso Agentino de Ciencias de la Computación. CACIC'95. Bahía Blanca. 1995.

[11] Rosso, A. y Daniele, M. Algunos errores sistemáticos detectados en el proceso de aprendizaje de la Algortmica. II Congreso Agentino de Ciencias de la Computación. CACIC'96. San Luis. 1996.

[12] Gries D. The Science of programming. Springer-Verlag, 1981.

[13] Ferreira Szpiniak, A. Resolución de problemas sobre estructuras de datos lineales. Proceedings of World Congress on Computer Science, Engineering and Technology Education, WCCSETE 2006. Sao Paulo, Brazil. ISBN 85-891120-31-7. 2006

[14] Ferreira Szpiniak, A. Máquinas abstractas como recurso didáctico para la construcción de algoritmos que resuelvan problemas de tratamiento de secuencias. Proceedings of World

Congress on Computer Science, Engineering and Technology Education, WCCSETE 2006. Sao Paulo, Brazil. ISBN 85-891120-31-7. 2006.