XIV Conferencia Latinoamericana de Informática
17avas. Jornadas Argentinas de Informática
e Investigación Operativa
Buenos Aires, Setiembre 1988.

Un Laboratorio de Estructuras de Datos

Jorge Villalobos
Mario J. Otero
Alejandro Quintero
Olga Mariño

Departamento de Sistemas y Computacion
Universidad de los Andes
Apartado Aereo 4976 - Telex 42343 Unand Co
Bogotá 1, D.E., COLOMBIA

Resumen: El propósito de este articulo es describir brevemente la metodología de enseñanza del curso Estructuras de Datos dictado en la Universidad de los Andes y presentar el ambiente de programación LED (Laboratorio de Estructuras de Datos) desarrollado en la Universidad como apoyo práctico del curso. Este laboratorio permite al estudiante "experimentar" con sus algoritmos y sus estructuras de datos, dandole facilidades gráficas para observar su comportamiento, medir su eficiencia y poder compararlo con otras posibles soluciones de un problema dado. Le permite ademas disponer de mecanismos para seguir en la practica el proceso de desarrollo y evaluacion que se sigue en la teoría. El laboratorio constituye el complemento práctico del libro "Estructuras de Datos: Un Enfoque desde Tipos Abstractos" [VQ088], publicado por el Departamento de Ingeniería de Sistemas y utilizado actualmente en varias universidades.

Palabras Clave: Estructuras de Datos, Tipos Abstractos de Datos, Ingeniería de Software, Ambientes de Programación, Lenguajes de Programación.

1. Introducción.

Una de las realizaciones importantes del Departamento de Ingeniería de Sistemas y Computación de la Universidad de Los Andes en los últimos tiempos, ha sido la de dar a la totalidad de su trabajo una sólida base teórica y formal. Es así como en algunos de los cursos ya se ha logrado obtener un material nuevo que refleja los últimos avances en esta área. Siguiendo esta línea de desarrollo, se ha publicado un texto especialmente diseñado para el curso de Estructuras de Datos [VQ088], en el cual se presenta el material que podríamos llamar "clásico" en la enseñanza de las estructuras de información, enfocado desde un punto de vista de tipos abstractos de datos (TADs). Este libro recoge las colaboraciones, el trabajo y la experiencia docente de varios profesores del Departamento de Sistemas de la Universidad, lo cual permite afirmar que el material que allí se presenta corresponde a una metodología bastante depurada, con alto grado de formalidad y a la vez práctica para definir tipos y manipular estructuras de datos de manera constructiva.

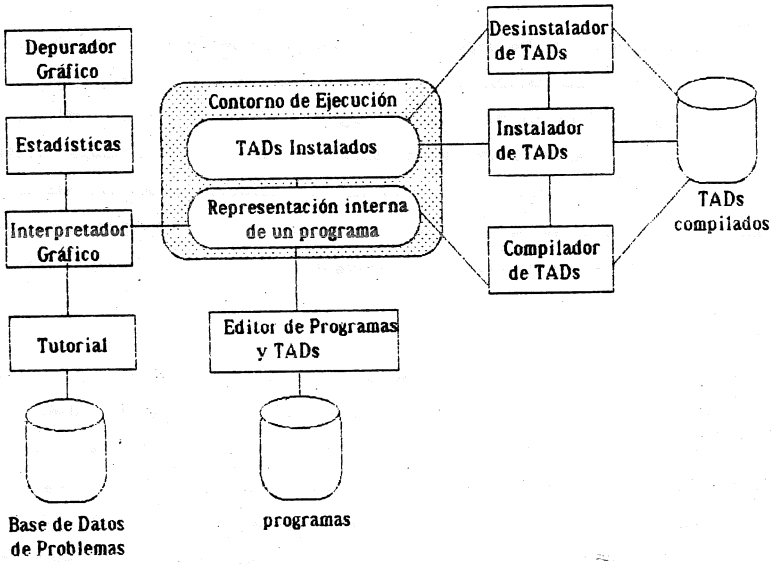
La metodología se basa en dos aspectos: una disciplina de diseño a través de Tipos Abstractos de Datos y una disciplina de desarrollo mediante la utilización de técnicas formales de programación, en las cuales el lenguaje gráfico juega un papel vital. Este nuevo enfoque ha sido utilizado desde hace dos años con resultados realmente alentadores (en cada semestre se dicta para unos 200 estudiantes de diversas áreas de Ingeniería), y aunque no se contaba inicialmente con un componente práctico satisfactorio se ha podido probar la metodología con un conjunto significativo de estudiantes.

Desde el mismo momento en que se comenzó a utilizar este nuevo enfoque en la enseñanza de Estructuras de Información, se sintió la necesidad de contar con un laboratorio en el cual el estudiante pudiera "experimentar" con sus algoritmos y sus estructuras de datos, así como disponer de herramientas suficientes para seguir en la práctica el proceso de desarrollo que se presenta en la teoría. Dicho laboratorio fue desarrollado en el último año, y cuenta con un editor guiado por la sintaxis (que se adapta a la metodología de desarrollo y de diseño), un interpretador gráfico, un depurador gráfico, un manejador de TADs (compilador, interpretador, generador de datos de prueba, etc.), un manejador de estadísticas (dibuja la función tiempo contra

número de datos procesados, calcula el tiempo de ejecución de cada rutina, etc.), un módulo tutorial con un resumen del curso, y otras herramientas para el estudiante (traductor del lenguaje del laboratorio a Pascal o a C, etc.). Los anteriores modulos se describen en los numerales 3 a 8 de este articulo.

El objetivo de este articulo es describir superficialmente el enfoque del curso (para una descripción más detallada consultar [VQ088]) y mostrar el laboratorio incluyendo algunos aspectos de su implementación. Se presenta además el lenguaje utilizado, que corresponde a una extensión de LCG (Lenguaje de Comandos Guardados) descrito en [MAR88] [BOH86] y que se utiliza en la Universidad como primer lenguaje algoritmico en la enseñanza de programación. El lenguaje y la metodología de desarrollo están basados en el trabajo de Dijkstra y Gries [DIJ76] [GRI81].

El siguiente es un esquema de la estructura del laboratorio, que muestra los diferentes módulos que lo componen y sus relaciones:



Es importante reconocer el trabajo de los estudiantes Adriana Bucheli, Henry Castellanos, John Díz, Ricardo Gómez, Luis Fernando Leal, Sandra Sanchez, y Jorge

Restrepo, quienes fueron los encargados de implementar el laboratorio, así como agradecer el apoyo del Departamento en este proyecto.

2. Metodología del curso.

En el curso se utiliza una metodología formal de desarrollo de programas, en la que se abandona el método tradicional y se pasa a una programación mucho más "disciplinada" y efectiva. En esta metodología guiada por objetivos, el estudiante utiliza refinamiento por pasos y aserciones obteniendo al final, además del programa, la prueba de su corrección.

Las aserciones se utilizan con tres propósitos. El primero es para especificar. Se debe colocar una aserción antes y después del problema y de cada uno de los subproblemas en los cuales ha sido dividido (pre y postcondición). Estas aserciones definen de manera no ambigua el problema y la interfaz entre cada una de sus divisiones. En segundo lugar las aserciones sirven como herramienta de desarrollo formal que hace posible "calcular" un programa en lugar de "improvisarlo". Esto permite por ejemplo encontrar la condición de salida más adecuada de un proceso iterativo mediante la manipulación y observación de las aserciones (el invariante del ciclo y su postcondición), en lugar de hacerlo por pura intuición. De esta manera existe garantía de obtener programas correctos [BOH86]. El tercer propósito es la documentación. Al final, las aserciones se pueden utilizar como una descripción del programa haciendo innecesario el uso de comentarios (es posible entender el funcionamiento de un programa viendo únicamente sus aserciones intermedias).

Esta metodología ha sido adaptada a las necesidades específicas del curso, y se ha definido una semántica para el lenguaje gráfico, vital en la construcción de aserciones que manejan estructuras de datos. La experiencia ha sido tan positiva, que se está comenzando a utilizar la misma metodología de desarrollo desde el primer curso de programación, para el cual ya se desarrolló el respectivo material escrito [MAR88].

Con respecto al diseño, se hace énfasis en la necesidad de separar la parte del algoritmo que maneja las estructuras de datos de la parte que resuelve el problema en sí. Esto con el fin de garantizar modularidad, facilidad en el mantenimiento y flexibilidad al

cambio de especificación. Esta separación nos lleva a la siguiente ecuación (parafraseando la celebre ecuación de Wirth):

Programa = Estructuras de Datos + Algoritmo de Datos + Algoritmo de Control

donde el Algoritmo de Control es totalmente independiente de las estructuras de datos utilizadas para representar la información. Así, es posible portar la solución a otra representación interna al costo de reescribir únicamente los algoritmos de manejo de datos lo mismo que resolver un problema sin necesidad de comprometer la solución a un tipo especifico de representación. Siguiendo esa misma linea, es posible ver las estructuras de datos como objetos que sólo pueden ser manipulados por un conjunto limitado de operaciones, que los algoritmos de datos se limitan a ejecutar Una clase de objetos, unida a su conjunto de operaciones, forma un Tipo Abstracto de Datos (TAD), que puede ser portado de un programa a otro sin ningún problema. De esta forma se llega a la ecuación

Programa = TADs + Algoritmo de Control

que resume el enfoque del curso desde el punto de vista de diseño. Este enfoque ofrece entre otras ventajas, la posibilidad de reutilizar software (un TAD es perfectamente portable), lo cual lo hace aún más interesante.

En el curso se enseña al estudiante a diseñar, especificar, implantar y utilizar Tipos Abstractos de Datos como parte de sus programas y de sus soluciones, estudiando en detalle los correspondientes a Listas, Pilas, Colas, Arboles y Grafos. Un proyecto tipico del curso es resolver el problema del transporte sobre el TAD Grafo (o sea en terminos de sus operaciones), luego desarrollar varias implementaciones posibles de dicho Tipo Abstracto, calcular la complejidad de cada operación en cada implementación y finalmente probar en el laboratorio los algoritmos diseñados para comparar los resultados teoricos con los practicos. Más información sobre Tipos Abstractos se puede encontrar en [CAR86] [CAR87] [GUT77] [GUT78]

3. El Lenguaje LCC.

El lenguaje en el cual se encuentra basado el laboratorio es una extension del utilizado en los dos primeros cursos de programación. Incluye el nucleo básico de todos los lenguajes imperativos: un comando de asignación, uno condicional (if-fi) y uno de iteración (do-od). En la sintaxis toma muchas construcciones de Pascal, ademas de sus tipos de datos básicos (integer, boolean, char, pointer, array y record). Tiene las siguientes diferencias con respecto a los lenguajes tradicionales:

- no maneja variables globales (es una manera de exigir modularidad y buen diseño)
- se elimina la necesidad del ";" para separar instrucciones
- no requiere la utilización de bloques begin-end
- incluye aserciones como parte del lenguaje (las aserciones son verificadas cada vez que el control del programa pasa sobre ellas)
- permite asignación entre variables estructuradas (asigna por ejemplo un arreglo a otro copiando todo su contenido)
- permite retorno funcional de valores de cualquier tipo, aún estructurado
- no existe programa principal sino una rutina por la que comienza la ejecución (como en C)
- es perfectamente ortogonal; ésto facilita su aprendizaje (no existen excepciones con respecto a las reglas de construcción de expresiones. Si por ejemplo una función retorna un valor de tipo apuntador, es válido escribir f(a)↑.info, mientras que si retorna un arreglo es posible utilizar la expresion f(a)[5], lo cual es incorrecto en muchos lenguajes)
- maneja Tipos Abstractos Genericos de Datos. Es posible por ejemplo definir el TAD Lista[Elemento] que recibe como parámetro el tipo de los elementos que va a contener cada uno de sus objetos abstractos. En ese caso es valida la declaracion:

  var lst1: Lista[**integer**];
  lst2: Lista[**boolean**];

  en la cual se generan dos objetos que se comportan de manera similar, pero cuyo contenido interno corresponde a elementos de diferente tipo.
- maneja funciones polimórficas. Una misma función puede retornar diferentes tipos de datos. Esto es indispensable para el manejo de las operaciones de un TAD, en las cuales el tipo de retorno depende del objeto en si que sea pasado como argumento. Por ejemplo en las siguientes asignaciones:

val2 := InfoLista(lst1,5)

val3 := InfoLista(lst2,5)

donde InfoLista(lst,i) retorna el i-ésimo elemento de la lista lst. corresponden en el primer caso a una asignación de enteros y en el segundo a valores lógicos.

4. El Editor.

El editor del Laboratorio de Estructuras de Datos (LED) permite la escritura y modificación de programas escritos en LCG. Dispone de una interfaz notablemente ágil con los demas módulos del laboratorio, lo cual facilita la depuracion y validacion igualmente se utiliza para generar y editar nuevos TADs

De acuerdo con la metodología expuesta, la escritura de programas debe ceñirse a dos aspectos: primero, obviamente, al lenguaje LCG y segundo, a la disciplina de programación guiada por objetivos. Por estas dos razones, se ha desarrollado un editor de programas y de TADs guiado por la sintaxis del lenguaje, que además de indicar la construcción correcta de cada comando que el programador desea utilizar y de dar a todo programa generado una estructura estándar (reglas fijas de indentación, pretty-printing, etc), garantiza su corrección sintáctica

Para dar soporte al aspecto de la disciplina de programación utilizada, el lenguaje se ha enriquecido, a nivel de su gramática, para permitir la inclusión de aserciones (en terminos de las variables de estado del programa, de ciertos formalismos gráficos básicos y de un conjunto razonable de predicados definidos sobre los TADs de base) y el uso de éstas a manera de comandos. Con respecto a ésto último, no sólo se han incluido las aserciones en la gramatica del lenguaje para permitir su escritura dentro de los programas, sino que también son validables en tiempo de ejecución

Siendo un editor guiado por la sintaxis, valida en cada momento el comando suministrado con respecto a la definición gramatical correspondiente. Cualquier tipo de error en este sentido debe corregirse, para que el editor permita continuar y eventualmente terminar la edición de un programa o rutina

Para escribir una instrucción el programador puede seleccionar de un menú el comando del lenguaje que desea utilizar, siendo la respuesta del sistema el despliegue de la estructura general del comando, ésto es, resaltando la palabra o simbolo clave, escribiendolo en el nivel de indentación que lleva el programa, mostrando los nombres de sus diferentes componentes según lo indican los simbolos no terminales de la producción gramatical correspondiente, y colocando todo en la pantalla de acuerdo con la estructura y reglas de indentación adoptadas (pretty-printing)

Para agilizar el trabajo de las personas familiarizadas con el lenguaje, el editor permite escribir los comandos directamente, sin restricción alguna. Al terminar cada linea el editor la toma, la analiza y de ella extrae, valida y reorganiza, como en el caso anterior el comando o los comandos que la conforman.

Cualquiera que que sea la alternativa utilizada para escribir un comando, el editor presenta los posibles errores cometidos y permite y sugiere formas de corregirlos
Para citar un ejemplo, es posible trabajar bajo la modalidad de declaracion automática, en cuyo caso se reporta todo simbolo no declarado que aparezca como parte de un comando y se sugiere la declaración más adecuada de acuerdo con el papel que juega el símbolo dentro de la expresión, indicando además el punto del programa donde podria incluirse tal declaración. Obviamente existe la alternativa de editar el simbolo, puesto que puede tratarse de un simple error de mecanografía.

Además de implementar las funciones habituales de un editor de texto, tales como insertar, borrar, moverse sobre el texto en cualquier sentido, imprimir, etc., (siempre que su acción resulte en situaciones sintácticamente correctas), el editor permite, entre otras operaciones, recuperar la última versión grabada del texto (i.e., ignorar los últimos cambios hechos), copiar y mover bloques completos de texto siempre y cuando sean correctos y tengan sentido en el sitio donde se copian.

Otro aspecto importante que influye en el funcionamiento del editor, es que la unidad básica de trabajo en el Laboratorio es el procedimiento o rutina, por lo cual sólo se puede estar editando una rutina a la vez y se dispone de operaciones a este nivel para ver, abrir, cerrar y crear rutinas.

Para el trabajo con tipos abstractos, además de la edición que tiene las mismas características descritas para el editor de programas, es posible "instalar" y "desinstalar" TADs en un programa. Estos términos se refieren respectivamente a poner y a retirar del alcance del programa o rutina en edición, implementaciones particulares de los tipos abstractos que maneja. De esta manera es posible, por ejemplo, probar un mismo programa de aplicación (cuyo texto permanece intacto) con diversas implementaciones, lo cual es uno de los objetivos del curso.

El producto de la edición de un texto es el árbol de sintaxis correspondiente al programa LCG escrito por el usuario. Su representación se hace mediante árboles binarios ordenados para almacenar la información de constantes, tipos y rutinas. Las constantes y tipos son globales para todas las rutinas que forman el programa (incluida "main"), las cuales contienen a su vez el árbol sintáctico que las describe.

5. El Manejador de TADs.

El manejador de tipos abstractos de datos es el módulo del sistema que se encarga de tomar las estructuras generadas en la edición de un TAD y de compilar esta descripción. El código que aquí se genera queda a disposición de los diferentes programas, listo para ser "instalado" y utilizado por ellos. Los componentes principales del manejador de TADs son el compilador y el instalador/desinstalador.

Antes de compilar sus definiciones de tipos abstractos, el programador puede depurarlas y probarlas, colocando el conjunto de operaciones como parte de un programa LCG que se encarga de llamarlas. Finalmente, cuando el usuario está satisfecho con el código que ha escrito, hace una llamada al compilador.

Para compilar la definición de un TAD, se ha definido una máquina intermedia a cuyo lenguaje se traducen las estructuras sintácticas que representan el texto del programa LCG: el compilador toma únicamente el conjunto de funciones de la declaración y genera el código correspondiente, que consiste en una secuencia de instrucciones en código intermedio, organizadas en bloques que se almacenan en disco. Esta será la definición del TAD que pueden tomar posteriormente los diferentes programas e instalar en su ambiente de trabajo.

El módulo instalador y desinstalador de TADs está disponible desde diferentes modulos del sistema para hacer más flexible el manejo de los diferentes tipos de datos en el ambiente de laboratorio. Así por ejemplo, es posible utilizarlo desde el editor para crear un programa LCG que utiliza una definición determinada de un TAD, o bien desde el interpretador para alternar dinámicamente diferentes implementaciones de dicho tipo abstracto y comparar su desempeño.

6. El Interpretador Gráfico.

Este módulo ejecuta el código escrito por el usuario presentando el desarrollo de la ejecución a través de una interfaz que permite conocer con mucho detalle diversos aspectos de la ejecución, así como afectarla de varias maneras.

El interpretador presenta de manera simultánea el programa fuente que se está ejecutando, el efecto de la ejecución de las instrucciones sobre el estado del programa y los resultados que produce.

Existe entonces una ventana en la cual se muestra el programa, indicando el comando que se ejecuta en cada momento. Esta ventana se va desplazando sobre el texto fuente a medida que progresa la ejecución, siendo afectado este desplazamiento según el modo de ejecución seleccionado. El "modo programa" permite rastrear el punto de ejecución sólo dentro del programa principal (o rutina main) señalando el comando que se ejecuta o la llamada a procedimiento cuya implementación se está interpretando.

Por defecto, el modo de ejecución es "procedimiento a procedimiento", el cual se desplaza sobre el cuerpo de cada una de las rutinas que se ejecutan. Esto se logra abriendo una nueva ventana de texto por cada nueva rutina, mostrando el desplazamiento del punto de ejecución sobre ella y, finalmente, cerrándola al terminar la ejecución: ésto tiene un ventaja adicional: la de mostrar con el abrir y cerrar de ventanas el cambio de contextos de ejecución que, visualizado de esta manera, ayuda más que muchas explicaciones a la comprensión del concepto.

Finalmente, es posible ver la ejecución instrucción por instrucción (sea avanzando por unidades de tiempo determinadas, sea avanzando a pedido del usuario).

Cualquiera que sea el modo de ejecución, es posible detenerla temporalmente en puntos de interés, mediante la colocación de marcas especiales (señales de "pare") sobre el texto. En estos puntos de detención es posible consultar el estado del programa y evaluar expresiones en el contexto vigente.

En la ventana de resultados es posible ver las modificaciones que van sufriendo los diversos elementos de información que conforman el estado del programa, así como, en conjunción con el depurador, detener la ejecución en ciertos puntos para realizar operaciones o evaluar expresiones en el contexto y estado vigentes. Es de esta misma forma que se puede hacer recuperación de errores sin necesidad de cancelar el ambiente y editar el programa: afectando el estado del programa y retornando el control al interpretador.

Las aserciones son validables durante la ejecución y es posible controlar, a varios niveles, la verificación que de éstas se hace: desde ignorarlas por completo, hasta hacer que la ejecución aborte si una aserción determinada no se satisface.

Es posible también activar un mecanismo de reloj, que permite medir en tiempo virtual la ejecución de los programas y producir estadísticas de la ejecución de varios procedimientos a la vez. La información que se tiene de cada uno corresponde, para cada llamado, a la hora de comienzo y terminación y al número de instrucciones simples ejecutadas.

7. El Depurador Gráfico.

Para facilitar la depuración y mostrar al estudiante la ejecución de su programa de manera gráfica, el laboratorio cuenta con un depurador gráfico. Mediante éste es posible apreciar la manera como se modifican o recorren las estructuras de datos de una aplicación, lo mismo que preguntar por el valor de cualquiera de las variables involucradas en la solución e incluso editarlas. Es posible, por ejemplo, ver en una

ventana la manera como se va balanceando un árbol AVL, o la forma como se va recorriendo un grafo para buscar un camino entre dos de sus vértices.

El depurador funciona en dos modos: interactivo o automático. En el primer modo, el usuario puede preguntar, durante la ejecución, por el valor de cualquier variable o por la representación gráfica de cualquiera de sus estructuras de datos, y puede interactuar con ellas. En el modo automático, el sistema va mostrando cómo afectan las estructuras de datos a medida que progresa la ejecución.

8. El Tutorial.

El carácter de aplicación educativa que tiene el Laboratorio de Estructuras de Datos hace imprescindible la existencia de este módulo, que tiende a satisfacer varios objetivos.

Primero, servir de repaso o "guía rápida" del material del curso. De esta manera un estudiante puede ver una presentación rápida del tema que le interesa y donde el sistema puede proponerle diferentes tipos de ejercicios que refuercen y/o evalúen su comprensión.

Por otra parte, le permite obtener explicación sobre errores cometidos en otras partes del sistema. Efectivamente, al ocurrir errores en módulos como el editor, el depurador o el interpretador, el estudiante tiene la alternativa de entrar al tutorial para obtener explicaciones más detalladas y precisas. Estas explicaciones, que se dan en el mismo contexto donde cometió el error, van acompañadas de ejemplos o de ejercicios que ayudan a aclarar las fallas ocurridas y muestran la forma correcta de expresarse en situaciones similares.

En una situación como la que se acaba de describir, el ambiente de ejecución no se ha perdodo y por lo tanto, una vez aclarada la fuente de los errores, el estudiante puede corregir los errores y retornar el control al interpretador.

Por último, el tutorial puede plantear retos de complejidad creciente que ayuden al estudiante a reforzar los conceptos del curso y a evaluar su propio dominio del tema.

9. Conclusiones

La ausencia de un laboratorio donde se pudieran manejar los conceptos con el nivel de abstracción apropiado, hacía que las sesiones prácticas previstas para el curso de Estructuras de Datos fueran apenas suficientes para la escritura en lenguaje Pascal (y validación por el método de prueba y error!!) de las operaciones básicas de los tipos de datos, logrando en muy pocas oportunidades llegar a la fase de probar programas de aplicación en términos de dichos TADs. Además, y casi más grave aún, no existía ningún tipo de herramienta para expresar (y verificar la correcta aplicación de) la disciplina de especificación y desarrollo descrita.

Tal como está implementado, el laboratorio presenta múltiples ventajas: permite que el estudiante ejecute sus soluciones a los problemas directamente en lenguaje algorítmico evitando la traducción a (y la dependencia de) un determinado lenguaje de programación; le permite utilizar y "ver" los conceptos en el mismo lenguaje gráfico al que está habituado en las sesiones de teoría, comparar el comportamiento de la misma solución implementada bajo diferentes formas y ver cómo su programa realmente satisface los objetivos (tanto los intermedios como el objetivo final) que su método de solución al problema había planteado.

Por último, el Laboratorio de Estructuras de Datos, cuya implementación en máquinas Apple Macintosh Plus comienza a probarse en agosto de 1988, corrige una de las principales deficiencias que tuvo el curso durante algún tiempo: la falta de un ambiente apropiado, en el cual se pudieran realizar prácticas de los temas presentados en las sesiones de clase. Por lo tanto, el Laboratorio de Estructuras de Datos constituye el complemento práctico al texto del curso, "Estructuras de Datos: Un Enfoque desde Tipos Abstractos", comformando así un paquete completo de material didáctico para la enseñanza del tema.

10. Referencias

[BOH86] Bohórquez, J., Cálculo de Programas: Una Metodología Precisa para el Diseño de Programas de Computador, Memo de Investigación #7, Universidad de los Andes, 1986.

[CAR86] Cardoso, R., Diseño e Implementación de Tipos Abstractos de Datos, Memo de Investigación #9, Universidad de los Andes, 1986.

[CAR87] Cardoso, R., Práctica en Tipos Abstractos de Datos, Memo de Investigación #17, Universidad de los Andes, 1987.

[DIJ76] Dijkstra, E.W., A Discipline of Programming, Prentice-Hall, 1976.

[GRI81] Gries, D., The Science of Programming, Springer-Verlag, 1981.

[GUT77] Guttag, J., Abstract Data Types and Software Validation, comm. ACM, vol. 21, num. 12, Diciembre/78.

[GUT77] Guttag, J., Abstract Data Types and the Development of Data Structures, comm. ACM, vol. 20, num. 6, Junio/1977

[MAR88] Mariño, O., Notas para el Curso Introducción a la Informática, Publicaciones de la Universidad de los Andes, Junio/88.

[VQ088] Villalobos, J., Quintero, A., Otero, M. Estructuras de Datos: Un Enfoque desde Tipos Abstractos, Publicaciones de la Universidad de los Andes, Enero/88.