HERRAMIENTAS NO CONVENCIONALES PARA EL APRENDIZAJE DE LA PROGRAMACIÓN

Perla Señas-Norma Moroni

Grupo InE
Departamento de Ciencias de la Computación
Instituto de Ciencias e Ingeniería de Computación
Universidad Nacional del Sur.
Av. Alem 1253 - 8000 - Bahía Blanca - ARGENTINA
ccsenas@criba.edu.ar - moroni@criba.edu.ar

Resumen

En la búsqueda permanente de nuevas estrategias metodológicas para la enseñanza de la programación insistimos con la idea de usar la computadora como un recurso natural para tal fin, en todas sus etapas. Si bien hemos creado y aplicado con éxito un entorno interactivo para el desarrollo de algoritmos por computadora, hemos apreciado falencias en las etapas extremas del mencionado proceso, a saber: comprensión acabada del problema y verificación del algoritmo.
Dicho entorno está basado en el paradigma imperativo y permite el trabajo en programación estructurada y modular, de acuerdo con las características de los cursos en los cuales se aplica. Cuenta con un editor interactivo de algoritmos y con un traductor automático de algoritmos a programas.
Proponemos en este trabajo complementar el entorno existente con las siguientes herramientas: Mapas Conceptuales Hipermediales (MCH) para la etapa previa al desarrollo del algoritmo y Esquemas de Ejecución de Algoritmos (EEA) para una etapa posterior en la que se realiza el testeo.
Los MCH y una plataforma diseñada especialmente para trabajar con ellos han sido creados con el propósito de facilitar la construcción, mantenimiento, interconexión y lectura de los mapas conceptuales tradicionales. Son una herramienta específica para la representación de las ideas y constituyen un recurso visual poderoso para la identificación de los conceptos fundamentales y de sus relaciones. El uso exitoso de los MCH en distintas áreas de aplicación, también incluye a la de resolución de problemas computacionales.
Los EEA son representaciones gráficas para visualizar la ejecución de un algoritmo. Ayudan a comprender la relación entre el algoritmo como entidad estática y el dinamismo de su ejecución. Por otra parte los EEA son una herramienta valiosa para la etapa de testeo y verificación.
Es así como introducimos un editor interactivo de MCH y un constructor de EEA en el entorno computacional para el aprendizaje de la programación.
El trabajo con el nuevo entorno permitirá al alumno crear el MCH del tema sobre el que trata el problema a resolver, lo que representa una ayuda significativa para la comprensión del mismo. Una vez comprendido el problema podrá usar el editor interactivo de algoritmos para construir los algoritmos que considere necesarios para la resolución del problema. Antes de pensar en la codificación, podrá usar el módulo de verificación, optando entre el constructor de trazas o el constructor de EEA. A partir de allí podrá optar entre el traductor automático a lenguaje Pascal, o usar un entorno específico para escribir programas en Pascal, según la metodología para el aprendizaje del lenguaje de codificación que se esté usando.

Palabras claves

programación - aprendizaje - MCH - EEA

HERRAMIENTAS NO CONVENCIONALES PARA EL APRENDIZAJE DE LA PROGRAMACIÓN

1. Introducción

Existe una búsqueda permanente de nuevas estrategias metodológicas para el aprendizaje de la programación. Se conocen resultados de numerosas experiencias y hay justificaciones que avalan cada una de las distintas posturas. Sostenemos que la computadora, que es propuesta como una herramienta útil para la enseñanza de otras disciplinas, es también una herramienta valiosa para la enseñanza de la programación en todas sus etapas. Es así como se puede introducir tempranamente a los alumnos en el medio ambiente de la máquina con la que van a seguir trabajando posteriormente [For96].

Consideramos el caso de un curso introductorio basado en el paradigma imperativo y siguiendo una metodología estructurada y modular. El alumno debe lograr una expresión precisa del problema a resolver, una estrategia de resolución, un algoritmo que plasme dicha resolución y por último el programa correspondiente en el lenguaje de programación elegido [Rue95].

El trabajo apunta a que el alumno desarrolle capacidades que le permitan encarar la resolución de un problema de manera eficaz y a que pueda resolver problemas cada vez más complejos. Es sabido que adquirir habilidades para escribir algoritmos no es una tarea trivial; resulta conveniente que los algoritmos se expresen en lenguaje natural y que se cuente con herramientas adecuadas que facilite su elaboración [Gei94].

En esta búsqueda de recursos metodológicos se ha acordado muchas veces sobre la importancia de disponer de un lenguaje algorítmico (menos rígido que un lenguaje de programación). Además con el propósito de salvar problemas advertidos al trabajar con algoritmos en una forma tradicional se diseñó y elaboró un entorno interactivo para el desarrollo de algoritmos. Este entorno controla con naturalidad la edición de forma tal que no se incurra en errores sintácticos ni errores semánticos. Permite invocar otros algoritmos como primitivas, lo que promueve el concepto de modularidad. Además el entorno se complementa con un traductor automático de algoritmos a programas lo que permite la aplicación de un método global para el aprendizaje del lenguaje de programación usado para la implementación.

Son ampliamente conocidas las ventajas que se obtienen al usar un lenguaje algorítmico:

- Permite centralizar la atención en la resolución del problema y no en los detalles propios de la rigidez de un lenguaje de programación.
- Se atenúan los inconvenientes que surgen en un curso integrado por alumnos con distintos niveles en el manejo del lenguaje de programación.
- Se pueden adquirir habilidades y conocimientos importantes para el desarrollo de tareas de programación, que podrán aplicarse independientemente del lenguaje con el que finalmente se codifique.

El uso del editor interactivo de algoritmos agrega las siguientes ventajas :

- La construcción de algoritmos resulta una tarea atractiva y motivadora.
- El autor no necesita memorizar esquemas ni reglas fijas.
- El alumno no abandona su uso una vez que ha aprendido el lenguaje de programación.

El uso del traductor automático al lenguaje de programación permite la aplicación en forma natural de un método global para el aprendizaje de dicho lenguaje. El uso de esta metodología en lugar de la tradicional tiene beneficios importantes sobre los tiempos de aprendiaje y sobre la complejidad de los problemas que se pueden manejar inicialmente.

1.1 Sobre el entorno existente

Se trata del entorno para el aprendizaje de la programación Cubik presentado en [Gar96]. Cuenta con un editor interactivo de algoritmos y un traductor de algoritmos a programas en Pascal.
El editor interactivo de algoritmos ofrece aquellos elementos necesarios para una programación estructurada y modular. Mediante su uso el alumno puede centrar su atención en cómo plantear el algoritmo, despreocupándose de los detalles de la escritura.
El traductor brinda la posibilidad de obtener el programa correspondiente a un algoritmo (y subalgoritmos involucrados) que se ha editado previamente. De esta manera el alumno puede obtener automáticamente el programa que él mismo debería escribir en una próxima etapa de codificación.
Para su desarrollo se puso énfasis en el uso de una interfaz amigable, gráfica y compatible con los nuevos Sistemas Operativos orientados a ventanas.
El objeto principal que provee su interface es la Ventana de Edición de Algoritmos. Dicha Ventana se encarga de realizar el Análisis Léxico, Sintáctico y Semántico de los Algoritmos cada vez que se agrega o elimina un elemento. Cuando el Analizador Sintáctico detecta que se ha ingresado una palabra clave para una estructura, agrega por sí misma el resto de la estructura de forma tal que resulta imposible la aparición de un error sintáctico. El mismo proceso ocurre cuando se ingresa un operador en una expresión.
El otro objeto que se utiliza es la Ventana de Edición de Textos que simplemente permite ver y modificar los subprogramas Pascal generados por el Traductor a partir de los Algoritmos creados en la Ventana de Algoritmos.
Cubik permite el manejo de Múltiples Ventanas, tanto de Edición de Algoritmos como de Edición de Textos.
Se realizaron experiencias en cursos iniciales de programación en los que se usó el entorno Cubik. Se obtuvieron resultados muy satisfactorios en relación a los obtenidos con la enseñanza tradicional.

1.2 Problemas que persisten

Si bien con la incorporación del entorno Cubik en el trabajo de un curso introductorio de programación se advirtieron mejoras metodológicas importantes, también se pudo observar que en un porcentaje no despreciable de alumnos persisten los inconvenientes para lograr una comprensión correcta y acabada de los problemas, como así también para realizar la verificación de los algoritmos.
Existen herramientas conceptuales y computacionales que podrían colaborar efectivamente en tal sentido. Para el primer problema mencionado se proponen los mapas conceptuales hipermediales (MCH) y una plataforma desarrollada especialmente para trabajar con ellos y para el segundo aspecto, las trazas, los esquemas de ejecución (EE) y los constructores que se proponen más adelante.

2. Herramientas para la comprensión del problema

Observando que la primera etapa en la resolución de un problema es la comprensión acabada del mismo, es natural pensar en los recursos disponibles para poder lograrla.

2.1 Mapas conceptuales hipermediales

Los MCH constituyen una valiosa herramienta para ayudar a los alumnos a lograr aprendizajes significativos, para lograr la comprensión de un tema en forma acabada, destacando los conceptos fundamentales, su jerarquización desde un punto de vista semántico, y las relaciones que existen entre ellos. Mantienen toda la riqueza educativa de los MC de Novak potenciada con los beneficios que brinda la tecnología hipermedial: mayor facilidad para el manejo operativo, mayor riqueza y versatilidad para la representación de la información y un mayor

atractivo desde el punto de vista motivacional, especialmente para los estudiantes jóvenes, [Señ96a].
En un MCH cada nodo de la hipermedia contiene una colección de no más de siete conceptos relacionados entre sí por palabras enlaces. A cada uno de estos nodos se lo denomina vista. Cada vista puede ser visualizada en una ventana.
Se distinguen dos tipos de conceptos: los propios de la vista y los importados a la misma. Los primeros corresponden a aquellos que constituyen inicialmente la vista y los segundos son los que se toman desde otra para poder así establecer relaciones entre conceptos de distintas vistas.
Las relaciones entre conceptos de una misma vista se denominan relaciones internas y las relaciones entre conceptos de distintas vistas se denominan relaciones externas.
En cada vista los conceptos propios se representan mediante elipses rotuladas, los conceptos importados por rectángulos rotulados, ambos con el nombre del concepto que representan, y las relaciones (internas o externas) por arcos etiquetados con palabras enlace.
Para representar las relaciones externas se establece un arco etiquetado entre el concepto propio y el concepto importado. Dicha relación debe figurar en ambas vistas.
Cada vista se identifica con un nombre, (el del concepto propio más abarcativo de dicha vista) y con un color que es usado en todas las elipses que representan conceptos propios. Los conceptos importados mantienen el color de la vista donde fueron definidos originalmente.
Un concepto C perteneciente a una Vista1 explota en otra Vista2 cuando dicho concepto C se desarrolla en la Vista2, es decir C constituye el nodo raíz del MC desarrollado en Vista2. En este caso se dice que el MC representado en la Vista2 es un submapa del correspondiente a la Vista1.
La elipse que representa un concepto propio que explota en otra vista es un botón elíptico que posibilita el acceso directo a esa vista. El rectángulo que representa un concepto importado C es un botón rectangular que permite acceder directamente a la vista donde C está definido como propio.
Distintas apariencias (gráfico, sonido, animación, etc.) se pueden asociar a un concepto terminal (que no es botón). Todo concepto puede estar asociado a una apariencia ya que siempre existirá una vista del MC donde dicho concepto esté definido como terminal.
Además esta tecnología computacional permite asociar una base de información hipermedial creada a partir de las fuentes que dieron lugar al MCH. El acceso a la misma se puede realizar durante la exploración del mapa. Se logra entonces que en un MCH aquellos conceptos que por su riqueza pueden aportar nueva información, tengan la posibilidad de ser explorados más profundamente usando una hipermedia. Se crean de esta manera dos planos: el plano principal o MCH propiamente dicho y uno secundario o de información ampliatoria que pueden consultarse alternadamente.
Con esta extensión los MCH trascienden las posibilidades educativas de los MC tradicionales de Novak. Los mecanismos propuestos hacen factible el acceso a las fuentes de información que dieron origen a la creación del mapa. Esta bibliografía hipermedial puede ser de suma utilidad para quien esté interesado en ampliar el tema, como así también para quien realice una evaluación del mismo, [Señ96b].

2.2 Sobre la Plataforma para MCH

El diseño de una plataforma específica para el tratamiento de MCH apunta a contar exclusivamente con aquellos recursos necesarios para el desarrollo de los mismos. Se evitan así posibles dispersiones en aspectos y opciones de implementación que carecen de importancia desde un punto de vista estrictamente educativo y que entorpecen el trabajo de abstracción, [Mor96a]. Así la atención del autor se centra en los aspectos más importantes y más ricos desde el punto de vista formativo: la elección de los conceptos, la clasificación por jerarquía y el establecimiento de las relaciones.
Se pensó en una plataforma específica para la creación y lectura de MCH cuya característica esencial fuese la facilidad de uso. Esta plataforma presenta dos modalidades de trabajo, una de ellas es el modo correspondiente al autor en el cual se elabora y modifica el mapa y la otra correspondiente al lector en la que éste puede ser recorrido e inspeccionado. Se permite una

interacción inmediata entre ambas modalidades. La modalidad de autor presenta características específicas para la creación de los MCH que liberan totalmente al autor de cualquier aspecto superfluo a la comprensión del tema.
Esta plataforma presenta las características adecuadas para convertirse en el editor de MCH para el nuevo entorno de programación que se está presentando.

3. Herramientas para la verificación de algoritmos

Para realizar la verificación de un algoritmo se puede optar entre la realización de una traza o la de un esquema de ejecución de algoritmo (EEA). En ambos casos se puede conformar una base de verificación para guardar el conjunto de valores de entrada y los correspondientes de salida.
Por lo tanto el Entorno de Verificación cuenta con un constructor de trazas y un constructor de EEA.

3.1 El Constructor de Trazas

El Constructor de Trazas permite optar entre la realización de una traza completa o de una simplificada

3.1.1 Traza completa

Una traza completa es un cuadro donde se plasman los datos que están especificados en un algoritmo y el estado de los datos en cada momento de su ejecución.
El cuadro presenta una columna para especificar la acción que se está ejecutando en ese momento, una columna por cada uno de los datos que intervienen en el algoritmo, ya sean datos de entrada, de salida o locales y una columna que refleja el valor de verdad de la condición que es evaluada si la estructura de control es condicional o repetitiva. La primera fila del cuadro contiene el nombre del algoritmo que comienza a ser ejecutado, la segunda fila contiene los nombres de los items que representan las columnas. Las siguientes filas se van completando a medida que se ejecuta cada una de las acciones.

3.1.2 Sobre el Constructor de Trazas completas

Con el propósito de especificar en la traza la acción del algoritmo que se desea verificar, el constructor numera las acciones del algoritmo antes de comenzar con la construcción del cuadro. Las acciones quedan numeradas en orden creciente comenzando desde el 1, considerando el orden textual de la especificación del algoritmo, no el orden de ejecución. Eventualmente estos órdenes pueden coincidir.
El constructor abre en primer lugar una ventana que muestra el algoritmo con las acciones numeradas y a continuación abre una segunda ventana en la cual comienza a desplegar la traza.
Al comenzar a desarrollar el cuadro, el constructor genera automáticamente la primera y segunda fila, constituyendo éstas el encabezamiento del mismo. A continuación se completa la tercer fila con los valores de entrada, para ello el constructor interactúa con el usuario. Cada fila subsiguiente está encabezada por el número de acción que se está ejecutando y se completa con el valor de la condición si es una estructura de control condicional o repetitiva o bien con el nuevo valor que se asigne a un dato en esa acción .
Por ejemplo para el caso del siguiente algoritmo:

```
Algoritmo DCMDos
        Datos de Entrada: m,n
        Datos de Salida: dcm
        Datos Locales: resto
Acciones
        repetir
                resto ← m MOD n
```

```
            m ← n
            n ← resto
        hasta resto = 0
        dcm ← m
Fin.
```

En primer lugar el constructor enumera las acciones y aparece una ventana como:

```
Algoritmo DCMDos
      DE: m,n
      DS: dcm
      DL: resto
Acciones
          repetir
(1)               resto ← m MOD n
(2)               m ← n
(3)               n ← resto
(4)       hasta resto = 0
(5)       dcm ← m
Fin.
```

A continuación se abre automáticamente otra ventana con el encabezamiento del cuadro, y luego se solicitan los datos de entrada

DCMDos					
Acción	condición	m	n	resto	dcm

Por ejemplo, si el usuario entra los valores 26 y 8 para m y n respectivamente, el cuadro se va completando por filas hasta quedar de la siguiente manera:

DCMDos					
Acción	condición	m	n	resto	dcm
		26	8		
1				2	
2		8			
3			2		
4	f				
1				0	
2		2			
3			0		
4	v				
5					2

En cada momento la acción que se está ejecutando se corresponde con la última fila generada en la traza.

Cuando se trabaja con algoritmos que activan otros algoritmos, la traza del algoritmo llamador deberá quedar en suspenso hasta que el algoritmo llamado resuelva su problema y devuelva el control automáticamente al llamador. La traza del algoritmo invocado tiene la misma forma que las trazas descriptas hasta el momento. El constructor crea automáticamente el encabezamiento de la nueva traza, copia los valores correspondientes a los datos de entrada, desarrolla el cuadro y una vez completado éste, copia los valores de los datos de salida en la traza del algoritmo llamador (en las columnas de los datos correspondientes).

Por cada primitiva el constructor abre dos ventanas: una para el algoritmo con las instrucciones numeradas y otra para el desarrollo de la traza. En cada momento por defecto son visibles las

ventanas correspondientes al último algoritmo activado. Sin embargo, el usuario puede acceder a otras ventanas (que no han sido expresamente cerradas).
El Constructor de Trazas cuenta con las siguientes características:

- Mantiene el encabezamiento de la traza siempre visible aún cuando la longitud del cuadro supere el tamaño de la ventana.
- La velocidad con que se desarrolla la traza es adecuada a la capacidad del ser humano para percibir los cambios. Existe posibilidad de ajuste.
- La ejecución puede detenerse en cualquier momento con la posibilidad futura de continuar o bien de abortar.
- Respeta el código cromático establecido en el Editor de Algoritmos.

3.1.3 Traza simplificada

Como el desarrollo de una traza completa suele ser de una extensión considerable y con excesivo nivel de detalle para el alumno que ya ha superado la etapa inicial, resulta de gran utilidad la construcción de trazas simplificadas.
En principio la enumeración de las acciones del algoritmo puede ser reducida. Frente a una sucesión de asignaciones el constructor pone un solo número de acción, como así también para las acciones condicionales y repetitivas. Con lo cual cada línea del cuadro que representa la traza revela la modificación de los estados de los datos producida por acciones sucesivas, no necesariamente de una sola acción como se realiza en las trazas completas.
Este tipo de traza tiene la ventaja de mostrar los cambios de la computación en menor cantidad de filas, por lo que resulta mejor para la visualización en una ventana.
Por ejemplo para el algoritmo DCMDos y los datos de entrada 26 y 8, las ventanas muestran:

```
Algoritmo DCMDos
        DE: m,n
        DS: dcm
Acciones
        repetir
(1)             resto ← m // n
                m ← n
                n ← resto
(2)     hasta resto = 0
(3)     dcm ← m
Fin.
```

DCMDos					
Accion	condición	m	n	resto	dcm
		26	8		
1		8	2	2	
2	f				
1		2	0	0	
2	v				
3					2

3.1.4 Sobre el Constructor de Trazas simplificadas

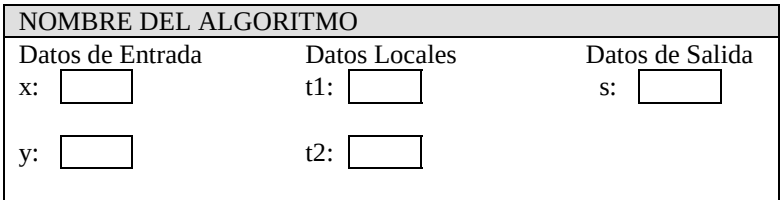
El diseño es análogo al del Constructor de Trazas completas. Difiere sólo en el cuadro que genera; lo realiza siguiendo fielmente la descripción del punto anterior.

3.2 El Constructor de Esquemas de Ejecución de Algoritmos

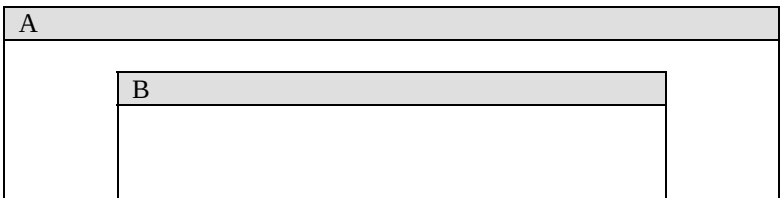
Con la construcción de estos esquemas se pretende captar en forma clara y precisa la relación entre el algoritmo como entidad estática y su ejecución como proceso dinámico. Están basados en los diagramas de ejecución para Pascal presentados en [Kow82].

3.2.1 Los Esquemas de Ejecución de Algoritmos

Se define Esquema de Ejecución como la representación gráfica que permite simular la ejecución de un algoritmo o de un programa en forma sistemática. Cuando se trata de ejecución de algoritmos nos referimos a EEA y en el otro caso a Esquemas de Ejecución para Pascal (EEP), teniendo en cuenta el lenguaje de implementación usado.
Un EEA está compuesto por rectángulos anidados. Cada rectángulo corresponde a la activación de un algoritmo, y contiene su nombre en la parte superior y la siguiente distribución para los datos de entrada, datos de salida y datos locales.



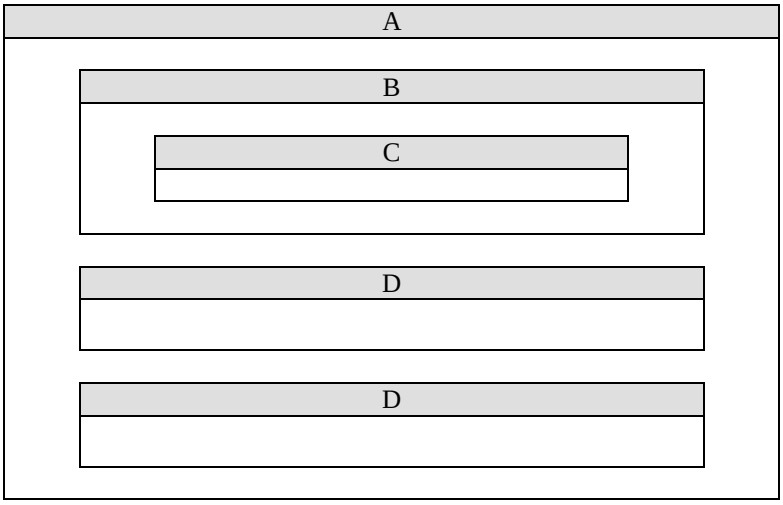
El rectángulo se cierra cuando termina la ejecución de la activación del algoritmo.
Durante el proceso de ejecución cada dato muestra el valor que tiene asociado en ese instante.
Si un algoritmo A invoca un algoritmo B, entonces en el EEA el rectángulo con encabezamiento B queda encajado en el correspondiente a A.



Dentro de un determinado rectángulo, los rectángulos encajados se ubican siguiendo el orden cronológico de las llamadas. Todas las invocaciones que se realizan desde un determinado algoritmo generan rectángulos que se disponen en el mismo nivel de anidamiento.
Por ejemplo para la ejecución del algoritmo A en el siguiente caso:

```
Algoritmo A          Algoritmo B          Algoritmo D          Algoritmo C
    ……
    B                    ……                   ……                   ……
    ……                   C
    D                    ……                   Fin                  Fin
    ……
    D                    Fin
    ……
Fin
```

Se observa que B hace una única invocación al algoritmo C y D no contiene invocaciones a otros algoritmos, al finalizar la ejecución de A se tendrá:

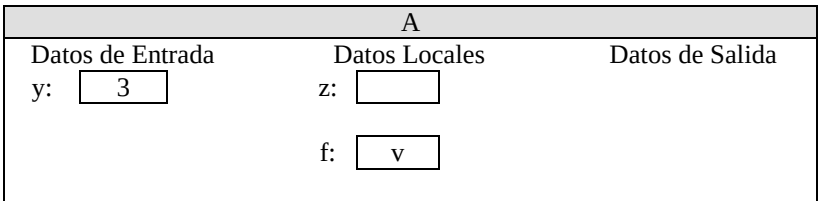


Los EEA muestran fielmente la semántica de la invocación de un algoritmo, en particular lo referente a los datos de entrada y a los datos de salida.
Supongamos que el algoritmo A invoca al algoritmo B de la siguiente manera: B (10,y,z). Se observa que B tiene dos datos de entrada y uno de salida, y que el valor del dato de salida de B es asignado a z en A cuando finaliza la ejecución de B.
Sean los encabezamientos de los algoritmos A y B como sigue:

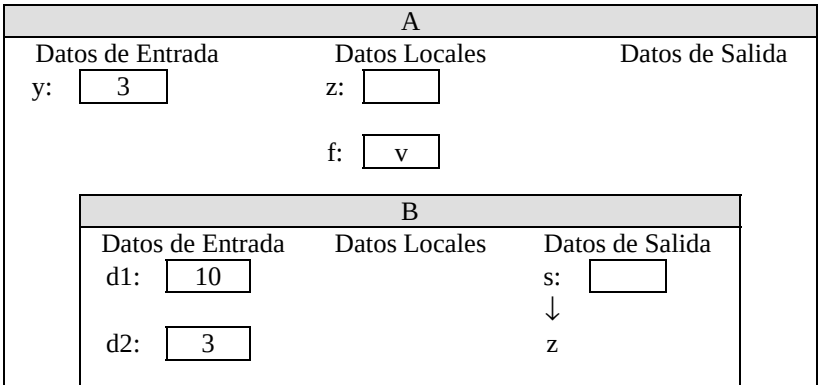
Algoritmo A
Datos de Entrada: y ∈ Enteros
Datos Locales: z ∈ Enteros
f ∈ Booleano

Algoritmo B
Datos de Entrada: d1, d2 ∈ Enteros
Datos de Salida: s ∈ Enteros

Si el rectángulo asociado con A en el instante anterior a la invocación de B es:

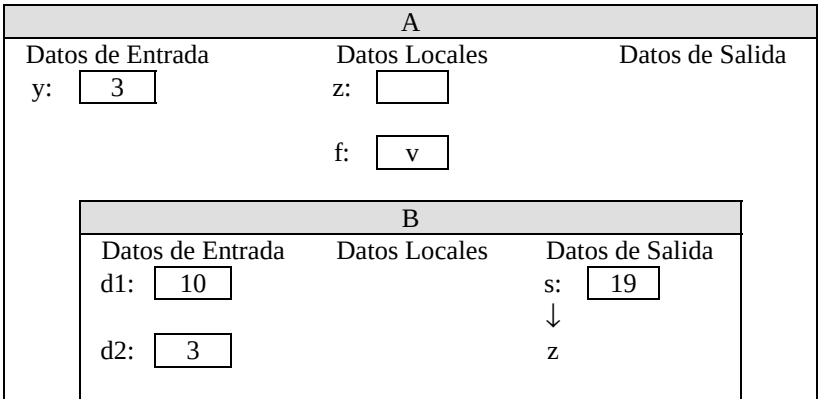


Al realizarse la llamada a B quedará:

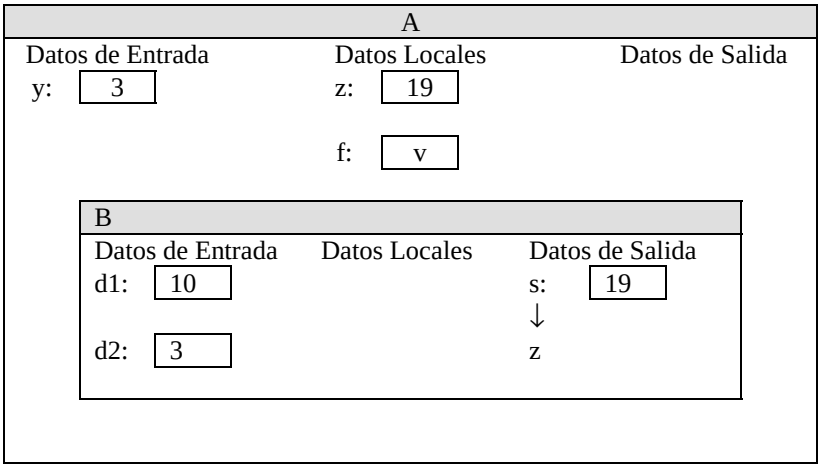


Puede observarse que los datos de entrada de B (d1 y d2) toman los valores 10 y 3 respectivamente y que al dato de salida s se le asocia el nombre z que indica el dato que recibirá el valor final que tomará s cuando se termine con la ejecución de la activación B. El dato que

recibe el valor está unívocamente determinado, ya que cada algoritmo sólo maneja datos de su propio ámbito, por lo tanto z pertenece al rectángulo que anida a B.
Si al terminar la ejecución correspondiente a la activación B se tiene:



Al cerrarse el rectángulo B, el dato en A tomará el valor 19, y queda entonces:



3.2.2 Sobre el Constructor de Esquemas de Ejecución de Algoritmos

El Constructor de EEA se comporta de manera análoga al Constructor de Trazas. Muestra el algoritmo en una ventana y despliega el EEA en otra, siguiendo fielmente la descripción presentada en el punto anterior e interactuando con el usuario en los siguientes casos:

- entrada de datos
- interrupción temporaria o definitiva de la construcción del EEA

4. Nuevo Entorno de Programación

El Nuevo Entorno de Programación queda entonces compuesto por:

- un *Editor interactivo para MCH*
- un *Editor interactivo de algoritmos*
- un *Constructor de trazas*
- un *Constructor de EEA*
- un *Traductor automático de algoritmos a programas*
- un *Editor de textos* que permite ver y modificar los programas fuente generados automáticamente por el Traductor

Trabajar con el nuevo entorno le permitirá al alumno usar el Editor para crear el MCH del tema sobre el que trata el problema a resolver, lo que representa una ayuda importante para la comprensión significativa del mismo. A continuación podrá usar el Editor interactivo de algoritmos para construir aquellos que considere necesarios para la resolución del problema. Antes de pensar en la codificación, podrá usar el módulo de verificación, optando entre el Constructor de trazas o el Constructor de EEA. A partir de allí tiene la posibilidad de optar entre el Traductor automático a lenguaje Pascal, o de usar un entorno específico para escribir programas en Pascal, según la metodología para el aprendizaje del lenguaje de codificación que se esté usando. Para analizar y eventualmente modificar el programa fuente generado por el Traductor tiene a disposición el Editor correspondiente.

5. Conclusiones

Resulta novedosa la inclusión de herramientas para la representación de las ideas en un entorno para el aprendizaje de la programación y el diseño de una plataforma computacional que abarque todas las etapas del proceso de desarrollo de un programa.
La existencia de herramientas específicas para las etapas de comprensión del problema y de verificación de algoritmo dentro del nuevo Entorno de Programación, ayudará a los alumnos a tomar conciencia de la importancia del cumplimiento de las mismas y a superar las dificultades que muchos de ellos presentan actualmente.
Así, este nuevo Entorno ofrece sin dudas mayores posibilidades para lograr con éxito el aprendizaje de la programación
Actualmente se están implementando los Constructores de trazas y de EEA respetando la propuesta teórica acá presentada y ya se ha insertado el Editor de MCH en el entorno existente.
Una extensión interesante del Entorno de Programación propuesto consiste en la inclusión de un Constructor de EEP. Puede resultar de gran utilidad para la comprensión de distintos aspectos semánticos del lenguaje de programación usado para la implementación.

6. Bibliografía

[Cha91] Darah Chavey- Beloit College. *A Structured Laboratory Component for the Introductory Programming Course*. SIGCSE BULLETIN ACM. 1991.

[For96] R. Forcier. *The computer as a productivity tool in education*. Merrill-Prentice Hall. 1996

[Gar96] García, Juan - Señas, Perla - Moroni, Norma. *Cubik: Una herramienta de apoyo en la enseñanza de la programación*. IV Ateneo de Profesores Universitarios de Computación. 1996.

[Gei94] Robert Geitz. *Concepts in the classroom, programming in the lab*. ACM SIGCSE BULLETIN. 1994.

[Kow82] *Implementación de Lenguajes de Programación*. 1988

[Lev94] Lisa Levy Kortright. *From Specific Problem Instances to Algorithms in the Introductory Course*. . SIGCSE BULLETIN ACM. 1994.

[Mor96a] Moroni, N. - Vitturini, M. - Zanconi, M. - Señas, P. *Una plataforma para el desarrollo de mapas conceptuales hipermediales*. Taller de Software Educativo - IV Jornadas Chilenas de Computación. Valdivia. 1996.

[Mor96b] Moroni, Norma y Señas, Perla. *Un entorno para el aprendizaje de la programación*.IV Ateneo de Profesores Universitarios de Computación. 1996.

[Pyo91] Pyott, S. and Sanders,I. *ALEX: an aid too teaching algorithms*. ACM SIGCSE BULLETIN. 1991.

[Rue95] Rueda, S. ; Castro, S y Zanconi, M. *Resolución de Problemas y Algoritmos*: notas de curso. 1995.

[Sen96a] Señas,P. , Moroni,N., Vitturini,M. y Zanconi,M. *Hypermedial Conceptual Mapping: A Development Methodology*. 13th International Conference on Technology and Education. University of Texas at Arlington, Departament of Computer Science an Engineering. New Orleans 1996.

[Señ96b] Señas, P. , Moroni, N., Vitturini, M. y Zanconi, M. *Combining Conceptual Mappings and Hypermedia*. ED-MEDIA96. Boston. 1996.

[Ter95] Terry, P. *Umbriel - Imperative Programming for unsophisticated students*. ACM SIGCSE BULLETIN. 1995.

[Thu95] Thuring, Hannemann and Haake. *Hypermedia and Cognition: Designing for Comprehensión*. Communications ACM. August 1995.