CONSTRUCCIÓN DE UN DEPURADOR PORTABLE DE CÓDIGO. EVALUADOR DE EXPRESIONES

José L. Arjona Fernández, José M. Prieto Pérez, R. Corchuelo Gil
Dpto. de Lenguajes y Sistemas Informáticos
Facultad de Informática y Estadítica
Universidad de Sevilla
Web: http://www.lsi.us.es

En este tercer artículo veremos la forma en la que podrá interactuar el usuario con el depurador, los comandos que podrá usar; para despues centrarnos en la construcción del evaluador de expresiones.

Comandos. Interacción con el usuario.

Una parte primordial en el desarrollo del depurador ha sido la interacción que debe llevar a cabo en todo momento con el usuario. Pensemos que la función de un depurador de código es la detección de errores en un programa. Pero esta detección se debe de basar en una colaboración intensa entre la persona que se encargue de ello y el propio depurador.

Para poder conseguir esta colaboración es esencial que el depurador cuente con una interfaz de usuario a través de la cual reciba todo tipo de órdenes por parte del mismo. Estas órdenes serán ejecutadas por un obediente depurador. La interfaz de usuario en nuestro caso se basa en una línea de comandos, muy similar a la que ofrecen los sistemas operativos interactivos tales como MS-DOS o UNIX. Por lo tanto, la interacción con el depurador se lleva a cabo a través de un prompt en pantalla preparado para recibir las órdenes.

Pensando siempre en la posibilidad existente de que el programa que estemos depurando conste de varios módulos contenidos en sus correpondientes ficheros de texto, el prompt está pensado para que en todo momento indique al usuario donde se encuentra dentro del programa, y esto simplemente se consigue indicando el nombre del fichero por donde va la traza de la ejecución y la línea del mismo donde se encuentra esa traza. En concreto, este prompt tiene el siguiente aspecto:

```
(fichero:línea)? _
```

Como ya se ha dicho con anterioridad, el depurador ofrece una serie de órdenes o comandos, cada uno de los cuales tiene una sintáxis y una semántica definida. La oferta de órdenes se ha intentado que fuera lo más potente posible para dar la posibilidad al usuario de realizar una traza del programa donde se le escaparan los menos detalles posibles. Un resumen de las órdenes que se pretenden implementar se relacionan en la tabla 1:

ORDEN	DESCRIPCIÓN
Órdenes de ayuda y detalles	
help	Ayuda en línea del depurador. Ofrece un resumen en pantalla de todos los comandos posibles.
line	Muestra en pantalla la línea del programa fuente por donde va la traza de ejecución.
source	Muestra en pantalla el contenido de un fichero. Con esta orden se puede echar un vistazo a los ficheros que constituyen el programa fuente.
Órdenes de ejecución	
next	Se ejecuta la línea de código correspondiente a la situación actual de la traza y se pasa a la siguiente instrucción.
step	Idéntica a la orden *next* con la diferencia esencial de que se salta la ejecución de las rutinas.
run	Se ejecuta el programa desde donde se está hasta que se llegue al final de la ejecución o bien se evalúe un punto de ruptura a "cierto".
animate	Idéntica a la orden *run*, solo que se puede especificar un retraso en milisegundos entre cada instrucción del programa que se ejecuta.
Órdenes de evaluación del valor y tipo de una expresión	
watch	Permite evaluar una determinada expresión, devolviendo su valor y su tipo.
whatis	Permite tanto mostrar qué es cierto objeto definido en el programa como evaluar el tipo de un determinada expresión.
Órdenes de asignación de valores	
set	Permite asignar a un determinado objeto del programa un valor.
Órdenes de manejo de puntos de ruptura	
break	Permite colocar un punto de ruptura en algún punto del programa.
unbreak	Permite eliminar un punto de ruptura previamente colocado.
vbreak	Permite visualizar los puntos de ruptura colocados.
Órdenes de salida del depurador	
exit	Salida inmediata de la ejecución del programa.

TABLA 1. RESUMEN DE ÓRDENES DEL DEPURADOR.

Expresiones del depurador.

Una vez que se ha dado una idea de como interactúa el depurador con el usuario y de las órdenes que podrá ejecutar, tenemos que pasar a detallar aspectos más concretos relativos a la línea de comandos del mismo. Por lo tanto, es obligatorio dedicar a una sección de este artículo a las expresiones que permite el depurador.

Por *expresión* podemos entender a un conjunto de *operandos* que se relacionan entre sí mediante unos *operadores*. Como consecuencia de tal combinación, de la evaluación de una expresión siempre se genera un valor concreto. Por ejemplo, una expresión muy simple pudiera ser 1+1, donde se tiene un solo operador (la suma, +) y dos operandos (los números 1). Como se podrá imaginar, existen expresiones muchos más complejas que este ejemplo. De esta expresión se generará un valor, en este caso numérico (2).

Ya que podemos mezclar operadores diferentes en una misma expresión, se necesita definir siempre lo que se conoce como *precedencia* y *asociatidad* de los mismos. Con precedencia se hace referencia a la prioridad que se asigna a cada operador, es decir, qué o cual operador se evalúa antes que otro. La asociatividad de los operadores hace referencia al orden de evaluación de varios operadores iguales.

Otro aspecto importante de las expresiones es que cada una de ellas tiene asociado un tipo, es decir, una determinada expresión puede ser de tipo entera, real, etc. Debido a lo anterior, no se puede aplicar normalmente cualquier operador sobre cualquier tipo de operando. De esto que exista lo que se llama *compatibilidad de tipos* en las expresiones por la cual se especifica qué operadores

se pueden aplicar a qué tipos de operandos y cúal es el tipo del resultado de realizar dicha operación sobre los operandos.

Pues bien, el depurador define sus propios operadores, niveles de precedencia, asociatividad de los mismos, así como su sistema de compatibilidad de los tipos existentes.

Una posible gramática abstracta para el evaluador de expresiones del depurador sería:

```
Binaria :: ti,td:Expresion; op:OperadorBin /*  OPERACIONES BINARIAS */

Unaria :: t:Expresion; op:OperadorUn       /* OPERACIONES UNARIAS */

      /* OPERADORES BINARIOS */
OperadorBin :: OperadorBArit     /* Aritméticos */
             | OperadorBLog      /* Lógicos */
             | OperadorBRel      /* Relacionales */
             | OperadorBAcceso   /* Acceso a Registros, Uniones y Arrays */
             | OperadorBAsig     /*  Para  el comando SET -> Asignación*/

      /* OPERADORES UNARIOS */
OperadorUn :: MasU            /* (+) unario */
            | MenosU          /* (-) unario */
            | Indireccion     /* (*) Para poder acceder al contenido de
                                 punteros, dir de memoria */
            | Direccion       /* (&) Dirección del objeto */
            | Not             /* (!) inversor lógico */

OperadorBArit :: Suma         /* (+) Suma */
               | Resta        /* (-) Resta */
               | Multip       /* (*) Multiplicacion */
               | Division     /* (/) División */

OperadorBRel :: Mayor         /* (>) Mayor */
              | Menor         /* (<) Menor */
              | MayorIg       /* (>=) Mayor o igual */
              | MenorIg       /* (<=) Menor o igual */
              | Distinto      /* (!=) Distinto */
              | Ig            /* (==) Igual */

OperadorBLog :: And     /* (&&) And lógico */
              | Or      /* (||) Or lógico */

OperadorBAcceso :: Aarray      /* ([]) Acceso a un elemento de un array */
                 | Aregistro   /* (.) Acceso a un campo de un registro o
                                  unión*/
                 | Aflecha     /* (->) Acceso indirecto a un campo del
                                  registro o unión*/
```

Precedencia y asociatividad de los operadores.

La tabla 2 muestra todos los operadores del depurador. Están ordenados desde la precedencia más alta, es decir, los más prioritarios, hacia la precedencia más baja. Además, aquellos operandos que se encuentran entre líneas tienen el mismo nivel de precedencia. Por otra parte, en dicha tabla también se indica la asociatividad de los mismos, indicando con "izquierda" aquellos cuya asociatividad es por la izquierda, con "derecha" aquellos que la tienen por la derecha y con "no asociativo" aquellos a los que no se le aplica asociatividad.

OPERADOR	DESCRIPCIÓN	ASOCIATIVIDAD	PRECEDENCIA
.	Acceso a registro	Izquierda	más alta
[]	Acceso a elemento de un array	Izquierda	
->	Acceso indirecto a registro	Izquierda	
!	Negación lógica	Derecha	
+ (unario)	Más unario	Derecha	
- (unario)	Menos unario	Derecha	
* (desreferencia)	Contenido de un puntero	Derecha	
&	Dirección	Derecha	
*	Producto	Izquierda	
/	División	Izquierda	
+	Suma	Izquierda	
-	Resta	Izquierda	
>	Mayor que	no asociativo	
<	Menor que	no asociativo	
>=	Mayor o igual que	no asociativo	
<=	Menor o igual que	no asociativo	
==	Igual a	no asociativo	
!=	Distinto de	no asociativo	
&&	And lógico	Izquierda	
||	Or lógico	Izquierda	más baja

TABLA 2. PRECEDENCIA Y ASOCIATIVIDAD DE LOS OPERADORES.

Compatibilidad de tipos.

Veamos ahora la compatibilidad de tipos para el depurador, la compatibilidad de tipos impone restricciones sobre la utilización de los operadores. Por ello es importante que quede absolutamente claro qué clase de valores podemos combinar con los distintos operadores que se ofrecen si se pretende que la utilización de las expresiones en el depurador sea correcta.

Puesto que básicamente los operadores existentes se pueden clasificar en *operadores binarios* (aquellos que necesitan dos operandos para poder aplicarse) y *operadores unarios* (aquellos que necesitan tan solo un operando para poder aplicarse), nos basaremos en esta clasificación para mostrar la compatibilidad de tipos respecto a estos operadores. Es decir, dado un operador cualquiera, ¿qué tipos podemos combinar con él?

La tabla 3 muestra la compatibilidad de tipos respecto a los operadores unarios. Estos, en nuestro caso son cinco, a saber: el más y el menos unario (+,-), la indirección o desreferencia (*), el operador de dirección (&) y el inversor lógico o negación (!). En dicha tabla, las columnas representan a estos operadores y las filas a los distintos tipos. Así por ejemplo, que exista un "sí" en

la casilla correspondiente a la fila “Entero” y a la columna “+/- (Unario)” significa que dicho tipo sí es compatible respecto a estos operadores unarios y, por lo tanto, una expresión tal como +6 ó -5 serían correctas.

	+/- (Unario)	! (Not)	* (Indirección)	& (Dirección)
Entero	sí	no	no	Sí
Byte	sí	no	no	Sí
Real	sí	no	no	Sí
Booleano	no	sí	no	Sí
Carácter	no	no	no	Sí
Cadena	no	no	no	Sí
Enumeración	sí	no	no	Sí
Vacío	no	no	no	No
Array	no	no	no	Sí
Registro	no	no	no	Sí
Unión	no	no	no	Sí
Puntero	no	no	sí	Sí

TABLA 3. COMPATIBILIDAD DE TIPO RESPECTO A LOS OPERADORES UNARIOS.

En la tabla 3 podemos observar varios puntos que pueden resultar confusos:

- El tipo *Vacío* es algo especial pues a él no le podemos aplicar ninguno de los operadores anteriores. Esto es debido a que nunca nos vamos a encontrar con el caso de que existan variables de este tipo. Por ejemplo, no tiene sentido que un compilador dé como válido una declaración tal como *a:Vacío;* pues ¿qué representa la variable “a”? ¿qué tamaño ocupa en memoria dicha variable? Teóricamente el tipo Vacío representa el “nada”, ¿tiene algún sentido operar algo con la “nada”? Evidentemente, NO. Sin embargo, como en C, sí tiene sentido declarar variables de tipo *Puntero a Vacío* para representar objetos que contienen una dirección de memoria que apunta a “yo no sé que tipo”, es decir, para representar una dirección de memoria genérica.

- El operador de dirección se puede aplicar sobre cualquier tipo, exceptuando al tipo *Vacío*. Esto es así pues se puede tener en memoria objetos de cualquiera de los tipos con los que se cuenta y por lo tanto podremos saber su dirección con este operando.

Por otro lado, los operadores binarios se pueden subdividir a su vez en los siguientes grupos:

1. Operadores aritméticos: + (suma), - (resta), * (producto), / (división).
2. Operadores relacionales: > (mayor), < (menor), >=(mayor o igual), <= (menor o igual), ==(igual), != (distinto).
3. Operadores lógicos: && (and), || (or).
4. Operadores de acceso: . (a registro), [] (a array), -> (indirecto a registro).
5. Operador de asignación: =.

La tabla 4 muestra la compatibilidad de los tipos respecto a los operadores aritméticos, relacionales y lógicos. Los operadores de acceso y de asignación son un poco más especiales que estos. El encabezado horizontal de dicha tabla hace referencia al tipo del operando a la derecha del operador correspondiente, mientras que el encabezado vertical se refiere al tipo del operando a la izquierda de ese operador.

Término Izquierda	Término Derecha											
	Entero	**Byte**	**Real**	**Bool.**	**Carac.**	**Cadena**	**Enum.**	**Vacío**	**Array**	**Reg.**	**Union**	**Punt.**
Entero	A,R	A,R	A,R				A,R					A(+,-)
Byte	A,R	A,R	A,R				A,R					R A(+,-)
Real	A,R	A,R	A,R				A,R					
Bool			L R(==)									
Carac.					R							
Cadena						R						
Enum.	A,R	A,R	A,R				A,R					R A(+,-)
Vacío												
Array												
Reg.												
Union												
Punt.	R A(+,-.)	R A(+,-)					R A(+,-)					R(1)

(1) Independientemente del tipo al que apunten.
A: Operadores Aritméticos [entre paréntesis aquel subconjunto de éste a los que se aplica].
R: Operadores Relacionales [entre paréntesis aquel subconjunto de éste a los que se aplica].
L: Operadores Lógicos.
casilla en blanco: los tipos no son compatibles respecto a ningún operador.

TABLA 4. COMPATIBILIDAD DE TIPOS EN OPERADORES BINARIOS ARITMÉTICOS, RELACIONALES Y LÓGICOS.

Por último, la tabla 5 muestra la compatibilidad de tipos respecto al operador de asignación. La lectura que se hace de la misma es idéntica a la tabla 4, excepto que en este caso solo se indica si es compatible, o no lo es, asignar al tipo de la expresión de la izquierda del signo de asignación (=) el tipo de la derecha del mismo.

Término Izquierda	Término Derecha											
	Entero	**Byte**	**Real**	**Bool.**	**Carac.**	**Cadena**	**Enum.**	**Vacío**	**Array**	**Reg.**	**Union**	**Punt.**
Entero	sí	sí	No	no	no	no	Sí	no	no	no	no	no
Byte	no	sí	No	no	no	no	No	no	no	no	no	no
Real	sí	sí	Sí	no	no	no	Sí	no	no	no	no	no
Bool	no	no	No	sí	no	no	No	no	no	no	no	no
Carac.	no	no	No	no	sí	no	No	no	no	no	no	no
Cadena	no	no	No	no	no	sí	No	no	no	no	no	no
Enum.	sí	sí	No	no	no	No	Sí	no	no	no	no	no
Vacío	no	no	No	no	no	No	No	no	no	no	no	no
Array	no	no	No	no	no	No	No	no	sí(1)	no	no	no
Reg.	no	no	No	no	no	No	no	no	no	sí(2)	no	no
Union	no	no	No	no	no	No	no	no	no	no	sí(3)	no
Punt.	sí	sí	No	no	No	No	sí	no	no	no	no	sí(4)

(1): ambos arrays deben ser estructuralmente iguales.
(2): ambos registros deben ser estructuralmente iguales.
(3): ambas uniones deben ser estructuralmente iguales.
(4): ambos punteros deben apuntar a objetos tipos “asignables”, o bien alguno de ellos ser un “Puntero a Vacío”.

TABLA 5. COMPATIBILIDAD DE TIPOS EN LA ASIGNACIÓN.

Tipo Resultado de la evaluación de una expresión.

La tabla 6 muestra la totalidad de combinaciones aritméticas posibles que podemos hacer con dichos cuatro tipos numéricos (tanto a nivel de operadores binarios como unarios) y a su vez el tipo que se induce de evaluar la expresión aritmética con la combinación correspondiente.

Operadores Binarios (+, -, *, /)					
Término Izquierdo	**Término Derecho**				**Operadores Unarios (+,-)**
	Entero	**Byte**	**Enumeración**	**Real**	
Entero	Entero	Entero	Entero	Real	Entero
Byte	Entero	Byte	Entero	Real	Byte
Enumeración	Entero	Entero	Entero	Real	Entero
Real	Real	Real	Real	Real	Real

TABLA 5. TIPOS EVALUADOS EN EXPRESIONES NUMÉRICAS.

Con respecto a los operadores lógicos y relacionales, decir que el tipo del resultado de la evaluación será siempre lógico, es decir, o CIERTO o FALSO.

El resultado de los operadores de acceso a uniones y registros será el tipo definido para el campo al que estamos accediendo. Igualmente al acceder a un array el tipo resultado será el tipo base del array (el tipo de los elementos almacenados en el array).

Al aplicar el operador de contenido (*) sobre un puntero, obtendremos como tipo resultado el tipo del objeto apuntado por ese puntero. Igualmente, el operador dirección (&) sobre un objeto de *tipo a* cualquiera dará como tipo resultado un puntero a *tipo a.*

El evaluador de expresiones.

Una vez definidas la precedencia, asociatividad y la compatibilidad de tipos para las expresiones que permitirá el depurador, la forma de proceder será la siguiente:

1. Se analizará léxicamente y sintácticamente la expresión.
2. Se analizará semánticamente, para ello a partir del árbol de sintaxis abstracta de la expresión, será necesario hacer un chequeo de tipos que nos permita saber si la expresión es semánticamente correcta (los operadores pueden ser aplicados a los operandos). También sería necesario comprobar si el usuario introduce correctamente los índices de los arrays o si accede a campos existentes de registros...
3. Una vez determinada que la expresión es correcta, sólo quedaría dar el resultado de su evaluación.

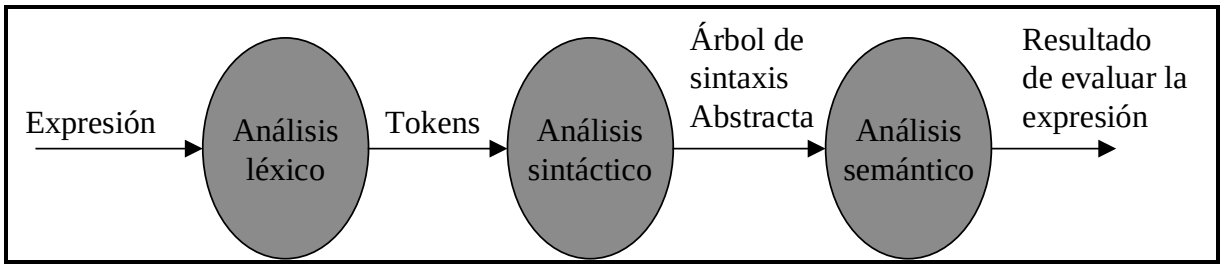

Figura 1. Proceso de evaluación de expresiones.