PRÁCTICA DE
PROCESADORES DEL LENGUAJE II

Suárez Pérez, Misrraim

misrraimsp@gmail.com

1. El analizador semántico y la comprobación de tipos

1.1 Descripción del manejo de la Tabla de Símbolos y de la Tabla de Tipos

A continuación se listan los 62 errores semánticos que el compilador captura. Posteriormente se hará los comentarios oportunos acerca de los mismos (los errores en amarillo corresponden a la verificación de tipos):

Ámbito sintáctico	Nombre del error	Descripción
decl constantes	ERROR_const1	identificador de constante literal ya declarado
decl tipos	ERROR_reg1	identificador de tipo usado por otro tipo
	ERROR_reg2	identificador de tipo usado por símbolo en ámbito actual
	ERROR_reg3	identificador de campo repetido en mismo registro
decl variables	ERROR_var1	identificador de variable usado por símbolo en ámbito actual
	ERROR_var2	identificador de variable usado por tipo en ámbito actual
	ERROR_tipoNoDecl	tipo registro no declarado
decl subprogramas	ERROR_sub1	identificador de subprograma usado por símbolo en mismo ámbito
	ERROR_sub2	identificador de subprograma usado por tipo en mismo ámbito
	ERROR_sub3	identificador de parámetro ya declarado
asignación	ERROR_sentAsig1	identificador no declarado
	ERROR_sentAsig2	identificador no corresponde a variable o parámetro
	ERROR_sentAsig3	tipos de variable/parámetro y expresión incompatibles
	ERROR_sentAsig4	registro no declarado
	ERROR_sentAsig5	identificador no corresponde a tipo registro
	ERROR_sentAsig6	registro no contiene campo especificado
	ERROR_sentAsig7	tipos de campo y expresión incompatibles
while	ERROR_sentWhile1	sentencia WHILE espera expresión de control booleana
if	ERROR_sentIf1	sentencia IF espera expresión de control booleana
writestring	ERROR_sentWRITEINT1	WRITEINT espera expresión entera
call procedimiento	ERROR_sentCall1	identificador de procedimiento no declarado
	ERROR_sentCall2	identificador no corresponde a procedimiento
	ERROR_sentCall3	número de parámetros formales y actuales no coincide
	ERROR_sentCall4	parámetros formales y actuales no coinciden
	ERROR_sentCall5	llamada a procedimiento de ámbito superior
call parámetros	ERROR_callParam1	identificador de argumento no declarado
	ERROR_callParam2	identificador de argumento no corresponde a variable (paso de parámetros por referencia)
	ERROR_callParam3	identificador de variable no corresponde a tipo registro
	ERROR_callParam4	variable tipo registro no contiene campo especificado
not	ERROR_exNot1	expresión NOT espera expresión booleana como argumento
div	ERROR_exDiv1	expresión DIVISION espera expresión entera en numerador

	ERROR_exDiv2	expresión DIVISION espera expresión entera en denominador
sum	ERROR_exSum1	expresión SUMA espera expresión entera como primer sumando
	ERROR_exSum2	expresión SUMA espera expresión entera como segundo sumando
and	ERROR_exAnd1	expresión AND espera expresión booleana como primer operando
	ERROR_exAnd2	expresión AND espera expresión booleana como segundo operando
menor	ERROR_exMenor1	expresión MENOR espera expresión entera como primer operando
	ERROR_exMenor2	expresión MENOR espera expresión entera como segundo operando
distinto	ERROR_exDist1	expresión DISTINTO espera expresiones del mismo tipo simple como operandos
id as expr	ERROR_exId1	identificador no declarado, no puede ser expresión
	ERROR_exId2	identificador de procedimiento, no puede ser expresión
	ERROR_exId3	identificador de función que necesita parámetros
	ERROR_exId4	identificador de variable tipo registro sin especificar campo
	ERROR_exId5	identificador de parámetro tipo registro sin especificar campo
	ERROR_exId6	llamada a función de ámbito superior
id.id as expr	ERROR_exReg1	registro no declarado
	ERROR_exReg2	identificador no corresponde a tipo registro
	ERROR_exReg3	registro no contiene campo especificado
call función	ERROR_exFun1	identificador de función no declarado
	ERROR_exFun2	identificador no corresponde a función
	ERROR_exFun3	número de parámetros formales y actuales no coincide
	ERROR_exFun4	parámetros formales y actuales no coinciden
	ERROR_exFun5	llamada a función de ámbito superior
nombre ámbito	ERROR_modName1	identificadores de inicio y fin de ámbito no coinciden
return	ERROR_ret1	función sin RETURN
	ERROR_ret2	procedimiento con RETURN
	ERROR_ret3	RETURN en ámbito global
	ERROR_ret4	función no devuelve tipo declarado
	ERROR_ret5	sentencia IF-THEN-ELSE con RETURN sólo en rama THEN
	ERROR_ret6	sentencia IF-THEN-ELSE con RETURN sólo en rama ELSE
	ERROR_ret7	RETURN con diferente tipo de retorno en ramas THEN y ELSE
	ERROR_ret8	varios RETURN en mismo ámbito con diferente tipo de retorno

Los anteriores errores se han identificado siguiendo una política *syntax-directed*, esto es, identificando como errores diferentes aquellos que, aún poseyendo una semántica idéntica, se dan en construcciones sintácticas diferentes. Por ejemplo, los errores ERROR_sentAsig6 y ERROR_exReg3 se consideran errores diferentes ya que tienen lugar en construcciones sintácticas diferentes, a pesar de ambos reflejar el hecho de un registro no contener un determinado campo al que se pretende hacer referencia.

Como única excepción a la anterior política están los errores asociados a la sentencia RETURN, que por su entidad se ha preferido agruparlos sin tener en cuenta el ámbito sintáctico. Así, en relación con el control semántico de la sentencia RETURN se destaca lo siguiente:

- Sólo se permite emplear sentencias RETURN dentro del ámbito de las funciones (obligatoriamente en este caso: ERROR_ret1). En procedimientos (ERROR_ret2) y en el ámbito global (ERROR_ret3) no están permitidas.

- El tipo de retorno declarado en las funciones debe coincidir con el tipo de su/s sentencia/s RETURN (ERROR_ret4).

- En las sentencias IF-THEN-ELSE la sentencia RETURN es válida si la poseen ambas ramas (ERROR_ret5 y ERROR_ret6), y las dos con el mismo tipo de retorno (ERROR_ret7).

- Se permite varias sentencias RETURN dentro de un mismo ámbito, pero todas deben ser del mimo tipo de retorno (ERROR_ret8).

- Existe una precedencia implícita en la detección de los errores. Los errores semánticos ERROR_ret5, ERROR_ret6 y ERROR_ret7 (errores asociados a RETURN dentro de sentencias IF) tienen precedencia sobre ERROR_ret8 (varios RETURN con diferentes tipos de retorno en un mismo ámbito), que a su vez tiene precedencia sobre los errores ERROR_ret2, ERROR_ret3 y ERROR_ret4 (RETURN fuera del ámbito de una función o con tipo de retorno diferente al definido en la función).

Por otro lado, destacar que se ha implementado la regla de que dentro de un mismo ámbito no se puedan repetir los identificadores, independientemente de la naturaleza del símbolo de que se trate (variable, subprograma, tipo, etc.). Así, El *id* de los subprogramas (también de las variables, de los tipos, de las constantes, etc.) es verificado contra tener el mismo nombre que cualquier otra variable, constante, etc. dentro del mismo *scope* (se comprueba que el nombre no esté en la tabla de símbolos del *scope* donde se declara el subprograma) y contra tener el mismo nombre que algún tipo declarado en el mismo *scope* (se comprueba que no haya en la tabla de tipos del *scope* donde se declara el subprograma ningún tipo con el mismo nombre).

Además, también se sigue la norma de que no se puede repetir el nombre de los tipos compuestos para todos los ámbitos. Es decir, los identificadores de tipo deben ser únicos para todos los ámbitos.

En relación con el operador DISTINTO cabe destacar que se ha permitido su uso con los operandos booleanos.

Se ha implementado el soporte a la recursividad directa. En cambio, para prevenir la recursividad indirecta (como se establece en el enunciado) se ha decidido restringir el conjunto de subprogramas que pueden ser invocados desde un ámbito concreto mediante dos reglas:

- Sólo se permite que un subprograma invoque a otro si éste último está definido en el mismo ámbito que el primero (*siblings* en el árbol de declaraciones de subprogramas, ó incluso el mismo subprograma (recursividad directa)), ó si está definido dentro del primero (*child*). Dicho de otra forma, no se permite que un subprograma llame a otro definido en un ámbito superior. Los errores ERROR_sentCall5, ERROR_exId6 y ERROR_exFun5 capturan esta situación desde los ámbitos sintácticos correspondientes.

- Un subprograma sólo puede llamar a otro, definido en el mismo ámbito, si éste último ha sido declarado primero.

Por último en relación con el análisis semántico, subrayar que para cada uno de los errores de la tabla anterior se ha desarrollado uno o más casos de prueba, los cuales están en la carpeta correspondiente a los tests del marco de trabajo.

2. Generación de código intermedio

2.1. Descripción de la estructura utilizada

La traducción a código intermedio implementada abarca la totalidad de las funcionalidades del lenguaje, tanto la parte obligatoria como la opcional.

A continuación se listan las 27 instrucciones de código intermedio utilizadas por el compilador. Posteriormente se hará los comentarios oportunos acerca de las mismas:

Código Intermedio	Descripción
ORG op1	indica en op1 la posición de memoria a partir de la cual se va a situar el código
MVA op1 op2	mueve a op1 la dirección de op2
MVP op1 op2	mueve a op1 el contenido de la dirección de memoria contenida en op2
MV op1 op2	mueve a op1 el contenido de op2
STP op1 op2	mueve a la dirección contenida en op1 el contenido de op2
BR op1	salto incondicional a op1
CMP op1 op2	resta los contenidos de op1 y op2, modificando los biestables de estado
BZ op1	salto a op1 si el bit Z está activo
BN op1	salto a op1 si el bit S está activo
ADD op1 op2 op3	suma el contenido de op2 y op3, almacenándolo en op1
DIV op1 op2 op3	divide el contenido de op2 y op3, almacenándolo en op1
AND op1 op2 op3	realiza el and lógico de op2 y op3, almacenándolo en op1
WRINT op1	escribe en la salida el entero contenido en op1
WRNL	escribe en la salida un salto de línea
WRSTR op1	escribe en la salida la cadena contenida en la región de memoria etiquetada con op1
CADENA op1 op2	reserva una región de memoria etiquetada con op2 para la cadena op1
INL op1	introduce la etiqueta op1
BEGIN_SUB	inicia la secuencia de llamada
SIBLING	establece el enlace de acceso para subp del mismo nivel
CHILD	establece el enlace de acceso para subp de un nivel inferior
PARAM op1	introduce el parametro op1 en el RA del subp (op1 siempre es una dirección)
CALL_FUN op1 op2	culmina las secuencias de llamada a op1 y la de retorno, colocando el valor devuelto en op2
CALL_PROC op1	culmina tanto la secuencia de llamada como la de retorno de op1
REGALLOC op1	reserva el espacio op1 en la pila para el RA
RETURN op1 op2	coloca el valor devuelto, op1, en su espacio reservado en el RA y salta a la secuencia de retorno, op2
END_SUB op1 op2	inserta la etiqueta op1 e inicia la secuencia de retorno ajustando el SP mediante op2
HALT	detiene la ejecución

El lenguaje de código intermedio empleado ha sido inspirado por el juego de instrucciones del ens2001. Se ha intentado llegar a un compromiso entre generalidad y facilidad de traducción a código final. De hecho, las instrucciones resaltadas son en gran medida dependientes de la máquina objeto, pero se ha decidido su uso basándose en que una pequeña dependencia con el ens2001 puede ser asumible, sobre todo si facilita tangiblemente la posterior traducción.

Destacar también el uso de la instrucción ORG, la cual es empleada en el código final para reservar espacio para los datos globales antes del código. De ahí que para saltar los datos globales (al usar ORG se introducen instrucciones NOP) la primera instrucción sea un salto incondicional que haga de *bypass* hasta la dirección de memoria de la primera instrucción del programa principal.

3. Generación de código final

3.1. Descripción de hasta dónde se ha llegado

La traducción a código final implementada abarca la totalidad de las funcionalidades del lenguaje, tanto la parte obligatoria como la opcional.

Se ha decidido colocar en memoria los datos globales antes del código haciendo uso de la directiva de ens2001 ORG. De esta forma se garantiza la no interferencia entre datos globales y código.

Así, el *layout* general sería:

- en la dirección /0 un salto incondicional al inicio del programa. De esta forma se saltan todas las instrucciones NOP introducidas por ORG.
- en la dirección /1 se almacena el valor proporcionado por la directiva ORG, que indica el desplazamiento necesario para reservar espacio a los datos globales.
- en la dirección /2 en adelante se sitúan las variables globales primero y las temporales globales después.
- donde finalice la región de datos globales comienza la región de código.

Otro aspecto a destacar es que el acceso a los datos no locales lo he implementado mediante la técnica de encadenamiento de accesos.

3.2. Descripción del registro de activación implementado

A continuación se muestra el esquema general de la organización de los registros de activación de los subprogramas (notar que la pila crece hacia direcciones decrecientes):

Dirección relativa al puntero de marco IX	Descripción
#0[.IX]	valor de retorno (para procedimientos también, por generalidad)
#-1[.IX]	enlace de control (puntero de marco del subprograma llamante)
#-2[.IX]	estado de la máquina
#-3[.IX]	enlace de acceso (puntero de marco del subprograma padre más recientemente invocado)
#-4[.IX]	parámetros

...	
...	
#-(nP + 4)[.IX]	dirección de retorno
#-(nP + 5)[.IX]	variables locales
...	
...	
#-(nP + tV + 5)[.IX]	temporales locales
...	
...	

Las regiones con fondo coloreado pueden no existir. El número de parámetros será nP (nP puede ser 0), ocupando necesariamente nP posiciones de memoria ya que el paso de parámetros es por referencia y éstas ocupan una palabra. El número de variables será nV (nV puede ser 0), ocupando en total tV palabras en memoria, resultado de sumar los tamaños de cada variable. El número de temporales será nT (nT puede ser 0), ocupando necesariamente nT posiciones de memoria. Así, el tamaño de un registro de activación será: nP + tV + nT + 5. Como ejemplo,el registro de activación más simple sería:

Dirección relativa al puntero de marco IX	Descripción
#0[.IX]	valor de retorno (para procedimientos también, por generalidad)
#-1[.IX]	enlace de control (puntero de marco del subprograma llamante)
#-2[.IX]	estado de la máquina
#-3[.IX]	enlace de acceso (puntero de marco del subprograma padre más recientemente invocado)
#-4[.IX]	dirección de retorno

4. Indicaciones especiales

En medio del desarrollo de la práctica tuve que cambiar de Windows 10 a OS X. En ambos entornos he trabajado con el IDE Eclipse sin problemas, pero al cambiar tuve que comentar la importación inicial:

//import compiler.semantic.*;

De no hacerlo se generaba error, y al comentar la sentencia el error quedaba mitigado y todo funcionaba correctamente.