IIC2323 — Construcción de Compiladores
Entrega 2

Primer Semestre 2014

Descripción General

La entrega consta de cuatro partes: Análisis sintáctico, construcción y uso de tabla de símbolos, construcción y análisis de grafo de tipos, y análisis de corrección de subtipos.

Parte 1: Análisis Sintáctico (30 pts)

En esta parte, se debe utilizar Bison para generar un parser que permita obtener el árbol de sintaxis abstracto. Más específicamente, a partir del lexer de la primera entrega, se deberá imprimir el árbol con el siguiente formato[1]:

Nodo raíz: Program

```
program
|-- <class1>
|-- <class2>
|-- ...
```

Definición de clases

```
class NOMBRE : SUPERTIPO1 SUPERTIPO2 ...
|-- <class element 1>
|-- <class element 2>
|-- ...
```

En caso de que la clase no tenga supertipos, se deben omitir estos.
Los supertipos deben imprimirse en el mismo orden en que están definidos.

Definición de atributo

```
attr [private] ID:TYPE
```

[1] En algunos casos se agregó una forma alternativa. Esta se prefiere, y debería ser más simple, pero ambas son válidas.

Definición de método

```
method [private] NOMBRE : TIPO_RETORNO
|-- arguments
|---- ID1:TYPE1
|---- ID2:TYPE2
|-- body
|---- <statement 1>
|---- <statement 2>
|---- ...
```

Formato alternativo

```
method [private] NOMBRE : TIPO_RETORNO
|-- arguments
|   |-- ID1:TYPE1
|   |-- ID2:TYPE2
|-- body
    |-- <statement 1>
    |-- <statement 2>
    |-- ...
```

La sección 'arguments' será omitida si no hay argumentos. Los argumentos deben ser impresos en orden.

Definición de iterador

```
iterator [private] NOMBRE : TIPO_RETORNO
|-- arguments
|---- ID:TYPE [once]
|---- ...
|-- body
|---- <statement 1>
|---- ...
```

Formato alternativo

```
iterator [private] NOMBRE : TIPO_RETORNO
|-- arguments
|   |-- ID1:TYPE1
|   |-- ID2:TYPE2
|-- body
    |-- <statement 1>
    |-- <statement 2>
    |-- ...
```

La sección arguments será omitida si no hay parámetros. De igual forma, se omitirá el tipo de retorno, si no corresponde.

Include

```
include UPCASE_ID
|-- ID1 --> ID2
|-- ID3 --> ID4
|-- ...
```

Las secciones con la flecha ($--$ >) corresponden a transformaciones (ver parte 4). Por ejemplo, el statement:
`include CARNIVORE eat_meat - eat`
debe imprimirse como:

```
include CARNIVORE
|-- eat_meat --> eat
```

If

```
if
|-- <expression1>
|---- <statement1>
|---- <statement2>
|---- ...
|-- <expression2>
|---- <statement>
|---- ...
|-- else
|---- <statement>
```

Formato alternativo

```
if
|-- condition
|   |-- <expression1>
|   |-- body
|       |-- <statement1>
|       |-- <statement2>
|       |-- ...
|-- condition
|   |-- <expression1>
|   |-- body
|       |-- <statement1>
|       |-- <statement2>
|       |-- ...
|-- else
    |-- <statement 1>
    |-- <statement 2>
    |-- ...
```

La sección else se debe omitir si no está definida.

Return

```
return
|-- expression
```

Se debe omitir expression, si no hay retorno.

Typecase

```
typecase
|-- ID
```

```
|-- <type_specifier>
|---- <statement1>
|---- <statement2>
|---- ...
|-- <type_specifier>
|---- <statement_list>
|-- else
|---- <statement_list>
```

Formato alternativo

```
typecase ID
|-- <type_specifier1>
|   |-- <statement1>
|   |-- <statement2>
|   |-- ...
|-- <type_specifier2>
|   |-- <statement1>
|   |-- <statement2>
|   |-- ...
|-- else
    |-- <statement 1>
    |-- <statement 2>
    |-- ...
```

La sección else se debe omitir si no está definido.

self

```
self
```

Variables

```
local ID
```

Asignación a expresión

```
assign
|-- <expression>
|-- <expression>
```

En este caso se asigna la segunda a la primera. Por ejemplo, count:=36, se imprime como:

```
assign
|-- local count
|-- literal INT 36
```

Asignación con declaración

```
assign ID : TIPO
|-- <expression>
```

En las asignaciones con tipo implícito, es decir de la forma h::=47, se debe omitir el tipo:

```
assign ID
|-- <expression>
```

Declaración de variables

```
decl ID : TIPO
```

Llamada a método/atributo

```
call ID
|-- caller
|---- <expression>
|-- arguments
|---- <expression1>
|---- ....
```

Formato Alternativo

```
call ID
|-- caller
|   |-- <expression>
|-- arguments
    |-- <expression1>
    |-- ....
```

Se debe omitir arguments si no hay argumentos.

Llamada a iterador

```
call ITER_NAME
|-- caller
|---- <expression>
|-- arguments
|---- <expression1>
|---- ....
```

Formato Alternativo

```
call ITER_NAME
|-- caller
|   |-- <expression>
|-- arguments
    |-- <expression1>
    |-- ....
```

Se debe omitir arguments si no hay argumentos.

void

```
void
```

void test

```
void
|-- <expression>
```

new

```
new
```

create

```
create <type_specifier>
|-- arguments
|---- <expression1>
|---- <expression2>
|---- ...
```

Formato Alternativo

```
create <type_specifier>
|-- <expression1>
|-- <expression2>
|-- ...
```

and

```
and
|-- <expression1>
|-- <expression2>
```

or

```
or
|-- <expression1>
|-- <expression2>
```

binary

```
binary <binary_op>
|-- <expression1>
|-- <expression2>
```

unary

```
unary <unary_op>
|-- <expression>
```

loop

```
loop
|-- <statement1>
|-- <statement2>
|-- ...
```

yield

```
yield
|-- <expression>
```

La sección <expression> se omite si no corresponde.

quit

```
quit
```

while

```
while
|-- <expression>
```

break

```
break
```

Expresiones literales

```
literal TIPO VALOR
```

En este caso TIPO es INT, CHAR, BOOL o STR, y VALOR es el valor literal. En el caso de strings y chars, los caracteres escapados se deben reemplazar por el valor literal, por ejemplo, el string "$\backslash tstr$", se debe imprimir como:

```
literal STR " str"
```

En caso de error sintáctico, se debe detener el proceso, e imprimir el mensaje:
`line: Syntax error`
donde line indica la línea donde ocurrió el error. Opcionalmente (bonus), se podrá agregar más información al mensaje, o manejar el error, es decir, no detener el proceso (ver manual de Bison). En caso de hacer esto, se deberán incluir los detalles en el archivo de texto.

Parte 2: Tabla de Símbolos (30 pts)

En esta parte, se pide construir la tabla de símbolos del programa. Esta consiste en un árbol (o un bosque, si se prefiere), donde cada nodo corresponde a un scope del programa, y contiene información acerca de los símbolos definidos en este: las variables, atributos, métodos, e iteradores.

Se considerarán tres tipos de nodos: las clases, que comienzan con la definición y terminan con su último *class_element*; los métodos, que comienzan con su definición y terminan con su último *statement*; y los definidos por variables, que comienzan al declararlas y terminan junto a la lista de statements donde está la declaración.

Por ejemplo, en:

```
loop 3.times!;
    a:INT := 15
    a := a + 3;
    if (a = 0) then
       a := 17
    end
    #OUT+a+"\n"; -- Prints out successively 18, 18, 18
end;
```

el scope de 'a' termina en la penúltima línea.

Sobre las variables, se debe tener al menos su ubicación, sobre los atributos, al menos su tipo de acceso y tipo, y sobre iteradores y métodos, el tipo de retorno, la cantidad y tipos de los argumentos, y el tipo de acceso.

Para evaluarla, se pide utilizar la estructura para contestar las siguientes preguntas (una por línea, en orden).

1. ¿Cual es el máximo de scopes distintos entre todas las clases?.
2. ¿Cuantos atributos privados tiene la primera clase?.
3. ¿Cuantos iteradores se definen entre todas las clases?.
4. ¿Cuantas veces se define la variable 'var1'?.

Considere la última definición de esta:

1. ¿En cuantos scopes distintos se puede utilizar?.
2. ¿Está ocultando alguna definición previa?[2].
3. ¿En que método fue definida?.

En el caso de la primera definición del método 'fun2':

1. ¿Está definido?.
2. ¿En qué clase está?.
3. ¿Es privado?.
4. ¿Cual es su tipo de retorno?.
5. ¿Cuantos argumentos recibe?.
6. ¿Cual es el tipo del segundo argumento?.

En las preguntas con respuesta booleana se debe responder con 1 si la respuesta es afirmativa, o 0 si es negativa. Si alguna pregunta no tiene respuesta, se debe responder NIL.

Para más detalles y ejemplos sobre scoping, se recomienda revisar [1].

Parte 3: Grafo de tipos (15 pts)

Al definir las clases se genera una relación de subtipo. Esta, por especificación, debe corresponder a un grafo dirigido acíclico, donde los nodos son los tipos, y hay un arco de A a B, ssi A es subtipo de B.

En esta parte, se pide construir e imprimir la matriz de adyacencia del grafo, utilizando unos y ceros. Para esto considere las siguientes clases predefinidas:

1. OB: Supertipo de todos los tipos.
2. STR: Subtipo de OB.
3. INT, BOOL, CHAR: Subtipos de STR y OB.
4. OUT, IN: Subtipos de OB.

[2]Aquí hay que tener presente que una variable puede ocultar a un método sin argumentos, o a un atributo

Para la impresión se debe utilizar el orden

$$OB, STR, INT, BOOL, CHAR, IN, OUT.$$

seguido de las clases definidas en el archivo fuente, en el mismo orden.

Si el grafo contiene un ciclo no se debe imprimir, y en su lugar se debe indicar esta situación con el mensaje:

`Semantic error: Cyclic subtyping`

Parte 4: Corrección de Subtipos (25 pts)

Si una clase A es subtipo de B, es decir se define como:

```
A < B
```

entonces A debe contener una implementación de todos los atributos, métodos e iteradores públicos de B. Por ejemplo, si B define el iterador público `next!`, entonces A también debe hacerlo. Para esto hay dos posibilidades: o bien A tiene una implementación explícita, o "incluye" una, usando cláusulas `include`:

Semántica de include

Que una clase C incluya una cláusula `include TYPE`, indica que todos los elementos públicos definidos en `TYPE`, son "copiados", o incluidos en C, y se pueden utilizar directamente. Además, al momento de incluirlos se pueden renombrar para evitar conflictos, utilizando las transformaciones. Por ejemplo, si en la clase A se tiene:

`include B plus - plusB`,

entonces en A se agrega un método de nombre `plusB`, que tiene las mismas características e implementación que el método plus de la clase B.

Al igual que en el caso de subtyping, los includes definen un grafo dirigido acíclico entre los tipos, ya que las inclusiones son transitivas.

Como evaluación, se pide imprimir la matriz de adyacencia del grafo definido por los includes, utilizando el mismo formato de la parte 3, y además detectar errores semánticos relacionados. En particular, estos son:

1. Si el grafo es cíclico, no se debe imprimir, y se debe imprimir el mensaje:

   `Semantic error: Cyclic class including`

2. Si algún elemento no está definido, se debe imprimir:

   `Semantic error: Class CLASS1 must define ID (required by CLASS2)`

   `Semantic error: Class CLASS1 must implement iterator ITER (required by CLASS2)`

3. Si se produce un conflicto, se debe imprimir:

   `Semantic error: Multiple definition of ID`

4. Si se intenta transformar un id que no existe (en el ejemplo, si B no tiene un método `plus`):

   `Semantic error: ID undeclared in INCLUDED_CLASS`

Entrega y evaluación

Se deberá entregar en el mismo repositorio de la entrega 1. Al igual que en el caso de esta, es revisará el contenido de la rama por defecto, al día de la entrega, es decir el día **Viernes 9 de Mayo a las 23:59 horas**.

En el repositorio se deberá incluir un archivo de texto, con una descripción breve y general de la solución de cada parte, que puede profundizar algunos aspectos según se estime conveniente.

La corrección será automática, utilizando un corrector y ejemplos que serán suministrados, los que ejecutarán la tarea usando un script (ver anexo).

En caso de atraso, se descontará un punto por día o fracción, con un máximo de tres días. Esta situación deberá ser avisada por email.

Anexo: Script de corrección

Para compilar y ejecutar la entrega, el corrector automático buscará un archivo [3] con nombre en:

```
ejecutar[a-zA-Z_0-9.]*
```

en el directorio raíz del repositorio, y lo ejecutará con los siguientes parámetros:

```
-i INPUT  : archivo de input
-o OUTPUT : archivo de output
-C        : compilar
-S        : análisis léxico (entrega 1)
-A        : generar el AST (parte 1)
-P        : respuestas a preguntas (parte 2)
-T        : grafo de subtipos (parte 3)
-I        : grafo de inclusiones (parte 4)
```

Para esto, se recomienda utilizar como base el script que será publicado.

Referencias

[1] http://www1.icsi.berkeley.edu/~sather/Documentation/LanguageDescription/webmaker/DescriptionX2Echapter2-1.html#HEADING1-107

[3] el primero que encuentre, se recomienda que solo exista uno