Laboratorio de Programación Funcional 2016
Compilador de MiniPascal

May 20, 2016

La tarea consiste en la implementación de un compilador de una versión reducida y simplificada de Pascal, que llamaremos MiniPascal. El compilador deberá realizar las tareas de chequeo de tipos y generación de código C equivalente al programa MiniPascal compilado.

1 MiniPascal

La siguiente EBNF describe la sintaxis de MiniPascal, donde ⟨*ident*⟩ es un identificador válido y ⟨*nat*⟩ un número natural:

⟨*program*⟩ ::= '`program`' ⟨*ident*⟩ '`;`' ⟨*defs*⟩ ⟨*body*⟩ '`.`'

⟨*defs*⟩ ::= '`var`' {⟨*ident*⟩ '`:`' ⟨*type*⟩ '`;`'}

⟨*type*⟩ ::= '`integer`' | '`boolean`'
| '`array`' '`[`' ⟨*nat*⟩ '`..`' ⟨*nat*⟩ '`]`' '`of`' ⟨*type*⟩

⟨*body*⟩ ::= '`begin`' [⟨*stmt*⟩{'`;`'⟨*stmt*⟩}] '`end`'

⟨*stmt*⟩ ::= ⟨*ident*⟩ {'`[`'⟨*expr*⟩'`]`'} '`:=`' ⟨*expr*⟩
| '`if`' ⟨*expr*⟩ '`then`' ⟨*body*⟩ '`else`' ⟨*body*⟩
| '`for`' ⟨*ident*⟩ '`:=`' ⟨*expr*⟩ '`to`' ⟨*expr*⟩ '`do`' ⟨*body*⟩
| '`while`' ⟨*expr*⟩ '`do`' ⟨*body*⟩
| '`writeln`' '`(`' ⟨*expr*⟩ '`)`'
| '`readln`' '`(`' ⟨*ident*⟩ '`)`'

⟨*expr*⟩ ::= ⟨*ident*⟩ {'`[`'⟨*expr*⟩'`]`'} | '`true` | `false`' | ⟨*nat*⟩
| ⟨*expr*⟩ ⟨*bop*⟩ ⟨*expr*⟩ | ⟨*uop*⟩ ⟨*expr*⟩

⟨*bop*⟩ ::= '`or`' | '`and`' | '`=`' | '`<`' | '`+`' | '`-`' | '`*`' | '`div`' | '`mod`'

⟨*uop*⟩ ::= '`not`' | '`-`'

```
program ejemplo1;
var
begin
end.
```

Figure 1: Programa más simple posible

```
program ejemplo2;
var w : integer;
    x : boolean;
    y : array [2 .. 4] of integer;
    z : array [1 .. 2] of array [1..4] of integer;
begin
   w := 10;
   x := true;
   y[w-8] := 10;
   z[1][1] := 9
end.
```

Figure 2: Declaración de variables y asignación

Un programa tiene un nombre, una lista (posiblemente vacía) de declaraciones de variables y un bloque **begin**−**end** compuesto de una secuencia de instrucciones. En la Figura 1 se muestra el ejemplo de programa más simple que cumple con la sintaxis dada por la gramática de MiniPascal. Notar que, aunque no se declaren variables, las palabra clave **var** es mandatoria.

Al declarar una variable se especifica su nombre y su tipo. En MiniPascal hay tres tipos: **integer**, **boolean** y **array**. En la Figura 2 se puede ver un programa en el que se declaran variables de distintos tipos y luego se realizan asignaciones de valores literales a dichas variables. Con respecto a Pascal se agrega la restricción de que no se puede asignar un arreglo a otro, es decir que por ejemplo z[1] := y no es válido.

Además de las asignaciones se tienen instrucciones de: selección **if**−**then**−**else** (no se puede omitir el **else**), iteración **for** y **while**, y de entrada/salida **writeln** y **readln**. Los procedimientos de entrada/salida pueden tomar sólo un parámetro de tipo entero. En la Figura 3 se muestra un ejemplo de un programa que utiliza varias de estas instrucciones.

Las expresiones pueden ser variables (ej. x, a[2]), literales booleanos, literales enteros, aplicación de un operador unario a una sub-expresión y la aplicación de un operador binario a dos sub-expresiones. Los operadores unarios son la negación − y el no lógico **not**. Los operadores binarios son los operadores lógicos **or** y **and**, los operadores de comparación = y <, y los operadores enteros +, −, ∗, **div** y **mod**.

```
program ejemplo3;
var i : integer;
    x : integer;
begin
   readln(x);

   if x < 1 then
   begin
      writeln(0)
   end
   else
   begin
      for i := 1 to x do
      begin
         writeln(i)
      end
   end
end.
```

Figure 3: Instrucciones

El módulo `Syntax`, provisto en el archivo `Syntax.hs`, contiene un tipo algebraico de datos Program que representa a los árboles de sintaxis abstracta de programas MiniPascal y una función de parsing, que dada una cadena de caracteres con un programa MiniPascal retorna su árbol de sintaxis abstracta o los errores de sintaxis encontrados al intentar parsear:

parser :: **String** −> **Either** ParseError Program

1.1 Chequeos

El compilador debe realizar chequeos de nombres y de tipos, como se describe a continuación. Estos chequeos deben ser realizados por la función del módulo `TypeChecker` que debe ser implementada como parte de la tarea:

checkProgram :: Program −> **Either** [Error] Env

La función toma como entrada el árbol de sintaxis abstracta de un programa y retorna el ambiente (Env) con las variables definidas en caso de no encontrar errores, o los errores encontrados en otro caso.

1.1.1 Chequeo de Nombres

Se debe chequear que no se usen nombres que no se hayan declarado o que no hayan múltiples declaraciones de un mismo nombre. Si se detectan errores en la declaración de variables no se continua con el chequeo del cuerpo. Por ejemplo,

```
program ejemplo4;
var x : integer;
    y : integer;
    x : integer;
    y : boolean;
begin
   readln(z);
   for i := 1 to 10 do
   begin
      w := w + z
   end
end.
```

Figure 4: Definiciones Duplicadas

```
program ejemplo5;
var x : integer;
    y : integer;


begin
   readln(z);
   for i := 1 to 10 do
   begin
      w := w + z
   end
end.
```

Figure 5: Nombres no definidos

en el programa de la Figura 4 se detectan los siguientes errores:

```
Duplicated Definition: x
Duplicated Definition: y
```

mientras que en el programa de la Figura 5 se detectan:

```
Undefined: z
Undefined: i
Undefined: w
Undefined: w
Undefined: z
```

Notar que se reporta cada uso del nombre no definido.

1.1.2 Chequeo de Tipos

Se debe verificar que los tipos de las variables y expresiones sean correctos. Esto es:

- Si se quiere acceder a una celda de un arreglo (ej. a[1]) la variable tiene que ser de tipo **array** y el índice tiene que ser una expresión entera.
- En la asignación los tipos de la variable y la expresión coinciden y no son arreglos.
- En la instrucción **if** la condición es una expresión booleana.
- En el **for** el iterador y las expresiones de inicio y fin son de tipo entero.
- En el **while** la condición es booleana.
- El parámetro de **writeln** es una expresión entera.

```
program ejemplo6;
var i : boolean;
    x : integer;
begin
   readln(i);

   if i and x then
   begin
      writeln(0)
   end
   else
   begin
      for i := 1 to x do
      begin
         writeln(i)
      end
   end
end.
```

Figure 6: Errores de Tipos

- El parámetro de **readln** es una variable entera.
- Las sub-expresiones de una expresión tienen los tipos correctos de acuerdo al operador utilizado (ej. en e1 + e2, e1 y e2 deben ser enteros).

Por ejemplo, en la Figura 6 se muestra un programa que generaría los siguientes errores:

```
Expected: integer Actual: boolean
Expected: boolean Actual: integer
Expected: integer Actual: boolean
Expected: integer Actual: boolean
```

Dado que se pasa una variable booleana al **readln**, uno de los operandos del **and** es de tipo entero, el iterador del **for** es booleano y se pasa una expresión booleana al **writeln**.

En la Figura 7 se muestran algunos errores en asignaciones, que son los siguientes:

```
Expected: integer Actual: boolean
Expected: integer Actual: boolean
Type integer is not an array
Expected: integer Actual: boolean
Array assignment: array [1 .. 2] of integer
```

```
program ejemplo7;
var x : integer;
    b : boolean;
    a : array [1..2] of integer;
begin
   x := b or true;
   b := b + 2;
   x[1] := 2;
   a[b] := 8;
   a := a
end.
```

Figure 7: Errores de Tipos en Asignaciones

Asignar el resultado de una expresión booleana a una variable entera, usar un booleano como operando de la suma, utilizar una variable entera como si fuera un arreglo, usar una expresión booleana como índice de un arreglo y asignar a una variable de tipo arreglo.

1.2 Generación de Código

Luego de superar de forma exitosa la fase de chequeos, el compilador debe proceder a generar código que pueda ser ejecutado por una máquina. En este caso generaremos código C, que luego pueda ser compilado y ejecutado.

La generación de código será realizada por la función del módulo `Generator` que debe ser implementada como parte de la tarea:

genProgram :: Program −> Env −> **String**

Como las construcciones de MiniPascal y C son similares, la traducción es bastante directa, salvo por algunos detalles que hay que tener en cuenta:

- En C no hay booleanos. Se pueden codificar con enteros, donde 0 es false y distinto de 0 (ej. 1) es true.
- Todos los arreglos en C comienzan en 0.
- Las palabras reservadas de MiniPascal y C no son las mismas, por lo tanto un identificador válido en MiniPascal puede ser una palabra reservada en C. Para resolver esto en la generación se debe agregar un caracter '_' al comienzo de cada identificador.

Teniendo lo anterior en cuenta, en las figuras 8, 9 y 10 se muestran los códigos generados a partir de los programas de las figuras 1, 2 y 3, respectivamente.

```
#include <stdio.h>
void main() {
}
```

Figure 8: Código generado a partir de `ejemplo1`

```
#include <stdio.h>
int _w;
int _x;
int _y[3];
int _z[2][4];
void main() {
_w = 10;
_x = 1;
_y[_w - 8 - 2] = 10;
_z[1 - 1][1 - 1] = 9;
}
```

Figure 9: Código generado a partir de `ejemplo2`

```
#include <stdio.h>
int _i;
int _x;
void main() {
scanf ("%d", &_x);
if (_x < 1){
printf ("%d\n",0);
}else{
for (_i=1;_i <= _x;_i++ ){
printf ("%d\n",_i);
};
};
}
```

Figure 10: Código generado a partir de `ejemplo3`

2 Se Pide

Además de esta letra el obligatorio contiene los siguientes archivos:

`Syntax.hs` Módulo que contiene el parser y el AST.

`TypeChecker.hs` Módulo de chequeos.

`Generator.hs` Módulo de generación de código.

`Compiler.hs` Programa Principal, importa los tres módulos anteriores y define un compilador, que dado el *nombre* de un programa MiniPascal obtiene el programa de un archivo *nombre.pas*, chequea que sea válido y en caso de serlo genera un archivo *nombre.c* con el código C correspondiente. En otro caso imprime los errores encontrados.

`ejemplo`*i*`.pas` Programas MiniPascal usados como ejemplos en esta letra.

`ejemplo`*i*`.c` Programas C generados en caso de que no se encuentren errores en los chequeos.

`ejemplo`*i*`.err` Mensajes de error impresos por el compilador en caso de que se encuentren errores en los chequeos.

La tarea consiste en modificar los archivos `TypeChecker.hs` y `Generator.hs`, implementando las funciones solicitadas, de manera que el compilador se comporte como se describe en esta letra.

Los únicos archivos que se entregarán son `TypeChecker.hs` y `Generator.hs`. Dentro de ellos se pueden definir todas las funciones auxiliares que sean necesarias. No modificar ninguno de los demás archivos, dado que los mismos no serán entregados.