CS3025 Compiladores

Proyecto Final

Análisis Sintáctico, Semántico , Interpretes y Generación de Código

Entrega: Viernes, 24 Noviembre, 2pm

Grupos de 3 (máximo)

El folder `proyecto_codigo` tiene la implementación del parser, printer, intérprete y typechecker del lenguaje IMP0 definido por la siguiente sintaxis:

```
Program ::= Body
Body ::= VarDecList StatementList
VarDecList ::= (VarDec)*
VarDec ::= "var" Type VarList ";"
Type ::= id
VarList :: id ("," id)*
StatementList ::= Stm (";" Stm)* …
Stm ::=   id "=" Exp |
          "print" "(" Exp ")" |
          "if" Exp "then" Body ["else" Body] "endif" |
          "while" Exp "do" Body "endwhile"
          "for" id ":" Exp "," Exp "do" Body "endfor"
Exp ::= BExp
BExp ::  CExp (('and'|'or') BExp)?
CExp ::= AExp (('<'|'<='|'==') AExp)?
AExp ::= Term (('+' | '-') Term)*
Term ::= Factor (('*'|'/') Factor)*
FExp ::= Unary ("**" FExp")?
Unary ::= ('!'|'-')? Factor
Factor ::= num | '(' Exp ')' | id
          "ifexp" '(' Exp ',' Exp ',' Exp ')'
```

Ademas, se incluye la implementacion del generador de codigo objeto para toas las expresiones y sentencias, con la excepción de los pedido en la pregunta 2.

Para el proyecto final, se les pide modificar el código dado en `proyecto_codigo` de tal manera que implemente cada uno de los puntos indicados abajo. Además, la entrega final deberá incluir un documento (breve) donde se explique las modificaciones hechas al código y, dependiendo de la pregunta, las definiciones de `tcheck` y `codegen` usadas en las implementaciones.

El proyecto incluirá una pequeña presentación donde se correrá el código y se responderán preguntas asociadas al código y reporte.

1) Comentarios

Agregar a IMP0 la posibilidad de incluir comentarios de una sola línea en cualquier punto del programa. Los comentarios deberán empezar con `//` y acabar con el fin de línea. Así, por ejemplo, se podrá escribir código IMP0 como el de abajo:

```
var int x, n;  // variables globales
n = 10;  // longitud
x = 0;
while x < n
   print(10** x)  // imprimir potencia de 10
   x = x+1;
endwhile
```

Reporte: ¿Que cambios se hicieron al scanner y/o parser para lograr la inclusión de comentarios?

2) Generación de código I

La tarea 3 pedía la implementación de constantes booleanas,los operadores and/or y una versión de foor lops. El código que acompaña este proyecto tiene estas implementaciones. Además contiene implementaciones vacías del Visitor que implementa `codegen` (para permitir que todo compile).
Implementar la generación de código objeto para constantes booleanas, los operadores `and/or` y la versión de for loops considerada por el intérprete: solo considera for-loops con valores ascendentes. Por ejemplo

```
for x : 5, 10 do Body endfor
```

ejecuta `Body` 6 veces con x : 5,6,…,10 pero

```
for x : 10, 5 do Body endfor
```

no ejecuta `Body` ni una sola vez.

Reporte: Incluir las definiciones de `codegen` utilizadas en las implementaciones.

Nota: La SVM acepta las instrucciones `and` y `or`.

3) Sentencia do-while

Implementar la interpretación estándar de `do-while`. Por ejemplo:

x= 0; do x = x+ 1; print(x) while x < 5

imprime 1,2,3,4,5
Nótese que el `do-while` no necesita un marcador como `enddo`. Para delimitar el fin de la sentencia. ¿Se puede implementar el parser? Pero, puede agregarse algo asi si lo desean.

Reporte: Indicar el cambio a la gramática y los puntos donde se hicieron cambios al código. Además, proveer las definiciones de `tcheck` y `codegen` usadas.

4) Sentencias break y continue

Implementar las interpretaciones estándar de `break` y `continue`, las sentencias que permiten salir y terminar un loop o saltar a la condición de control del loop. Notar que el salto asociado a un continue es distinto para un `while-do` que para un `do-while` (al comienzo o al final).

No es necesario implementar la parte asociada al interprete.

Reporte: Indicar el cambio a la gramatica y los puntos donde se hicieron cambios al código. Además, proveer las definiciones de `tcheck` y `codegen` usadas.