CS3025 Compiladores

Proyecto Parte I
TypeCheking y Generación de Código
Entrega: Lunes, 17 Junio, 11:59pm
Grupos de 3 (máximo)

El folder `proj1` tiene la implementación de la máquina virtual SVM, y la implementación del parser, printer e intérprete del lenguaje IMP-DEC 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"
Exp ::= BExp
BExp ::  CExp (('and'|'or') BExp)?
CExp ::= AExp (('<'|'<='|'==') AExp)?
AExp ::= Term (('+' | '-') Term)*
Term ::= Factor (('*'|'/') Factor)*
FExp ::= Factor ("**" Factor")?
Factor ::= num | '(' Exp ')' | id
          "ifexp" '(' Exp ',' Exp ',' Exp ')'
```

Además, se incluye las clases necesarias para la implementación del verificador de tipos (`imp_typechecker`) y generador de código objeto (`imp_codegen`) para todas las expresiones y sentencias, con la excepción de lo pedido en la pregunta 3.

Para este proyecto, se les pide modificar el código dado en `proj1` 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.

Para cada pregunta, incluir ejemplos de código fuente IMP-DEC que demuestren el correcto funcionamiento de sus implementaciones.

1) Verificador de tipos (typechecker) y Generador de codigo

Implementar el verificador de tipos y generador de código IMP-DEC. Las clases necesarias para este paso están listas, solo será necesario agregar código, y crear variables auxiliares si es necesario, a los métodos correctos. Los laboratorios de esta semana estarán dedicados al typechecker. Los laboratorios de la próxima semana se usarán como ayuda para finalizar el proyecto.

El proceso de compilación esta descrito en el Makefile:

```
>> make compiler
>> ./compile ejemplo22.imp
```

En el ejemplo de arriba, el compilador toma como input un programa IMP-DEC, lo parsea e imprime, realiza la verificación de tipos, ejecuta el programa (interprete) y, si no ha habido ningún error, genera el archivo `ejemplo22.imp.sm` con el código SVML correspondiente al programa.

El código generado debe de ser ejecutado correctamente por la máquina virtual SVM. La maquina virtual puede ser generada usando el Makefile:

```
>> make svm
>> ./svm ejemplo22.imp ejemplo22.imp.sm
```

El ejemplo de arriba ejecuta la máquina virtual SVM usando a `ejemplo22.imp.sm` como input. El programa imprime el programa SVML , lo ejecuta y muestra los contenidos de la pila.

Un aspecto clave de su implementación será la asignación de direcciones de memoria a las variables del programa. Estas direcciones serán guardadas por el environment direcciones. Como regla, asignaremos direcciones de izquierda a derecha por cada declaración de variables. El código de abajo muestra (en los comentarios) las direcciones de memoria asignadas a cada variable.

```
var int x, n;  // variables globales: x(1), n(2)
n = 10;
x = 0;
if x < n then
   var bool a, b, x; // validas dentro de if-then: a(3), b(4), x(5)
…
else
   var bool c, d;  // validas dentro de if-else: c(3), d(4)
endif
```

Notar que las direcciones empiezan en 1 dado que la dirección 0 es reservada, y la asignación de arriba hace uso eficiente de memoria: a, b, y c, d usan las mismas direcciones de memoria porque nunca existen al mismo tiempo. Así, la memoria necesaria para guardar las variables del programa de arriba es 5 y el programa generado (SVML) deberá empezar con la línea:

```
alloc 5
```

Reporte: ¿Cómo se calculó la memoria necesaria para las variables globales?

El cálculo de este número necesita una pasada (visit) por el árbol sintáctico; esto puede hacerse en el typechecker. De ser así, la información calculada por el typechecker debe de estar disponible para ser leída por `imp_codegen`. Esto puede hacerse pasando al typechecker como argumento al constructor de `imp_codegen`.

2) Generación de código I

Agregar a IMP-DEC 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 producto
   x = x+1;
endwhile
```

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

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? Si, pero, puede agregarse algo así 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.