Departamento de Cs. e Ingeniería de la Computación
Universidad Nacional del Sur

Compiladores e Intérpretes

Proyecto Nº 2

Aplicación utilizando JLex y CUP

Segundo Cuatrimestre de 2012

La realización del presente proyecto, tiene como principal interés la experimentación con herramientas automáticas para el desarrollo de compiladores, en este caso particular se utilizaran JLex y CUP.

El trabajo consistirá en el desarrollo de un intérprete de sentencias Python reducido. Para tal fin se les brindará la descripción (no completa) de éste lenguaje particular, de gran sencillez, al cual llamaremos μPy.

1. Especificación de μPy

El lenguaje μPy es capaz de procesar secuencias de sentencias separadas, al igual que en Python, por "$\backslash n$"(enters). Básicamente, estas sentencias pueden ser sentencias `print` o *asignaciones*, ambas siguiendo el estilo de Python pero considerando menos elementos. Las sentencias `print` permitirán mostrar por pantalla el resultado de evaluar una expresión. Las asignaciones tendrán la semántica clásica de una asignación en Python, permitiendo la declaración implícita de variables. De esta manera una asignación establece el tipo y el valor de una variable a partir de la evaluación de una expresión. Las variables tendrán, entonces, una ligadura dinámica a su tipo; por lo tanto, una variable podrá cambiar su tipo si, como parte de una nueva asignación, se le asigna un tipo diferente al que estaba ligada. Además, se permitirían utilizar las asignaciones compuestas "`+=`" y "`*=`" con su semántica clásica.

El lenguaje será capaz de manejar expresiones sobre *enteros* y *strings*, los cuales en efecto serán los tipos que pueden tomar las variables. Para los *enteros* se podrán utilizar los operadores aritméticos básicos `+`, `-`, `*`, `/` mientras que para los *strings* se podrá utilizar la concatenacion "`+`", la repetición "`*`", y *Slices*. Además, se cuenta con un operador de conversion "`str`" que transforma *enteros* en *strings*.

Los operadores aritméticos no admiten expresiones mixtas (operan con dos operandos del mismo tipo) y el operador de conversión sólo admite un operando del tipo esperado. Sin embargo, la sentencia **print** admite operandos de ambos tipos (enteros y string). Si se intenta operar un valor entero con un string o viceversa, se produce un error semántico. Además, si se utiliza una variable en una expresión esta tiene que haber sido declarada (implicitamente) en forma previa. En caso de utilizar una variable no declarada se produce un error semántico.

La los operadores de suma y resta tienen la menor precedencia, luego en un segundo nivel se tienen los operadores de multiplicación y división y finalmente los operadores slices y conversion quienes tienen la máxima precedencia. Para todos estos operadores la asociatividad es a izquierda.

Por ejemplo, si se considera la siguiente asignación:

```
a = ‘‘hola ’’ + ‘‘mundo ’’ + str (2*3+4)
```

El interprete de μPy al ejecutarla, en primer lugar evaluará la expresión de la derecha de la asignación que dará como resultado `‘‘hola mundo 10’’`. Luego, si `a` no fue previamente declarada, crea la entrada para `a` ligandola con el tipo *string* y el valor `‘‘hola mundo 10’’`. En caso que `a` hubiese estado declarada simplemente la liga al nuevo valor y, en caso de ser necesario, al nuevo tipo.

En lineas generales, el esquema del lenguaje es el siguiente:

<Program>	→	<StatementList> `.`
<StatementList>	→	<StatementList> `enter` <Statement> | <Statement>
<Statement>	→	<Assignment> | <Print>
<Print>	→	`print` <Expression>
<Assignment>	→	<ID> `=` <Expression> | <ID> `+=` <Expression> | <ID> `*=` <Expression>
<Expression>	→	<Expression> `+` <Expression> | <Expression> `-` <Expression> | <Expression> `*` <Expression> | <Expression> `/` <Expression> | <ID> | <ID><Slice> | <Num> | <String> | <String><Slice> | `str` <Expression> | `(` <Expresión> `)`
<Slice>	→	`[` <Expression> `]` | `[` <Expression>`:`<Expression> `]`
<ID>	→	<Letter> <ID'>
<ID'>	→	<Letter> <ID'> | <Digit> <ID'> | λ
<Num>	→	<Digit> <Num'>
<Num'>	→	<Digit> <Num'> | λ
<String>	→	`‘‘` <StrContent> `’’`
<StrContent>	→	<Letter><StrContent> | <Digit><StrContent> | `white`<StrContent> | λ

donde <Letter> es cualquier letra del alfabeto mayúscula o minúscula y donde <Digito> es cualquiera de los dígitos del 0 al 9. El terminal `enter` corresponde al salto de linea, mientras que el terminal `white` denota el espacio en blanco.

El intérprete, además de evaluar y ejecutar el programa, deberá realizar los chequeos de tipo necesarios para asegurar que la evaluación de las expresiones se realicen de manera acorde con las reglas de compatibilidad de tipos enunciadas.

2. Consideraciones del proyecto

Este proyecto se hará también de manera grupal. La comisión de trabajo estará formada por los mismos integrantes del proyecto 1. Los pasos o etapas a cumplir para el desarrollo del proyecto son:

1. Identificar los componentes léxicos del lenguaje y sus patrones asociados.
2. Dar una gramática completa para el lenguaje descripto sustituyendo los no terminales por terminales y eliminando las producciones en los casos en que las mismas contengan aspectos del lenguaje ya tratados durante el análisis léxico.
3. Dar una Definición Dirigida por la Sintaxis que realice las evaluaciones **semanticas** adecuadas para μPy (con los correspondientes chequeos de tipo y manejo de símbolos), sobre la gramática dada en el inciso anterior.
4. Implementar el lenguaje utilizando JLex y CUP. Este paso implica la construcción de un analizador léxico en JLex y un analizador sintáctico en CUP. Al analizador sintáctico implementado, se le realizará una extensión semántica para que ejecute correctamente las sentencias, calcule adecuadamente el valor de las expresiones, mantenga los valores de los identificadores en ejecución, realice el chequeo de tipos, y reporte los errores que se encuentren.
5. En la documentación del proyecto, también debe incluirse una descripción general de la aplicación y una explicación de los métodos auxiliares implementados o usados.

El resultado de la interpretación será la salida por pantalla (salida estándar) producida por la secuencia de `print`'s o un mensaje de error. El mensaje de error (a excepción de los errores sintácticos) deberá especificar claramente el tipo de error ocurrido (Léxico, Semántico o en Tiempo de Ejecución), el problema encontrado y el número de línea donde ocurrió. El código fuente se tomará del teclado (entrada estándar).