CS3025 Compiladores

Laboratorio 1

14/08/2023

En este laboratorio implementaremos un intérprete y un compilador de un lenguaje que, además de simple, todos conocemos bien: expresiones aritméticas. El intérprete será, básicamente, una calculadora. El compilador transformara expresiones aritméticas a una secuencia de instrucciones que podrán ser ejecutadas en una Maquina de Pila (Stack Machine).

1. El codigo inicial

Ir a Canvas, semana 1 del curso, y bajar el archivo lab1.cpp. El programa, además de la función main(), contiene las declaraciones y definiciones de las siguientes clases:

- `Token` y `Scanner`, a cargo del análisis lexicográfico (lexical analysis)
- `Parser`, a cargo del análisis sintactico (syntactic analysis).
- `Exp` y subclases, implementan/representan el árbol sintáctico abstracto (AST) de expresiones aritméticas.

El programa esta compuesto de un solo archivo. Esto es mala practica[1] pero, por ser primera vez, nos hará las cosas mas fácil. El código del scanner y parser están completos. La clase abstracta Exp define cuatro métodos virtuales:

- void print(): Imprime el AST – deberia coincidir con el input (implementado)
- int eval(): la calculadora (por implementar)
- void rpn(ostringstream& str) : genera la version RPN del input en `str`. (por implementar)
- void genCode(ostringstream& str): genera el codigo listo para ser ejecutado por un stack machine (por implementar).

El programa puede ser compilado con: `g++ lab1.cpp`
Utilicen su IDE favorito – yo generalmente trabajo de la línea de comandos.

El ejecutable necesita un string de input. Por ejemplo

```
./a.out "3 * 2 + 10 * (2 * 3)"
```

Genera

```
expr: 3 * 2 + 10 * (2 * 3)
eval: 0
rpn:
---------- stack machine code ------------------
```

Solo el codigo correspondiente a `exp->print()` es proporcionado. Ustedes harán el resto.

2. El intérprete: una calculadora

Implementar int Exp::eval() de tal manera que retorne el resultado de evaluar la expresión aritmética correspondiente. Para esto se necesitara hacer un recorrido del AST similar al hecho por `print()` pero en lugar de imprimir se harán operaciones aritméticas.

3. RPN: Reverse Polish Notation

[1] Me olvide de los destructores de Exp!!

Reverse Polish Notation (RPN o notacion polaca inversa) es la notación donde los operandos son escritos primero, seguidos de los operators i.e. notación posifija. Esto ayuda a eliminar ambigüedad y simplificar evaluación eg no se necesitan paréntesis.
Por ejemplo, la RPN de la expresión "`3 * 2 + 10 * (2 - 3)`" es :

```
3 2 * 10 2 3 - * +
```

Es importante notar que la manera natural de evaluar una expresión en RPM es utlizando una maquina de pila.
Implementar el metodo `void rpn(ostringstream& str)` donde la notación RPN del input es generada en el stream `str`. Notese que se necesitara un recorrido postfijo del AST.

4. El compilador: generación de código para una maquina de pila.

Una maquina de pila realiza las operaciones (aritméticas) con datos guardados en una pila operacional (stack). Las operaciones que necesitamos son:

- `push n` : coloca el numero n arriba de la pila.
- `op` : realiza la operación binaria (`add`, `sub`, `mult`, `div`) con los dos 2 primeros elementos de la pila. Remueve los 2 primeros elementos y coloca el resultado arriba de la pila.

Otras operaciones: `pop`, `swap`, `peek`, etc. Solo nos concentraremos las 2 mencionadas arriba.

Implementar el método `void rpn(ostringstream& str)` de tal manera que el programa final genera, para el input lo siguiente:

```
expr: 3 * 2 + 10 * (2 - 3)
eval: -4
 rpn: 3 2 * 10 2 3 - * +
---------- stack machine code ------------------
push 3
push 2
mult
push 10
push 2
push 3
sub
mult
add
```

La similitud con RPN es evidente.

Pregunta para el viernes:
Nuestra máquina de pila no implementa directamente la operación '^' (pow) e.g. 2^3 evalua a 8 en nuestra calculadora. Que código debería generarse o que otras instructiones son necesarias?