Universidad Simón Bolívar
Departamento de Computación y T.I.
CI–3725 Traductores e Interpretadores
Abril–Julio 2010

**Proyecto II – 2da entrega borrador, versión 0.1**

# Lenguaje para graficar funciones y datos

La segunda entrega del segundo proyecto consiste en completar parte del interpretador del lenguaje para graficar funciones, partiendo del analizar léxico/sintáctico elaborado en la primera entrega.

## 1. gnuplot

El interpretador que elaboraremos usará la herramienta de dominio público `gnuplot` para graficar las funciones. De todos los comandos de `gnuplot`, los únicos que pueden ser usados son descritos a continuación. Además, en la descripción se dicen exactamente que opciones de ellos podrán usarse. Por lo tanto, lo que genera su interpretador debe funcionar con lo descrito a continuación:

- `set term pdf`: especifica que los grafos se guardarán en PDF.
- `plot <ranges> <function> with <style>, <function>with <style>, ...`: donde `<ranges>` es de la forma `[n:m]`, con `n` y `m` son números, `<function>` es una expresion matemática cuya única variable libre es `x`, `<style>` es uno de los siguiente: lines, points, linespoints. Nótese que la variable libre en `<function>` es `x`, y no puede ser otra variable como `z` o `w`.

Usando estos comandos, es posible generar una grafica en formato pdf así:

```
gnuplot < a.pl > a.pdf
```

Donde el archivo a.pl contiene el siguiente texto:

```
set term pdf
plot [-pi:2*pi/5] x**3 with lines, sin(x) with points
```

Respecto al lenguaje de expresiones matemáticas descritas abajo, casi todo coincide con gnuplot, excepto la exponenciación. En gnuplot, `2**5` es equivalente a $2^5$.

## 2. Descripción del lenguaje

A continuación describimos recordamos los elementos del lenguaje que serán implementados para la entrega de esta parte.

Las expresiones matemáticas (*em*) son las siguientes y evaluan según las convenciones matemáticas normales. Clarificamos donde es necesario. Diremos que una *em* es *evaluable* si no contiene ninguna variable. Diremos que una *em* es un *escalar*, si retorna un numero siempre que todas las variables que ocurren en ellas sean números. Diremos que una *em* es un *arreglo*, si retorna un arreglo siempre que todas las variables que ocurren en ellas sean números. Diremos que una *em* es *evaluable a un número*, si es evaluable y no retorna un arreglo.

- Números, enteros o de punto flotante. Por ej: 8, -5, 1.2, 5.2e-3.
- Las constantes `pi` y `e`, equivalentes a una aproximación con al menos seis decimales correctos.
- Variables: cualquier cadena alfabética que no corresponda a otros elementos del lenguaje. Por ej: `x`, `y`, `ax`
- `( em )`, una expresión matemática parentizada
- El resultado de las operaciones binarias sobre dos *em*. Los operadores binarios son `+, -, *, /, ^`. Esta ultima es la exponenciación, por ej: `2^5` es equivalente a $2^5$.
- El resultado de la operación unaria - sobre una *em*. Dicha operación permite negar la expresión. Por ej: `-(1+x)`
- `f ( em )`, el resultado de aplicar una función $f$ sobre una expresión matemática *em*. Se cuenta además con las siguientes funciones predefinidas: `sin, cos, tan, exp, log, ceil y floor`. Por ej: `sin(1)`. Las funciones `sin, cos, tan` son las funciones trigonometricas usuales. Las funciones `exp` y `log` son exponenciación y logaritmo base `e`. Las funciones `ceil` y `floor` equivalen a redondear un numero punto fijo hacia el numero entero más cercano por debajo y por encima, respectivamente.
- Los arreglos de *em*'s son expresiones matemáticas, pero no serán considerados en esta entrega `range(x,y)` para obtener arreglos – aunque si más adelante con otra semántica – ni los arreglos por comprensión

Los arreglos son secuencias de expresiones matemáticas. Por ejemplo: `[1,8,sin(8)]` es un arreglo. Al aplicar una función o una operación unaria sobre un arreglo, el resultado es el arreglo con la función u operación aplicada a cada uno de sus elementos. Por ejemplo,

- `sin([1,2,3])` es equivalente a escribir `[sin(1),sin(2),sin(3)]`
- `-[1,2,3]` es equivalente a escribir `[-1,-2,-3]`

Al aplicar las operaciones binarias `+, -, *, /, ^` sobre un arreglo y una expresion que no sea un arreglo, el resultado será, por ejemplo:

- `1+[2,3,4]` es equivalente a `[1+2,1+3,1+4]`
- `[2,3,4]/5` es equivalente a `[2/5,3/5,4/5]`.
- `2^[2,3,4]` es equivalente a `[2^2,2^3,2^4]`

Al aplicar las operaciones binarias `+, -` sobre dos arreglos, el resultado será el arreglo que resulte de aplicar la operación, uno a uno, a los elementos de ambos arreglos. Por ejemplo,

- `[1,2,3]+[4,5,6]` es equivalente a escribir `[1+4,2+5,3+6]`

En cambio, para la operacion binaria `*`,

- `[1,2,3]*[4,5,6]` es equivalente a escribir `(1*4) + (2*5) + (3*6)`

Las otras operaciones binarias reportan error si son aplicadas sobre dos arreglos, también si las dimensiones de los arreglos no coinciden. Por ejemplo, `[1,2,3]+[4,5]` debería reportar error.

Una expresión graficable es una expresion graficable simple, o un arreglo de expresiones graficables simples. Una expresion graficable simple es una expresion matemática que es un escalar, es decir, que no es un arreglo. En una expresion graficable ocurre cero o más veces una misma variable, pero no pueden ocurrir dos variables diferentes. El caso de una expresión `'file'` no será considerado en esta entrega. Tampoco la funcion especial `if(cond,exp1,exp2)`. Los instrucciones del lenguaje son las siguientes:

- `f(x) = exp`, donde `f` es una cadena alfabética que representa el nombre de una función, `x` es una variable, y `exp` es una expresión matemática. La expresión sólo puede hacer referencia a la variable `x`, a constantes, y a funciones previamente definidas, pero no a `f`. De lo contrario debe reportarse un error.

- `plot rango , exp {with estilo}`, donde `exp` es una expresión graficable, la parte entre llaves – `{}` – es opcional, `estilo` es una de las siguientes palabras: `lines, points, linespoints`, y `rango` es una expresion de la forma `range(x,y)`, donde `x`, `y` son expresiones matemáticas evaluables a número.

  - Para `plot range(x,y) ...` se generará en gnuplot una llamada *plot [x:y] ...*. Es decir, que *range* no genera un arreglo, como se había dicho en la primera entrega.
  - El `estilo` también puede ser un arreglo de estilos, para el caso en que se esté graficando una expresión que sea un arreglo de expresiones. Por ejemplo: `[lines, lines, points]`.
  - Así, para la expresion `plot range(-10,10), x^2`, debería usarse gnuplot a fin de genera un grafico con un comando como `plot [-10:10] x**2`.
  - La expresion `plot range(-pi,pi/2) [y^(1+1), sin(y)] with [lines, points]` podría usar en gnuplot, por ejemplo, alguna de las siguientes expresiones
    - `plot [-pi:pi/2] x**2 with lines, sin(x) with points`
    - `plot [-pi:pi/2] x**(1+1) with lines, sin(x) with points`

    Nótese que aunque la expresión original estaba definida sobre la variable `y`, gnuplot requiere que la variable se llame `x`, por lo que debió ser renombrada.
  - Nótese que tal como está definido arriba, las expresiones a graficar con gnuplot no pueden contener llamadas a funciones, por lo que si tenemos en nuestro lenguaje
    - `f(y) = 2*y^2 + 5;`
    - `plot range(-10,10) [f(y),f(y)*y];`

    podria graficarse en gnuplot con una expresion como
    `plot [-10:10] 2*x^2 + 5, (2*x^2 + 5)*x`

- Las expresiones `v = exp`, `for v in rango {step paso} endfor`, y `push_back(v,exp)` no serán consideradas en esta entrega.

## 3. Segunda entrega

Basado en el analizador que hizo para la primera entrega, haga los cambios necesarios para obtener un graficador como fue descrito en la sección anterior. Puede que le resulte útil realizar cambios en la gramática, pero estos no serán evaluados. Además, todas las expresiones que formaban

parte del lenguaje según la primera entrega pero que no será usados en esta entrega, no serán usados en los casos de prueba de esta segunda entrega. Por lo tanto, no tiene que cambiar el comportamiento de su programa respecto a esos aspectos.

Los objetivos fundamentales son que el programa funcione adecuadamente, que internamente siga usando las herramientas de análisis léxico y sintáctico, que use adecuadamente la tabla de símbolos para almacenar las funciones, y que reporte los errores semánticos descritos arriba.

En esta entrega su ejecutable deberá llamarse `mygnuplot`. Para usarlo se ejecutará

`./mygnuplot <entrada>.txt`

que guardará en el archivo `<entrada>.pdf` el resultado de graficar `<entrada>.txt`. El que los resultados gráficos se llamen como el archivo de prueba facilitará la corrección.

Requerimientos adicionales:

- El proyecto deberá funcionar en Linux, en las máquinas del LDC.
- Entregar un `Makefile` e instrucciones para compilarlo.
- El código debe estar suficientemente comentado para ser comprendido.