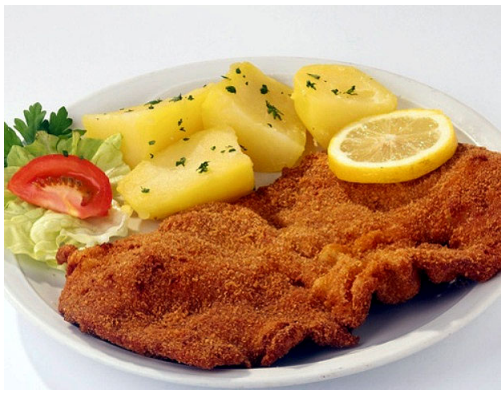
Trabajo Práctico
Teoría de Lenguajes
"Graficador de Curvas Mylanga"

Primer cuatrimestre 2014

1. Introducción

El objetivo del TP es crear un lenguaje de programación imperativo que permita definir funciones que serán evaluadas y nos permitirán graficar curvas paramétricas en el plano. El nombre del lenguaje es Mylanga viene de la expresión *my language*. Desafortunadamente el nombre se desvirtúa y nos recuerda una rica milanesa, como la de la figura:



En la Sección 2 daremos una especificación del lenguaje en forma coloquial. Esta especificación estará complementada por un conjunto de tests en donde pondremos ejemplos de uso del lenguaje. Es parte de la resolución del TP definir una gramática adecuada para el lenguaje.

En la Sección 3 damos un ejemplo que ayudará a definir la gramática y en la Sección 4 describimos el pipeline completo que procesará este lenguaje y que Uds. deben programar.

En la Seccion 5 ponemos algunas preguntas conceptuales que deben responder a medida que resuelven el tp. Las respuestas deben estar redactadas en el informe, cuyo contenido se detalla en la última Sección 6.

2. Lenguaje a emplear

Un programa bien formado de este lenguaje consta de una o más definiciones de funciones seguidas de un comando para dibujar por pantalla dos de ellas (puede ser dos veces la misma). Las funciones a graficar toman parámetros. Estos dependen de una sóla variable, que se mueve en un rango especificado(ver ejemplo en Sección 3).

Dentro de la definición de una función está el nombre, la signatura (los parámetros formales) y un bloque de instrucciones. Este último es o bien una instrucción sola o bien una llave, un conjunto de instrucciones y una llave de cierre.

Una instrucción puede ser la asignación de una expresión a una variable. Podemos también utilizar estructuras de control como if/then/else y bucles while que tienen una guarda con una condición. Una condición es una comparación de dos expresiones aritméticas, o la negación (!cond) de una condición, o la conjunción o disyunción (&&, || de condiciones). Un cuerpo de una estructura de control está compuesto por un bloque de instrucciones.

Una instrucción también puede ser un return seguido de una expresión. Las funciones toman cero o más parámetros y devuelven siempre un único valor. Una función siempre debe terminar su ejecución en un return, si se queda sin instrucciones para ejecutar la función debe tirar un error.

El código debe admitir comentarios multilínea.

Como ejemplo, el lenguaje a emplear debe permitir programar las siguientes funciones: seno, cuadrática, factorial, polinomio de Pochammer (ver Apéndice A).

Entre otras operaciones, se pueden hacer expresiones (sumas, restas, multiplicaciones, divisiones, potencias, el menos unario y llamadas a función). La interpretación de la aritmética debe ser la usual (es decir utilizando la precedencia y asociatividad de operadores esperada).

Pueden suponer que los valores numéricos que se usarán se restringen al rango aceptado por los tipos de datos provistos por el lenguaje de programación que elijan.

El programa, además de los errores de sintaxis, debe informar de otros errores del código (por ej. errores en tiempo de ejecución). Un ejemplo de la informe de errores pueden verlo en la sección que sigue.

3. Ejemplos

3.1. Ejemplo correcto

A continuación se da un ejempo de un programa hecho con nuestro lenguaje, la sintaxis es correcta. Notar que se incluye comentarios de código:

```
function fact(n) {
    if n==0 || n==1 then
        return 1
    i = 1
    while n > 0 {
        i = i * n
        n = n - 1
    }
    return i
}

function sin(x) {
    x2 = x*x
    x3 = x*x2
    x5 = x3*x2
    x7 = x5*x2
    x9 = x7*x2
    return x - x3/fact(3) + x5/fact(5) - x7/fact(7) +
    x9/fact(9) /* primeros 5 terminos no nulos del polinomio de Taylor */
}

function id(x)
    return x

plot (id(x), sin(x)) for x=-pi..0.1..pi
```

En este ejemplo se definieron dos funciones: fact y sin. Podemos definir cuantas funciones queramos, la función a graficar se utilizará en la última instrucción de nuestro programa. En el ejemplo se plotea la función seno con la línea: plot (id(x), sin(x)) for x=-pi..0.1..pi

3.2. Ejemplos con errores

A continuación se detallan algunos premisas que se deben cumplir y bloques de código que no los satisfacen con mensajes de error explicativos. No es necesario que los errores tengan la misma descripción que la de los ejemplos, sí lo es que sean explicativos y que empiecen con la palabra *error*.

- Toda función debe devolver un valor y terminar con un return:

```
funcion fact(n){
}
ERROR: la funcion 'fact' no devuelve nada
```

- No escribir más de una función con el mismo nombre:

```
funcion fact(n){
    /*algun codigo*/
}
funcion fact(n){
    /*algun codigo*/
}
ERROR: la funcion 'fact' esta definida dos veces
```

- Tener cuidado con los tipos de datos:

```
funcion fact(n){
    if n then
    /*algun codigo*/
}
ERROR: el condicional del if no es un booleano
```

En la página de la materia habŕa un conjunto de tests. Estos cumplen una doble función: hacer un test no exhaustivo de la solución y complementar la especificación del lenguaje, presentada más arriba.

Además de errores sintácticos se testeará la precedencia de algunas operaciones, por ejemplo la expresión -3^2 debe evaluarse como -(3^2) y no como (-3)^2. Un problema similar deben resolver para el caso de expresiones aritméticas anidadas e ifs anidados.

Es requisito para aprobar que el programa pase los tests, pero éstos no son exhaustivos, por lo tanto no se confíen y hagan sus propios test, de considerarlos necesarios.

4. Clases a implementar

A grandes rasgos la implementación deberá contar con un *analizador léxico*, la implementación de un *parser* y la asignación de semántica. A continuación se detallan estos tres componentes:

- El analizador léxico recibe el código fuente donde están escritas las funciones disponibles para graficar y el llamado a la función plot. Este analizador se encarga de tokenizar el código fuente y enviarle al parser una lista de tokens. Se puede hacer a mano usando expresiones regulares apropiadas o una herramienta automática, por ejemplo Flex.
- El parser toma los tokens y verifica si el código fuente es acorde a la gramática definida. Se puede implementar alguno de los parsers explicados en clase o usar un generador de parsers automático (ver más abajo los generadores de parsers permitidos).
- El resultado de parsear la lista de tokens es una estructura llamada AST (Abstract Syntax Tree). Esta estructura es una representación de las funciones declaradas en el código fuente y debe permitir evaluar el código que representa. Por último, luego de evaluar la función, deben graficar los puntos obtenidos usando GNUPlot.

Toda la entrada y salida debe manejarse por stdin y stdout. Opcionalmente, podrá hacerse *además* a través de archivos.

4.1. Generadores de parser y lenguajes aceptados para el tp

Para implementar la solución pueden utilizar los generadores de parser BISON o ANTLR. Los lenguajes de programación a utilizar son C, Java o C++.

Si quieren usar otro lenguaje o generador de parsers, consúltenlo con los docentes.

5. Preguntas a responder

A continuación algunas preguntas a responder en el informe:

- Si queremos que las condiciones (las cláusulas de los if y los while) sean booleanos solamente ¿Cómo podemos verificar esto estáticamente? ¿Cómo incide en la gramática? ¿Cómo lo resuelven otros lenguajes de programación?
- ¿Por qué no hacen falta terminadores de sentencia (ej .';') como en C/C++? Expliquen por qué hacen falta en esos lenguajes y por qué no en nuestro caso? Justifiquen.
- Si quisiéramos que no importe el orden en que están definidas las funciones dentro del código, ¿Cómo lo haríamos? ¿Y para soportar recursión?

6. Detalles de la entrega

Deben entregar un informe impreso que incluya:

- breve introducción al problema a resolver,
- descripción del problema resuelto con decisiones que hayan tenido que tomar y justificación de las mismas,
- el resultado (los puntos y los gráficos) de tres funciones. Entre todas deben utilizar todos los recursos del lenguaje y deben mostrar las posibles ambiguedades y su resolución,
- la gramática obtenida a partir del enunciado y las transformaciones de la misma, que hayan sido necesarias para la implementación -explicándolas y justificándolas-,
- como mínimo tres ejemplos del programa funcionando correctamente y tres incorrectamente,
- respuesta a preguntas planteadas en la Sección 5,

- un pequeño manual del programa que detalle: la arquitectura del programa (diagrama de clases implementadas y un ejemplo de ejecución con un diagrama de secuencias), modo de uso y las opciones que acepta con requerimientos para compilar y ejecutar,
- Conclusiones.

Es parte de lo que se espera de la resolución del trabajo práctico la detección de puntos no especificados en el TP y su resolución. Pueden realizar consultas al respecto.

Si utilizaron un generador de parsers, incluyan en el informe sólo el código que no fue generado por el mismo. Si lo programaron a mano, incluyan en el informe sólo las funciones más relevantes.

Por último, enviar el código de la solución y el informe en un archivo comprimido a la dirección tptleng@gmail.com.

Fecha de entrega: Miércoles 2 de Julio

A. Polinomio de Pochammer

El polinomio de Pochammer es una función interesante ya que esta directamente relacionada con la cuestion P vs NP. Se escribe de la siguiente manera:

$$P(X) = \prod_{0 \leq j \leq 2^n} (X - j)$$

No se conoce un método eficiente para evaluar este polinomio y tampoco se ha logrado demostrar que no existe tal procedimiento. Si existiera una forma de evaluar Pochammer en tiempo polinomial entonces P=NP, por eso se intenta demostrar que es difícil de evaluar. Para ver que pinta tiene este polinomio pueden implementar la productoria como un while, pero tengan cuidado si eligen un n grande.