UNIVERSIDAD DE PUERTO RICO MAYAGUEZ

DEPARTMENTO DE INGENIERIA ELECTRICA Y DE COMPUTADORAS

POLYNOMIAL LAPLACE DOMINION CONVERTER (PLDC)

FASE 3: Reporte Final

CURSO: Programming Languages (ICOM 4036)

PROFESOR: Wilson Rivera PhD.

EQUIPO:
Laura Rivera
Melvin Vega
Velcy R. Palomino

Introducción

Polynomial Laplace Dominion Converter (PLDC), es un lenguaje de programación creado para encontrar la transformada de Laplace de polinomios de cualquier grado. La creación de este lenguaje es motivada debido a que la transformada de Laplace es utilizada en muchas aplicaciones, como es el caso de Circuitos; un tema bastante usado en el área de Ingeniería Eléctrica. Una característica importante de PLDC es que es muy lógico en cuanto a la forma en que acepta las expresiones polinomiales, ya que el usuario puede ingresar su expresión tal como se escribe el polinomio, sin preocuparse del grado, o si el polinomio está con términos completos, la única restricción es que el polinomio debe estar en forma descendente.

El objetivo principal de PLDC es que el usuario pueda utilizarlo en diferentes problemas de aplicación que requieren el uso de la transformada de Laplace de un polinomio, sin preocuparse en encontrar la transformada de Laplace manualmente, además que puede servir como herramienta de comprobación para aquellos que están aprendiendo a calcular la Transformada de Laplace. Por este motivo el presente documento es desarrollado con el propósito de proveer un tutorial y un manual de referencia sobre el uso de PLDC. Además de mostrar información sobre cómo fue desarrollado el lenguaje. Finalmente presentamos las conclusiones en cuanto a la implementación de PLDC.

Tutorial del lenguaje PLDC

1. Modo interactivo

Usamos el interprete en modo interactivo, esto es; cuando los comandos son leídos desde la consola. Una vez estamos en modo interactivo en la consola aparecerá el comando "**(+_+)>**" indicando que podemos ingresar nuestras instrucciones.

2. Operación de asignación para un polinomio:

El signo "=" es usado para asignar un polinomio a una variable. Por ejemplo, si deseamos que PLDC reciba un polinomio, este debe ser asignado a una variable (con cualquier nombre) como se muestra en los siguientes ejemplos.

- Asignando $3t^4 - 2t^2 + 5$

  ```
  (+_+)> var1= 3t^4-2t^2+5
  ```

- Asignando $t^7 + 5$

  ```
  (+_+)> poli = t^7+5
  ```

3. Función Laplace:

La Transformada de Laplace puede ser calculada de dos maneras.

Ejemplo: Deseamos calcular la Transformada de Laplace del polinomio $3t^4 - 2t^2 + 5$

- Para esto usamos la palabra *laplace* seguido del *polinomio entre paréntesis*; esto es:

```
(+_+) > laplace(3t^4-2t^2+5)
5.0/s-4.0/s^3+72.0/s^5
```

- Esta misma operación se puede hacer usando *laplace espacio* y el *polinomio*; esto es:

```
(+_+) > laplace 3t^4-2t^2+5
5.0/s-4.0/s^3+72.0/s^5
```

4. Función *laplace* de un polinomio asignado:

Si deseamos calcular la transformada de *laplace* al polinomio asignado en una variable.

- Primero asignamos el polinomio a una variable.

```
(+_+) > var=3t^2-4
```

- Ahora calculamos la transformada de *laplace* de *var*

```
(+_+) > laplace(var)
-4.0/s+6.0/s^3
```

5. Almacenar el resultado de la Transformada de Laplace en una variable.

Ejemplos:

```
(+_+) > result=laplace(var)
(+_+) > result2=laplace(3t^4-1)
```

6. Asignar una variable a otra variable:

Si deseamos asignar el polinomio almacenado en *var* a una nueva variable.

```
(+_+) > nuevaAsig=var
```

7. La función show.

Es usada para mostrar el contenido de una variable que fue asignada anteriormente.

Ejemplos:

- Deseamos mostrar lo que hay en result2 (que fue asignado previamente):

```
(+_+) > show result2
-1.0/s+72.0/s^5
```

- Deseamos ver el contenido de una variable que guardó el contenido de otra variable; es decir:

```
(+_+) > var=3t^2-4        #almacenamos un polinomio en var
(+_+) > nuevaAsig=var     #nuevaAsig guarda el contenido de
                          #var
#Ahora mostramos con show lo que contiene nuevaAsig
(+_+) > show nuevaAsig
3t^2-4
```

- También podemos imprimir un *string* con *show*

```
(+_+) > show "This is a String"
This is a String
```

Manual de referencia:

var=polinomio	Asigna el polinomio a *var*
laplace(polinomio)	Calcula la Transformada de Laplace de un polinomio
laplace polinomio	Calcula la Transformada de Laplace de un polinomio
resultado=laplace(polinomio)	Guarda la Transformada de Laplace del polinomio en resultado
show var	Muestra (imprime) el contenido de *var.*

Desarrollo del lenguaje

1. **Arquitectura del traductor**

   La implementación de PLDC fue hecha en *Python*. El primer paso fue establecer los *tokens*. Una vez definido los *tokens*, se procedió a construir el *lexer*. Para el desarrollo del *lexer* y *parser* se utilizó la librería PLY de *Python*, al importar *ply.lex* se construye el *lexer*.
   Posteriormente se prosigue con la implementación de la función *coef*, que se encarga de extraer los coeficientes del polinomio, para luego pasarlos a la clase *Polynomial*, quien será la encargada de calcular la *Transformada de Laplace* del polinomio.

2. Interfaces entre los módulos

Módulo: lex.py

Este módulo se basa en reglas de expresiones regulares. Para utilizarlo, se debe escribir las expresiones regulares para cada *token*, como las mostradas a continuación.

```
import ply.lex as lex
 tokens = ('LAPLACE',
          'SHOW',
          'NUMBER',
          'PLUS',
          'TIMES')

# reserved words
reserved = {
            'show':'SHOW',
            'laplace':'LAPLACE',
}

# Regular expression rules for simple tokens
t_PLUS    = r'\+'
t_MINUS   = r'-'
t_TIMES   = r'\*'
t_LPAREN  = r'\('

# A regular expression rule with some action code
def t_NUMBER(t):
    #r'-?\d+'
    r'\d+'
    t.value = int(t.value)
    return t
```

Módulo Yacc

Este módulo permite crear un programa que tome los "tokens" como entrada y que los reconozca como un lenguaje. Para ello usa la gramática BNF que describe el ensamblaje de los tokens.

Por cada regla se define una función que recibe como parámetro un objeto, como en una especie de lista, pero no se comporta como tal; este objeto contiene los valores de cada símbolo de la regla. Algunos de estos son:

```
def p_langfunctions(p):
    'langfunction : polynomial'
    p[0] = str(p[1])

def p_langfunctions1(p):
    'langfunction : function'
    p[0] = str(p[1])
```

```
import ply.yacc as yacc
yacc.yacc()
```

3. **Entorno de desarrollo utilizado para crear el traductor**
   Para el desarrollo de PLDC se utilizó Eclipse con el plugin de PyDev. PyDev es un entorno de desarrollo para el lenguaje Python en Eclipse.

4. **Metodologías de pruebas**

   Para probar PLDC durante su desarrollo, se usaron dos métodos:

   a) **Usando el modo interactivo:** Significa probar cada instrucción desde consola. Por ejemplo, si deseábamos probar la función *laplace* desde consola se escribe:

   ```
   (+_+) > laplace(3t^4-2t^2+5)
   5.0/s-4.0/s^3+72.0/s^5
   ```

   b) **Leyendo un archivo:** Si deseamos probar todas las funciones, para que no sea tedioso ingresar uno a uno cada instrucción, podemos leer un archivo que contenga todas las posibles instrucciones, y los resultados se muestran por consola.

   Ejemplo al leer "prueba.txt"

   ```
   varnam = 3t^2+t+6
   show varnam
   laplace(varnam)
   laplace(3t^2+t+6)
   res1=laplace(t^3-6)
   show res1
   res2=laplace 7t^5+6
   show res2
   other=varnam
   show ("This is a polynomial", varnam)

   #Para leer las instrucciones desde un file
   file = open('prueba.txt', 'r')
   ## Mostramos por pantalla lo que leemos desde el fichero
   for line in file:
        yacc.parse(line)
   ## Cerramos el fichero.
   file.close()
   #El resultado es:
   3t^2+t+6
   6.0/s+1.0/s^2+6.0/s^3
   -6.0/s+6.0/s^4
   6.0/s+840.0/s^6
   3t^2+t+6
   ```

Conclusión:

El desarrollo de este proyecto fue muy importante, porque nos permitió comprender el proceso que implica la creación de un lenguaje de programación. Y a la misma vez logramos entender de mejor manera los pasos que se deben seguir a la hora de crear un lenguaje. Este proyecto también nos enseñó a valorar más los lenguajes ya creados, porque nos imaginamos el trabajo que pasaron para crear lenguajes tan bien estructurados.

PLDC es un lenguaje de uso sencillo, con el cual podemos encontrar la transformada de Laplace de un polinomio, ya sea aplicando *laplace* directamente al polinomio o a uno almacenado en una variable, esto significa que PLDC nos permite asignar una variable a un polinomio, así como también; almacenar el resultado de la transformada en una variable; los resultados almacenados en alguna variable pueden ser mostrados con el uso del comando *show,* este comando permite a su vez imprimir cualquier "*string*".

Es importante resaltar que PLDC es aplicable a un polinomio de cualquier grado y sin importar la variable usada en el polinomio. Sin embargo; para un futuro requiere ser provisto de otras operaciones como es; la transformada inversa de Laplace, así mismo PLDC puede ser extendida para mas expresiones adicionales a polinomios.