Campus Monterrey (ITESM)

Nombre del trabajo:

Tarea 2

Curso:

Desarrollo de aplicaciones avanzadas de ciencias computacionales

Estudiante:

Sergio Tomás Vargas Villarreal | A00837196

Fecha de entrega:

3 de octubre del 2025

Lex & Yacc
Lex y Yacc son herramientas clásicas de C para crear analizadores léxicos y sintácticos. Lex utiliza *expresiones regulares* y *autómatas finitos deterministas* para crear el scanner que reconoce los tokens, mientras que Yacc emplea reglas *BNF* (básicamente una notación formal para describir la sintaxis de lenguajes) para construir el parser mediante algoritmos *LALR(1)*. Este parser es eficiente, lee la entrada de izquierda a derecha y usa un token de anticipación. Ambas herramientas son multiplataforma, generan código C y permiten insertar bloques de código personalizado en archivos .l y .y.

Flex & Bison
Flex y Bison son las versiones modernas y mejoradas de Lex y Yacc, desarrolladas como implementaciones libres bajo la licencia GPL. En cuanto a funcionamiento cumplen exactamente el mismo rol. Flex reemplaza a Lex para el análisis léxico con expresiones regulares y autómatas, y Bison sustituye a Yacc para el análisis sintáctico con reglas BNF y algoritmos LALR(1). Su principal *diferencia* está en la eficiencia, la portabilidad y e*l soporte actualizado* que ofrecen, lo que las convierte en las herramientas más utilizadas en la actualidad para la generación de analizadores en C/C++.

PLY
PLY es una implementación en Python inspirada en Lex y Yacc. Dicha herramienta permite escribir analizadores haciendo uso de *expresiones regular*es para el léxico y gramáticas estilo *BNF* en forma de funciones Python. Se ejecuta de manera interpretada en Python, es gratuito bajo licencia BSD y resulta ideal para proyectos pequeños o educativos. Su interfaz es muy natural para programadores de Python, dado a que basta con definir funciones con decoradores y añadir código propio dentro de ellas.

Lark
Lark es una biblioteca en Python que permite generar parsers de manera simple y multiplataforma. Acepta gramáticas en EBNF escritas como *strings* en el código y funciona de forma interpretada. Soporta varios algoritmos (LALR(1), Earley, CYK), genera árboles de parsing automáticamente y no requiere separar lexer y parser. Es gratuita (licencia MIT).

Ejemplo:

```
import ply.lex as lex, ply.yacc as yacc
tokens = ('NUMBER', 'PLUS')
t_PLUS = r'\+'
def t_NUMBER(t): r'\d+'; t.value = int(t.value); return t
lexer = lex.lex()

def p_sum(p): 'expr : expr PLUS NUMBER'; p[0] = p[1] + p[3]
def p_number(p): 'expr : NUMBER'; p[0] = p[1]
parser = yacc.yacc()

print(parser.parse("2 + 3 + 5"))
```

Levine, J. (2009). flex & byson. O'Reilly Media. https://web.iitd.ac.in/~sumeet/flex__bison.pdf
GeeksforGeeks. (2022, February 16). PLY (Python LexYACC) an Introduction. GeeksforGeeks. https://www.geeksforgeeks.org/python/ply-python-lex-yacc-an-introduction/
Levine John. (1990). lex & jacc. https://d1.amobbs.com/bbs_upload782111/files_33/ourdev_584393GCYRF3.pdf
Lark-Parser. (n.d.). GitHub - lark-parser/lark. GitHub. https://github.com/lark-parser/lark