Práctica 1

Compiladores (CS3402) 2021-0

Integrantes:

- Mayra Díaz Tramontana 201910147
- Joaquín Elías Ramírez Gutiérrez 201910277

Ejercicio 1:

¿Qué cadenas genera la gramática con la regla $A \rightarrow (A)A|\epsilon$? Formule la derivación de la cadena (()(()))()

Genera expresiones de paréntesis balanceados.

(()(()))() m.i.

A → (A)A → ((A)A)A → (()A)A → (()(A)A)A → (()((A)A)A)A → (()(()A)A)A → (()(()))A → (()(()))(A)A →→ (()(()))()A → (()(()))()

Ejercicio 2:

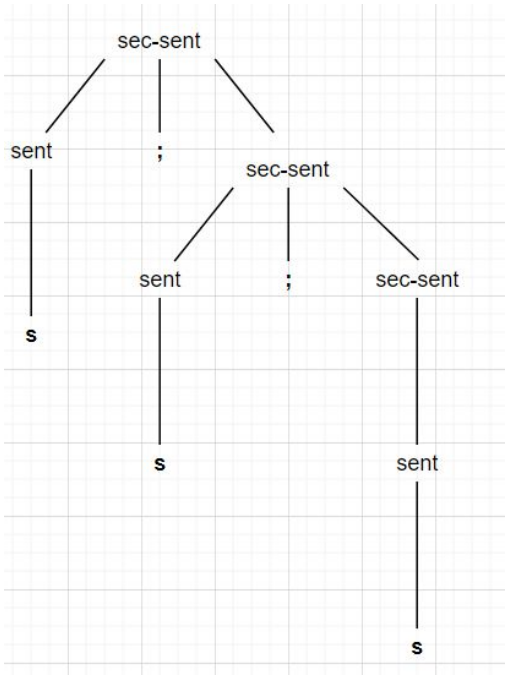
¿Qué cadenas genera la gramática ?
sec-sent → sent; sec-sent | sent
sent → s
Genere el árbol de análisis gramatical y abstracto para la cadena **s;s;s**,
¿Cómo lograría que hayan secuencias de sentencias vacías?

Genera una secuencia de 's' separadas por ';' (la última no lleva ';' al final).

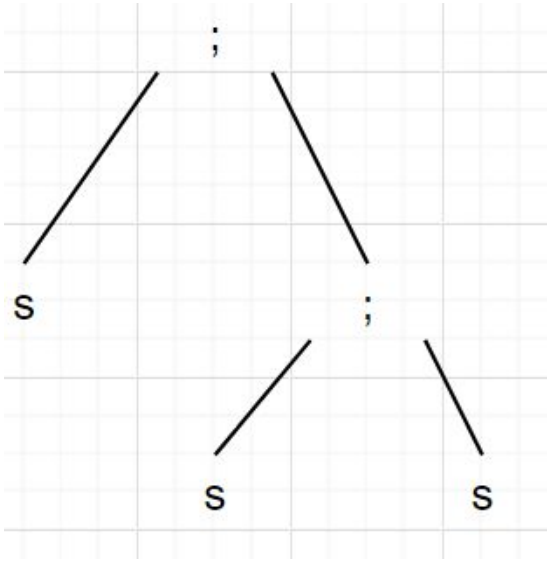
s;s;s m.i.

sec-sent → sent; sec-sent → s; sec-sent → s; sent; sec-sent → s;s; sec-sent → s;s;sent → s;s;s

Gramatical:



Abstracto:



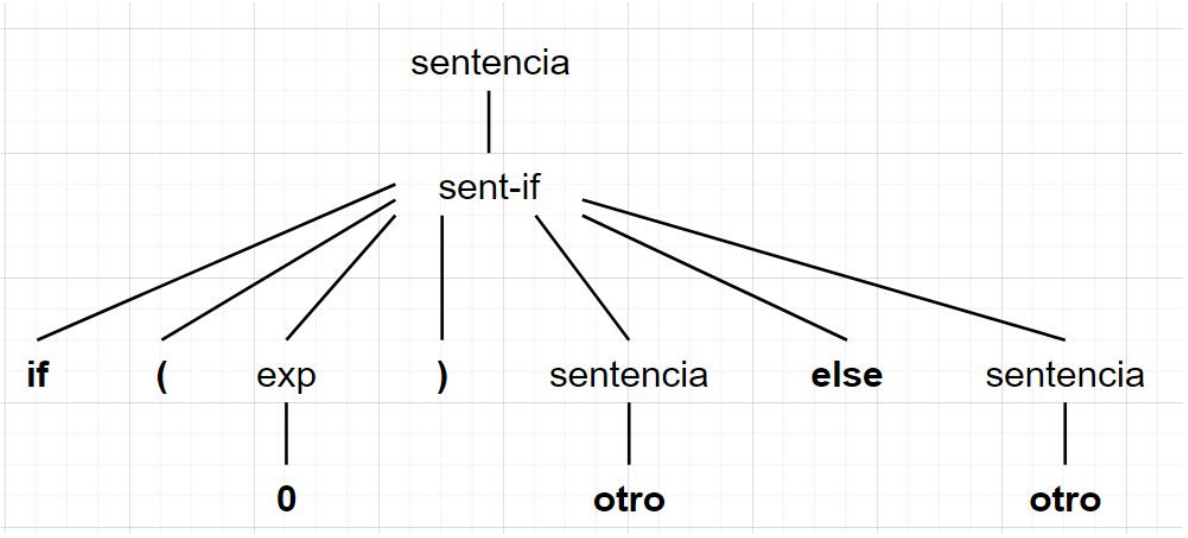
Ejercicio 3:

Genere el árbol de análisis gramatical y abstracto para la cadena **if (0) otro else otro**

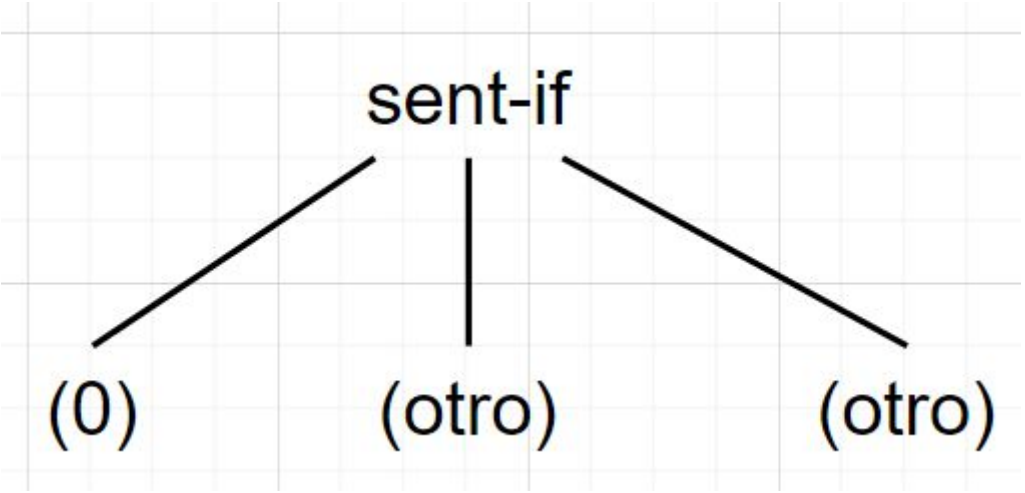
según la gramática:

sentencia → sent-if | otro
sent-if → if (exp) sentencia | if (exp) sentencia else sentencia
exp → 0|1

Gramatical:



Abstracto:



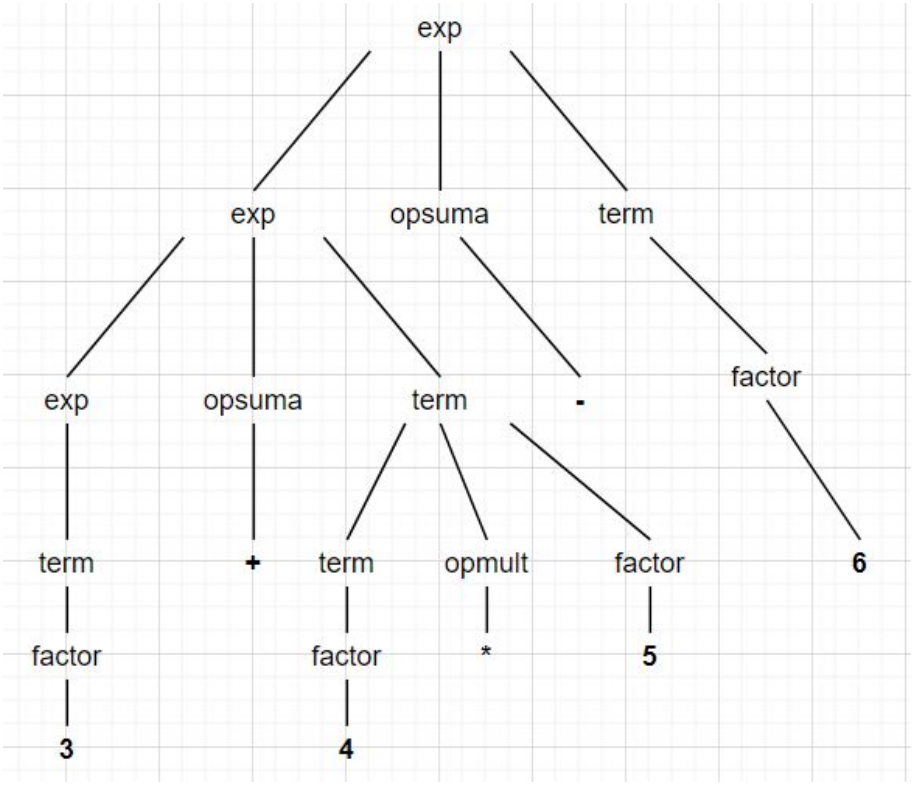
Ejercicio 4:

Genere derivaciones por la izquierda, árboles de análisis gramatica
abstractos según la gramática:
exp → exp opsuma term | term
opsuma →+ | -
term → term opmult factor | factor
opmult → *
factor → (exp) | numero
Para
a) $3 + 4 * 5 - 6$
b) $3 * (4 - 5 + 6)$
c) $3 - (4 + 5 * 6)$

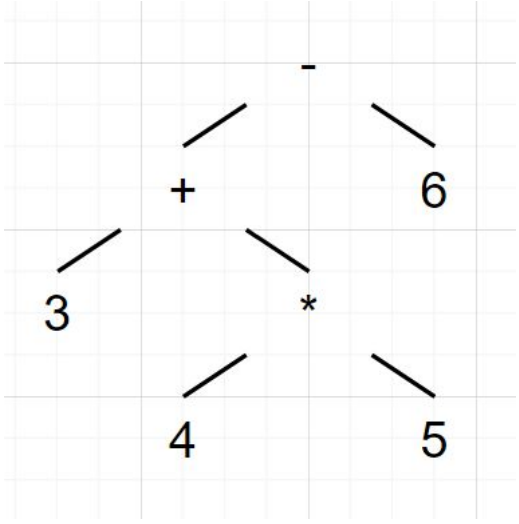
a) 3 + 4 * 5 - 6

m.i.
exp →
exp opsuma term →
exp opsuma term opsuma term →
term opsuma term opsuma term →
factor opsuma term opsuma term →
3 opsuma term opsuma term →
3 + term opsuma term →
3 + term opmult factor opsuma term →
3 + factor opmult factor opsuma term →
3 + 4 opmult factor opsuma term →
3 + 4 * factor opsuma term →
3 + 4 * 5 opsuma term → 3 + 4 * 5 - term → 3 + 4 * 5 - factor → 3 + 4 * 5 - 6

Gramatical:



Abstracto:

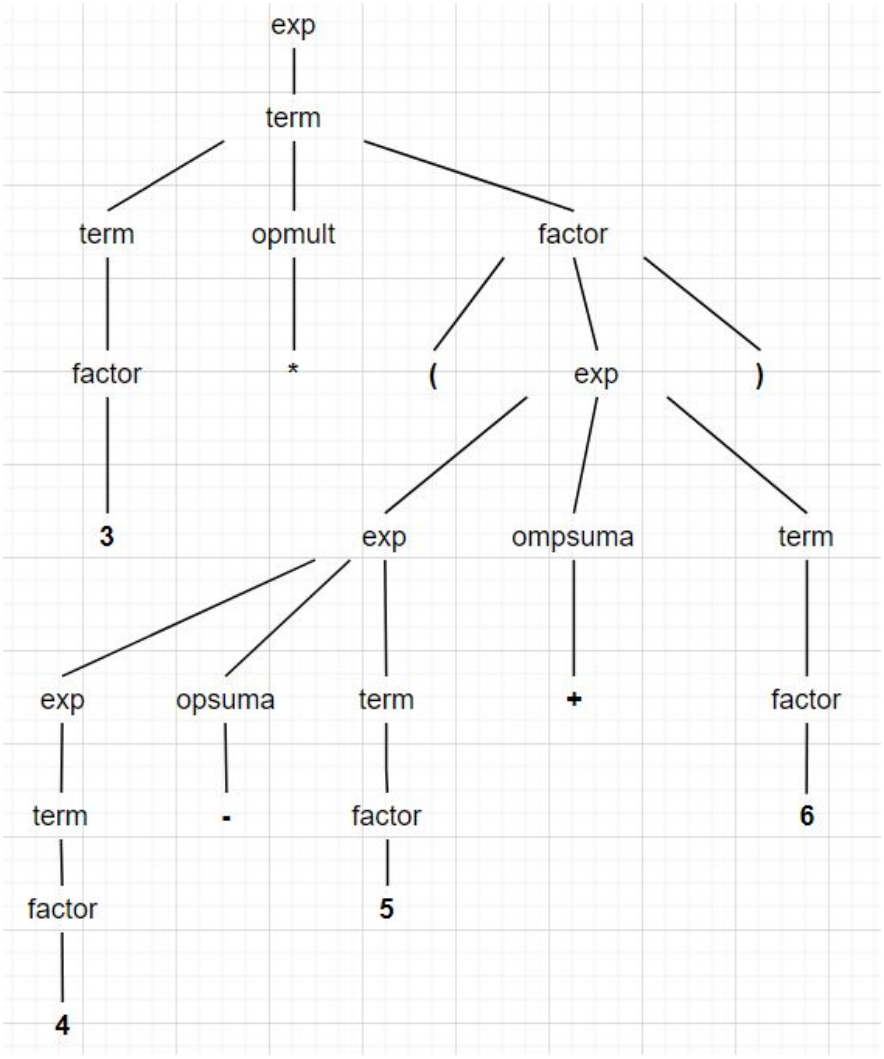


b) 3 * (4 - 5 + 6)

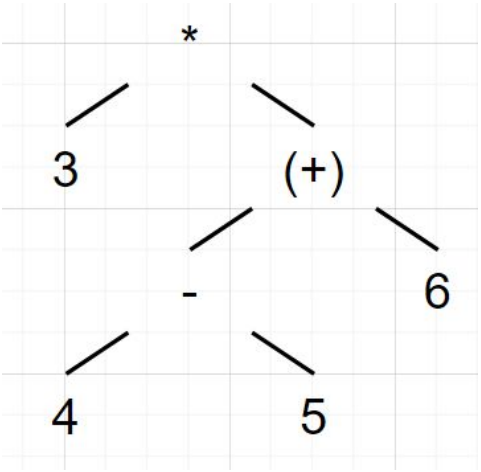
m.i.

exp → term → term opmult factor → factor opmult factor → 3 opmult factor → 3 * factor → 3 * (exp) → 3 * (exp opsuma term) → 3 * (exp opsuma term opsuma term) → 3 * (term opsuma term opsuma term) → 3 * (factor opsuma term opsuma term) → 3 * (4 opsuma term opsuma term) → 3 * (4 - term opsuma term) → 3 * (4 - factor opsuma term) → 3 * (4 - 5 opsuma term) → 3 * (4 - 5 + term) → 3 * (4 - 5 + factor) → 3 * (4 - 5 + 6)

Gramatical:



Abstracto:

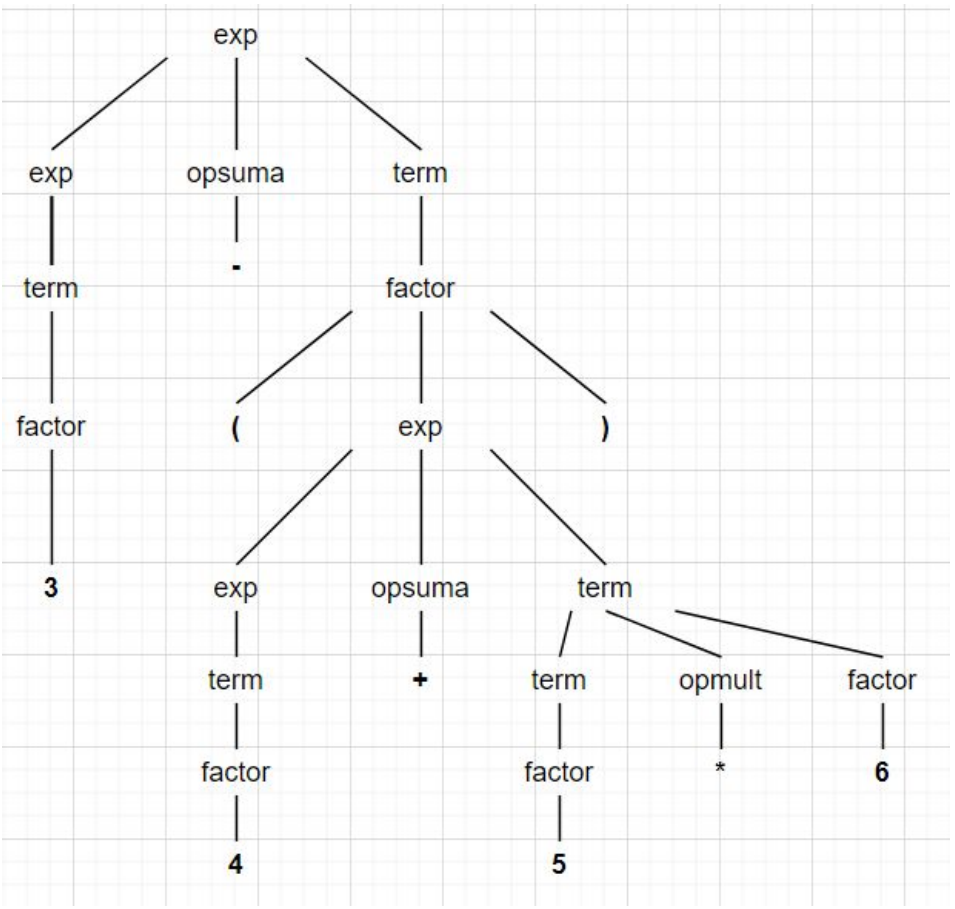


c) 3 - (4 + 5 * 6)

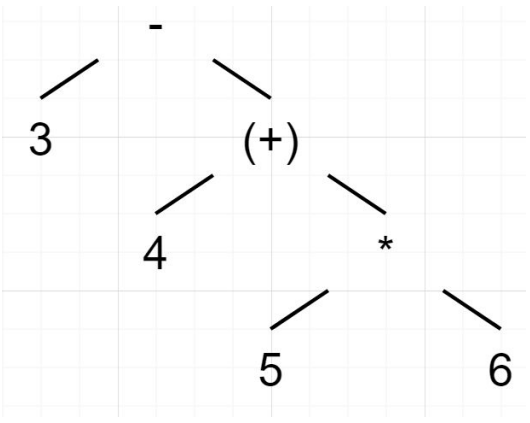
m.i.

exp →
exp opsuma term → term opsuma term → factor opsuma term →
3 opsuma term → 3 - term → 3 - factor → 3 - (exp) → 3 - (exp opsuma term)
→ 3 - (term opsuma term) → 3 - (factor opsuma term) → 3 - (4 opsuma term)
→ 3 - (4 + term) → 3 - (4 + term opmult factor) → 3 - (4 + factor opmult factor)
→ 3 - (4 + 5 opmult factor) → 3 - (4 + 5 * factor) → 3 - (4 + 5 * 6)

Gramatical:



Abstracto:



Ejercicio 5:

Elabore un programa en C++ que permita ingresar dos numeros enteros y compararlos con la funcion assert($x < y$). Si $x < y$ se imprime un mensaje con este resultado. En caso contrario termina el programa. Permita activar/desactivar assert() utilizando la directiva de preprocesamiento NDEBUG. Si esta definida, assert se desactiva. En caso contrario, se activa. Decida si definir (o no) NDEBUG al momento de compilacion.
Opcional: utilice un Makefile para elegir entre la opcion de compilacion con NDEBUG o sin NDEBUG.

Link de los archivos:
https://drive.google.com/drive/folders/1tEsYSh8eK1XuAz6CoulVJ32QK1wjrQaM?usp=sharing

ejercicio5.cpp:

```
#include <iostream>
#define assert(x,y) (x<y)

int main(int argc, char *argv[]) {
	int x, y;
	std::cin >> x >> y;
#ifndef NDEBUG
	if (assert(x, y))
		std::cout << x << " < " << y << '\n';
#endif
}
```

Makefile:

```
#make ndebug=1 ej5		// ndebug ON
#make ej5			// ndebug OFF
ej5:
ifdef ndebug
	g++ -DNDEBUG ejercicio5.cpp
else
	g++ ejercicio5.cpp
endif
clean:
	rm -r a.out
```