INSTITUTO POLITÉCNICO NACIONAL ESCUELA SUPERIOR DE CÓMPUTO

**PRACTICA 5**

“DECISIONES Y CICLOS”

ALUMNO:

ALBERTO ANDREI LOPEZ PEREZ

PROFESOR: TECLA PARRA ROBERTO

GRUPO: 5CM4

MATERIA: COMPILADORES

0

**Introducción**

En esta práctica se implementaron los conceptos base de compiladores y la implementación de YACC, donde se da continuidad a la práctica número 4, la cual fue una una "Calculadora de Vectores con Máquina Virtual de Pila". Lo que nos atañe realizar en esta práctica es:

Agregar decisiones ( if ) y ciclos ( while )

Donde, ya con la ayuda de la implementación de la *máquina virtual de pila*, ahora es más sencillo agregar este apartado en donde se busca que se puedan generar *decisiones* y *ciclos* con base en condiciones.

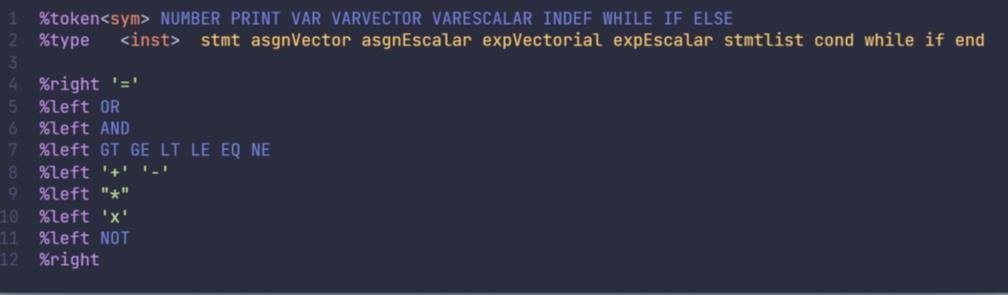
Un ejemplo de *decisiones* en nuestra calculadora de vectores sería poder comparar dos vectores o dos escalares o uno y uno para, con base en el resultado de esa comparación, asignar un valor a una variable o imprimir un vector en específico.

Un ejemplo de *ciclos* en nuestra calculadora de vectores sería poder, con base en una condición de comparación o lógica, poder repetir una impresión de algo o una asignación de una variable.

Para poder realizar lo anterior se nos proporciona el código del **HOC5.y** visto en clase donde se implementa precisamente *decisiones y ciclos* en una calculadora normal*,* por lo que con ayuda de ese nos apoyamos para construir nuestra propia implementación acomodada al contexto de la "Calculadora de Vectores".

# Desarrollo

Como habíamos comentado en la práctica pasada, el beneficio de implementar una *máquina virtual de pila* es **poder escalar de forma muy sencilla su funcionamiento**, ya sea por medio de la creación de nuevas reglas y nuevas instrucciones para la máquina.

Es por ello que de lo primero que hicimos fue definir los nuevos terminales y no terminales en nuestra sección de definiciones de YACC, donde agregamos los no terminales necesarios para crear los "statements" como lo son el crear la sentencia "if", "if-else", y "while", todo junto con lo que requieren sus producciones. De igual manera agregamos los terminales que corresponden a los operadores de comparación y los lógicos junto con su precedencia y asociatividad.

Nuevos terminales y no terminales en la sección de definiciones de YACC

Después agregamos las nuevas producciones en la parte de Reglas de YACC, las cuales se podría decir no llegaron a tener tantos cambios a los que ya habíamos visto previamente en el **HOC5.y** proporcionado por el profesor donde cada una de ellas va ingresando a la RAM lo necesario para poder esta ser ejecutada. Sin embargo, es digno destacar que:

Los **stmt** pueden llegar a ser asignaciones de cualquier tipo, ya sean vectoriales y escalares.

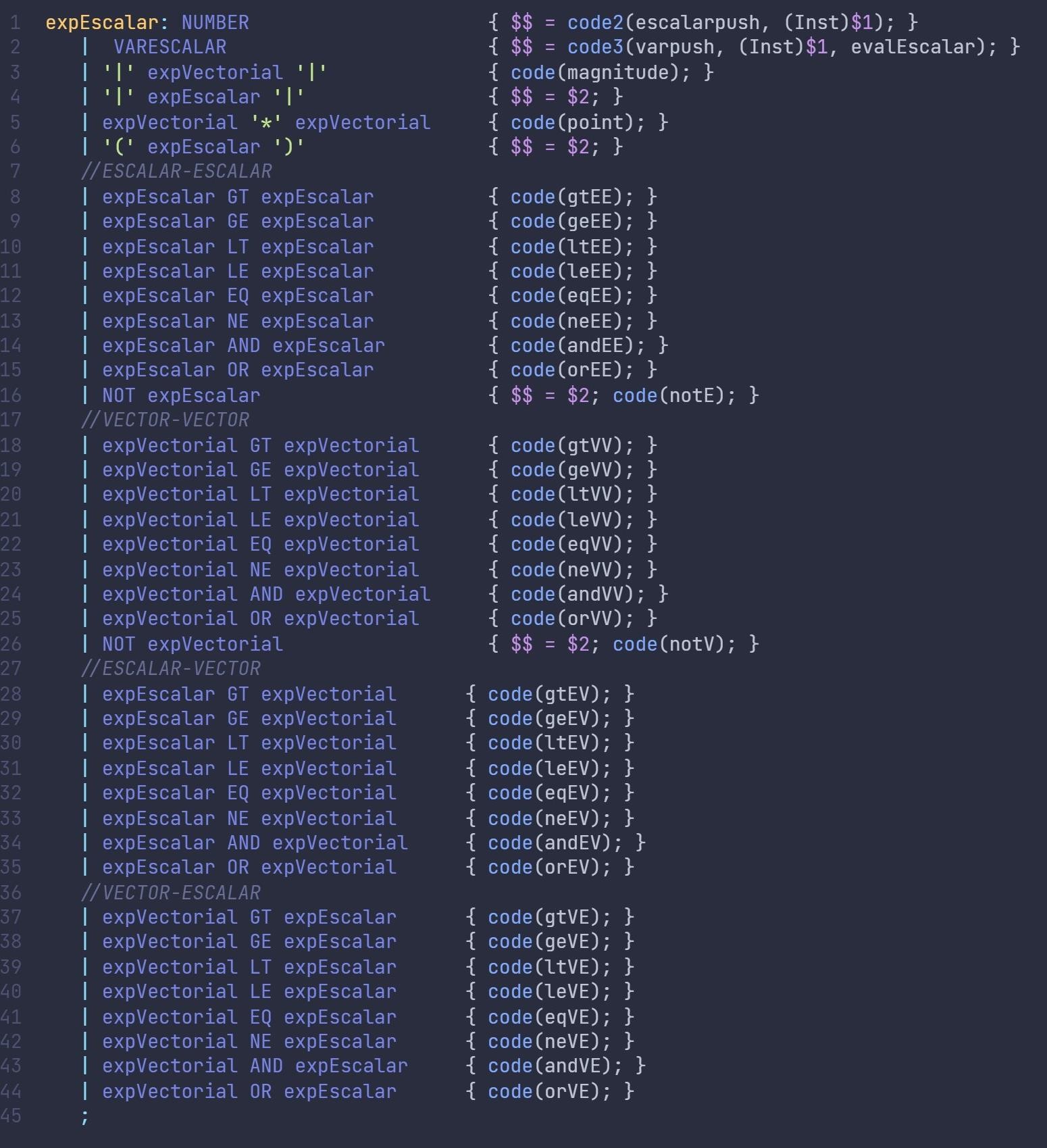
Lo que se puede llegar a imprimir en un **stmt** puede ser un vector o un escalar.

La condición que se evalúa ya sea en una sentencia if o while únicamente puede ser una expresión escalar, ya que el resultado de hacer cualquier operación lógica o de comparación siempre nos regresará un número o escalar representando el true o false.



Nuevas reglas de YACC para crear decisiones y ciclos.

También, una de las cosas que se agregaron fueron todas las expresiones que nos forman operaciones lógicas y de comparación, las cuales como hemos dicho, devuelven un escalar, por lo que fueron agregadas en **expEscalar**.



Nuevas reglas en expEscalar para generar operaciones de comparación y lógicas.

Como se puede observar, varias veces repetimos las operaciones, ya que pueden cambiar en cuanto sus argumentos. Por ejemplo, puede haber la comparación entre:



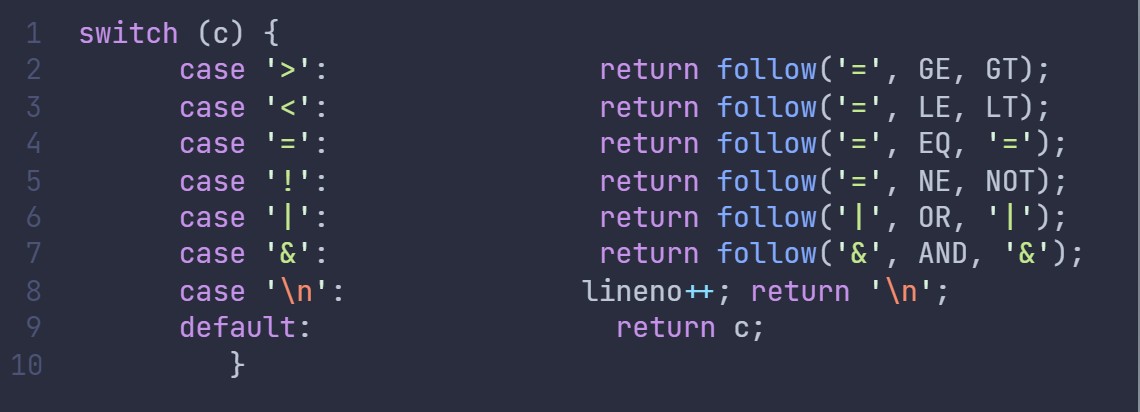
]

1 vector entre 1 escalar [0 1 0] == 1 ó 1==[0 1

0]

Todas las operaciones de comparación y lógicas son posibles ya que cuando se trata de operandos tipo vector, lo que se realiza es la obtención de su magnitud para poder

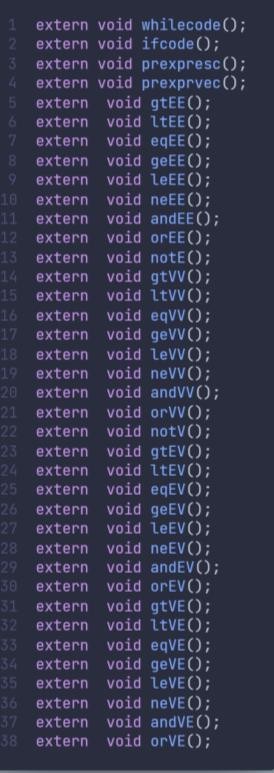
realizar correctamente las operaciones (por eso se tienen diversas instrucciones, las cuales están implementadas en **code.c**).

Para poder obtener los terminales de lógicos y de comparación fue necesario modificar el analizador léxico para que cuando aparezca alguno de ellos, se regrese el tipo de terminal correcto.

Parte del analizador léxico para observar el tipo de terminal. La función **follow()** solo ayuda a diferenciar operadores con solo un carácter como "=" a los de dos "==".

Finalmente, para poder lograr la funcionalidad se debieron agregar todas las implementaciones de las instrucciones nuevas en **code.c** y ser declaradas en **symbol.h**

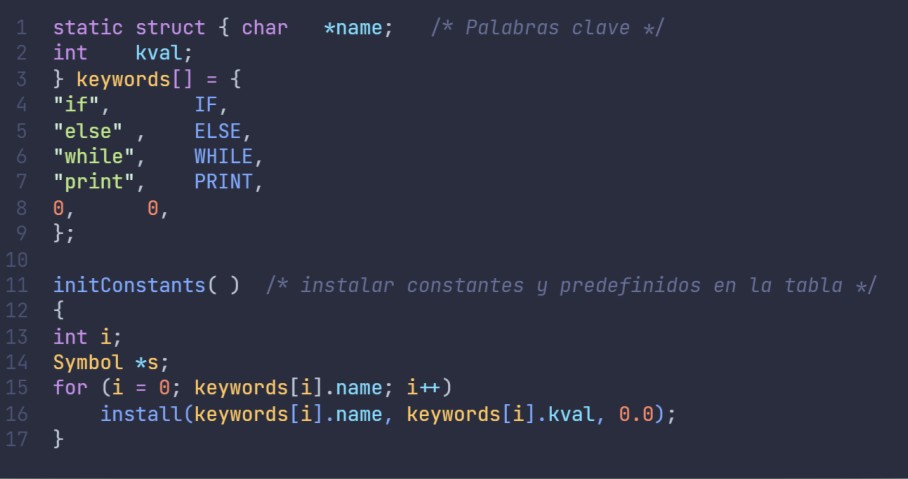
Declarando las



instrucciones externas en symbol.h

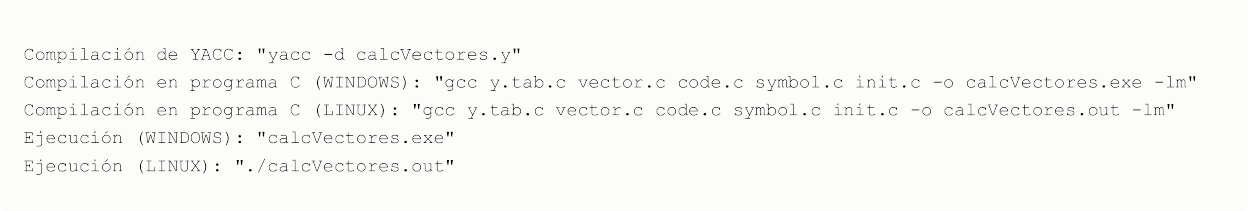
Instrucciones implementadas en code.c para generar decisiones, ciclos, operaciones lógicas y de comparación.

ℹ También algo importante para que se ejecute correctamente el programa es ingresar en la tabla de símbolos las palabras reservadas para generar las decisiones, los ciclos y la impresión. Esto lo hacemos en **init.c**



Instalación de keywords en la tabla de símbolos en init.c

# Compilación y Ejecución

Para poder compilar y ejecutar:

# Programa en Ejecución

Ejecución de if, if-else y ciclo while, usando vectores y escalares.

# Conclusión

Gracias a que en la anterior práctica creamos la máquina virtual de pila, la implementación de las decisiones y el ciclo while fue realmente sencillo. La complejidad estuvo a la hora de poder definir que es lo que vamos a utilizar como *statement* y sobre todo, como es que vamos a lograr las operaciones lógicas y de comparación, lo cual fue un tanto laborioso de resolver. También es importante entender cómo es que funcionan las acciones gramaticales de las sentencias if, if-else y while, ya que un manejo erróneo de la memoria RAM produciría un error catastrófico en el programa de la máquina virtual. Y por último, también es indispensable ver que cuando se tratan de terminales como *palabras clave* es **indispensable** instalarlas en la tabla de símbolos, ya que de otro modo se podrían tomar como variables, provocando que todas las producciones donde aparezcan se conviertan en errores de sintaxis.

# Video Explicación & Ejecución

