Icono

Descripción generada automáticamenteUn dibujo de una persona

Descripción generada automáticamente con confianza bajaINSTITUTO POLITÉCNICO NACIONALESCUELA SUPERIOR DE CÓMPUTO

**PRACTICA 3**

“TABLA DE SIMBOLOS”

ALUMNO:

LOPEZ PEREZ ALBERTO ANDREI

PROFESOR:

TECLA PARRA ROBERTO

GRUPO:

5CM4

MATERIA:

COMPILADORES

## Introducción

En esta práctica se implementaron los conceptos base de compiladores y la implementación de YACC, donde se da continuidad a la práctica número 1, la cual fue una una "Calculadora de Vectores". Lo que nos atañe realizar en esta práctica es:

Agregar una *tabla de símbolos* para permitir nombres de variables de más de una letra y agregar a la gramática la producción para el operador de asignación.

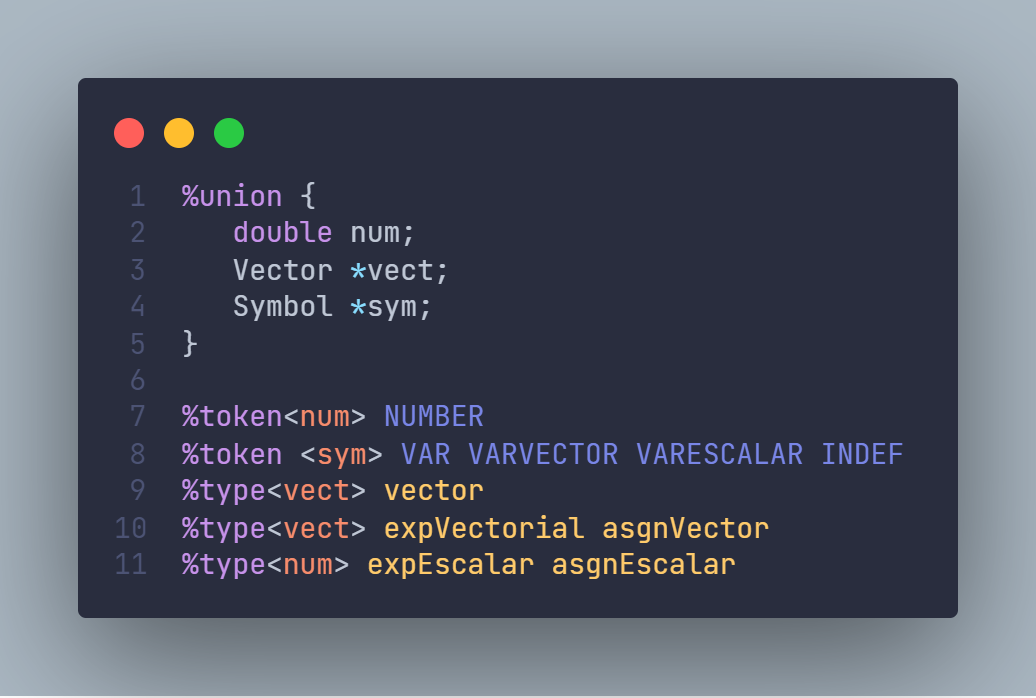
De modo que explícitamente para la calculadora de vectores se nos pide que que las variables sean capaces de guardar vectores (aunque en nuestra implementación también permitimos el alojamiento de escalares).

Para poder realizar lo anterior se nos proporciona el código del **HOC3.y** visto en clase donde se implementa precisamente una *tabla de símbolos, constantes* y *builtins,* por lo que con ayuda de ese nos apoyamos para construir nuestra propia tabla de símbolos acomodada al contexto de la "Calculadora de Vectores".

ℹ La práctica también pedía que **de ser posible** agregar *builtins*, sin embargo como la calculadora es de vectores **no hay funciones especiales** además de las ya implementadas por lo que solo se integraron algunas constantes

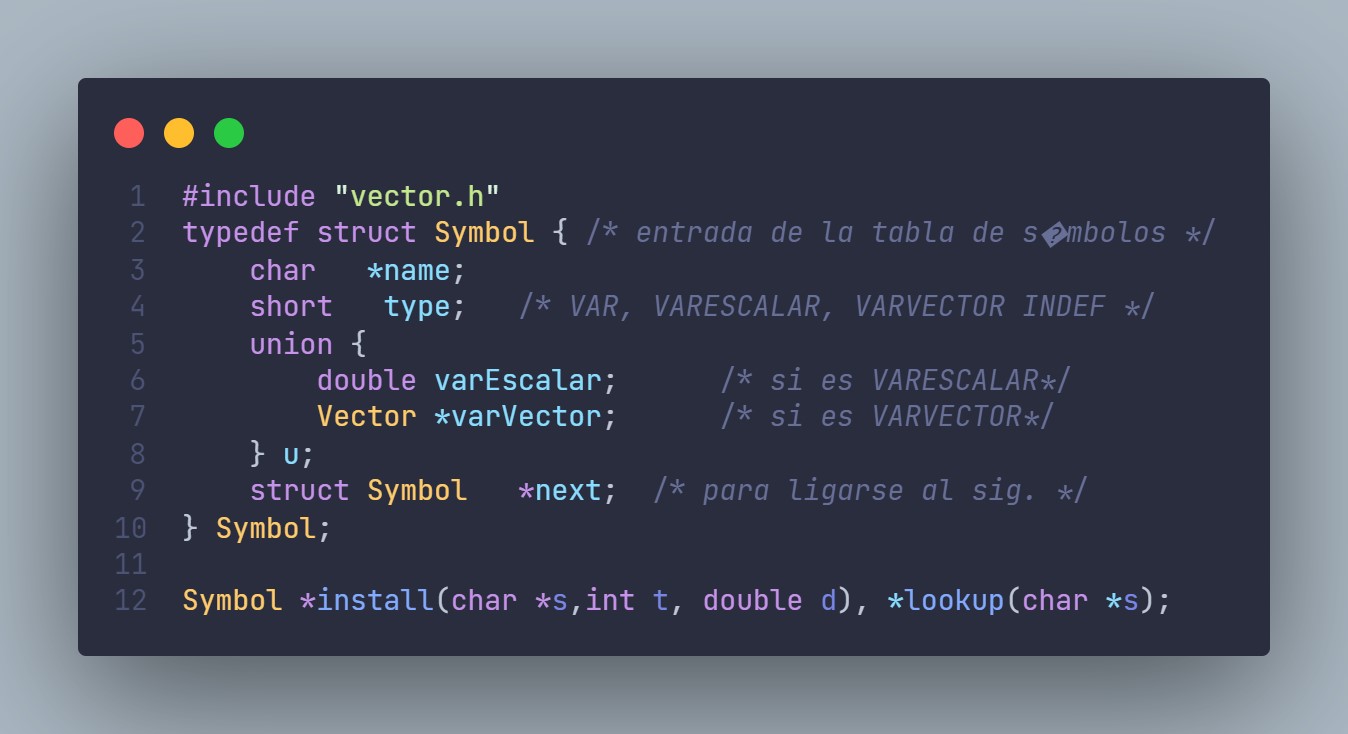
## Desarrollo

Para poder realizar esta práctica lo primero a tomar en cuenta es modificar nuestros tipos de elementos en nuestra pila de YACC modificando la unión que teníamos anteriormente, agregando una variable tipo Symbol\*la cual es el tipo de elementos en nuestra tabla de símbolos. Además se agregaron 3 terminales nuevos (**INDEF**, **VAR**, **VARESCALAR**, **VARVECTOR**) y 2 nuevos no terminales (**asgnVector** y **asgnEscalar**).



**Unión y definición de símbolos gramaticales**

Es importante mencionar que la implementación de la tabla de símbolos es una lista enlazada la cual para poder funcionar tiene dos métodos, **install()** para poder agregar los símbolos a la tabla y **lookup()** para poder obtener un símbolo de la tabla. Esta tabla y sus métodos que la manipulan estan declarados en su cabecera **symbol.h** e implementados en **symbol.c**.



**Estructura de lista enlazada (symbol como nodos) en symbol.h**

También se puede observar que cada nodo/símbolo esta compuesto de un nombre, su tipo y una unión la cual dependiendo del tipo, podrá ser un double representando una variable escalar y un Vector\* representando una variable vectorial.

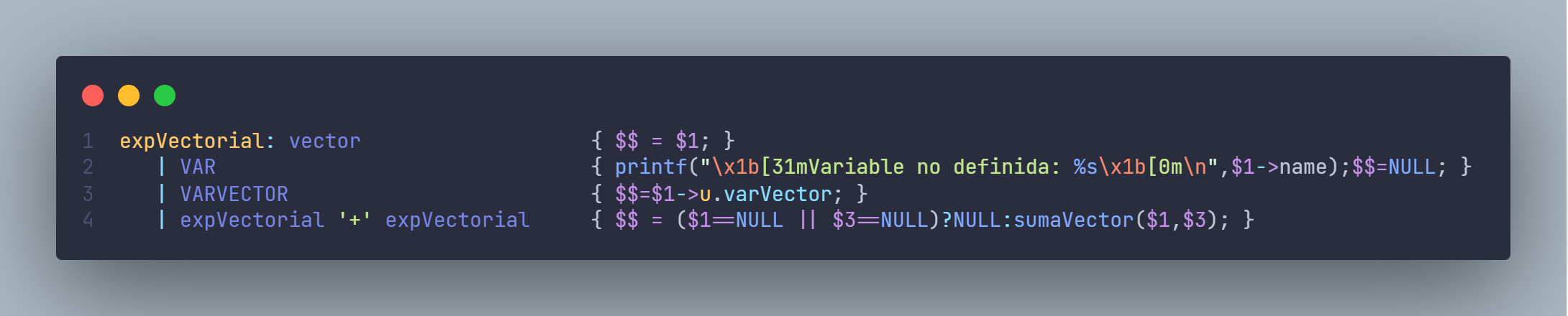
Pasando a la parte de las reglas, vemos que tenemos las asignaciones de las variables, donde si de inicio es una variable sin definir **VAR** se puede asignar un vector o un escalar

(dependiendo de si es **asgnEscalar** o **asgnVector**), aunque también damos la oportunidad de redefinir a las variables (ya sean estas anteriormente escalares o vectoriales). Cabe destacar que las declaraciones de escalares difieren con la de los vectores anteponiendo el símbolo de **'#'** a la variable.



**Producciones de asignación en la sección de reglas de calcVectores.y**

De ahí en fuera las demás producciones quedaron muy similares a las que teníamos en la anterior práctica, sin embargo agregamos que una **expEscalar** también puede ser un **VARESCALAR** y un **expVectorial** también puede ser un **VARVECTOR** y un **VAR** (para poder decirle al usuario que necesita definirla usando su acción).



Ejemplo de producciones usando los nuevos terminales de variables

Es importante mencionar que para que siga corriendo el programa cuando le ponemos una variable que no esa definida en algunas acciones usamos operadores ternarios para que, además de lanzar el mensaje de error, se pueda seguir el flujo del programa sin realizar la operación indicada en la acción (como ejemplo el **sumaVector()** que si se hiciese con una variable vacía mandaría un error y terminaría el programa).

Finalmente, algo que también se modifico fue el analizador léxico, el cual le permitimos obtener la cadena de caracteres que representa el nombre de la variable para que la busque o la instale en la tabla de símbolos, además de regresar el tipo. Es importante ver que no podemos usar el carácter **'x'** al inicio de una variable ya que como es un terminal que se usa en nuestras reglas lo omitimos.



**Analizador léxico en calcVectores.y**

Algo digno de mencionar es que en nuestro main() en **calcVectores.y** mandamos a llamar el método **initConstants()** presente en el archivo **init.c** donde metemos algunas constantes que se pueden ver a continuación. Es importante observar que incluimos a **y.tab.h** para poder hacer uso de sus macros (en específico **VARESCALAR**), por lo que para que todo funcione es necesario compilar con yacc -d nuestro archivo para que obtengamos además del **y.tab.c** su cabecera.



Inserción de constantes en la tabla de símbolos, código en init.c

### Compilación y Ejecución

Para poder compilar y ejecutar:

Compilación de YACC: "yacc -d calcVectores.y"

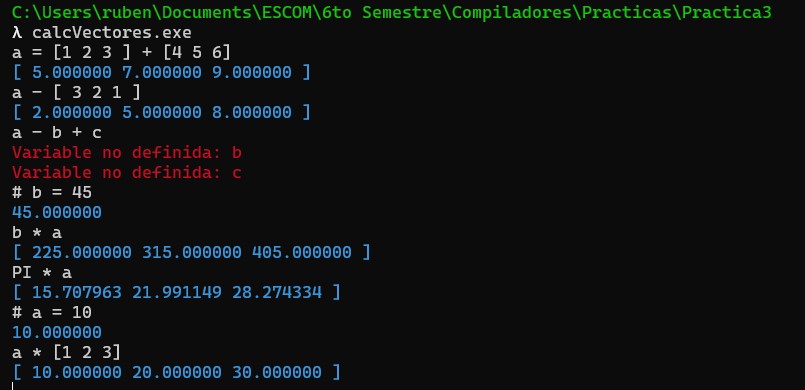
Compilación en programa C (WINDOWS): "gcc y.tab.c vector.c Pila.c symbol.c init.c -o calcVectores.exe -lm"

Compilación en programa C (LINUX): "gcc y.tab.c vector.c Pila.c symbol.c init.c -o calcVectores.out -lm"

Ejecución (WINDOWS): "calcVectores.exe"

Ejecución (LINUX): "./calcVectores.out"

### Programa en Ejecución



Ejecución de diversos ejemplos de variables escalares, vectoriales, redefiniciones y errores posibles.

## Conclusión

Realizar la tabla de símbolos puede que de inicio se torne un poco complicado si no se entiende bien como es que funcionan las uniones. Sin embargo, debido a que fue algo que se ocupó desde la práctica anterior realmente se facilitó la inclusión de un nuevo tipo. Por otro lado, donde creo que se batalló bastante para poder generar un mejor código fue en la implementación extra de tener dos tipos de variables ya que de inicio trataba de usar un mismo token para poder meter ambas, sin embargo, la división de **VARESCALAR** y

**VARVECTOR** (sobre todo en el analizador léxico) facilitó mucho las cosas y solo fue cuestión de pulir ciertos detalles en varias partes del código. Además, es importante observar que, a pesar de que se implementó esta práctica con base en el **HOC3.y**, considero que no es tan trivial pasarla a otro contexto, pero si se entiende bien como es que funcionan las cosas se puede llevar un buen manejo de todo lo anterior.