Dibujo en blanco y negro

Descripción generada automáticamente con confianza baja

COMPILADORES

PROFESOR: TECLA PARRA ROBERTO

GRUPO: 5CM4

PRACTICA 1

ALUMNO: LOPEZ PEREZ ALBERTO ANDREI

Texto

Descripción generada automáticamente con confianza media

# Introducción

En esta práctica se implementaron los conceptos base de compiladores y la implementación de YACC, donde para reforzar dichos conocimientos se creó la necesidad de crear un programa que resuelva una problemática, donde fue elegida la **Calculadora de Vectores**.

El enunciado de la problemática a resolver dice:

"*Suponga que cuenta con el código de el producto punto, el producto cruz , la multiplicación por un escalar, la suma, la resta y la magnitud. Escribir una especificación de YACC para evaluar expresiones que involucren operaciones con* ***vectores****."*

Además de proponernos ejemplos de Entradas y Salidas:

*Entrada* : [ 1 2 3 ] + [ 2 4 6 ]

*Salida* : [ 3 6 9 ]

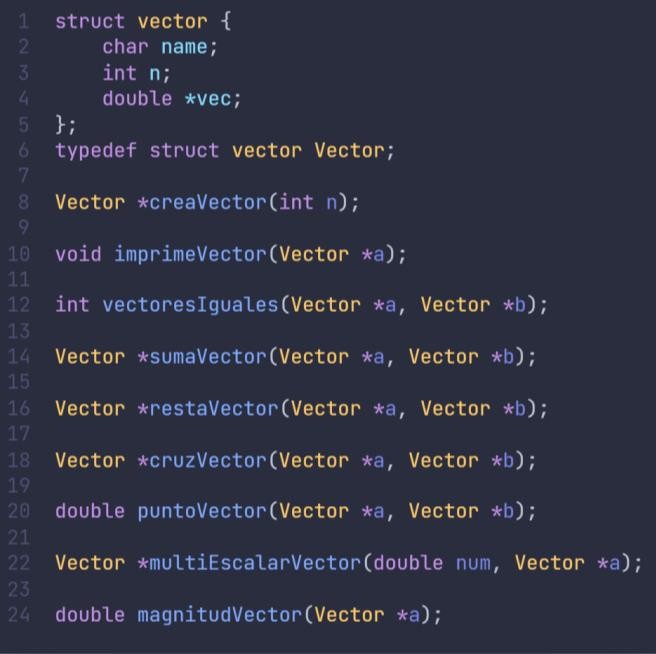
*Entrada* : [ 1 2 3 4 5 ] + [ 2 4 6 8 10 ]

*Salida* : [ 3 6 9 12 15 ]

Y para poder resolver lo siguiente se utilizaron diferentes reglas y se repasaron los conceptos de la materia **Análisis Vectorial** para poder generar una calculadora que pueda realizar las operaciones anteriormente mencionadas ajustándonos en sus limitaciones, asociatividad y precedencia.

**Desarrollo**

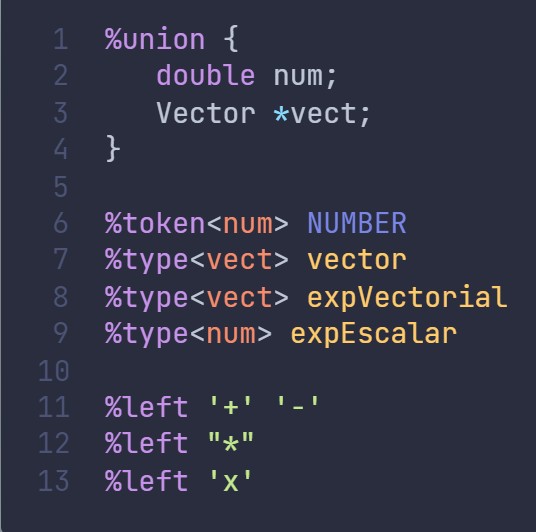
Para realizar esta práctica se utilizaron algunas funciones de vectores ya proveídas por el profesor y una estructura, sin embargo varias de ellas se modificaron y crearon nuevas, dando resultado a los archivos **vector.h** y **vector.c** donde vale la pena remarcar el contenido de la estructura y las funciones de los vectores.



E*structura vector y declaraciones de operaciones entre vectores en archivo vector.h*

Entonces, basándonos en todas las operaciones se construyo el archivo **calcVectores.y**

el cual contiene diferentes partes a remarcar.

Primero que nada, definimos los tipos de la pila de YACC usando la unión donde vamos a tener dos tipos, **num** de tipo double y **vect** de tipo Vector\*. Seguidamente definimos un token (que solo será **NUMERO**) y definimos los tipos de los no terminales que guardaran su valor en la pila de YACC.

Además inmediatamente de ahí definiremos nuestros operadores a utilizar su asociatividad y precedencia, los cuales sera **'+'** y **'-'** para suma y resta, **'\*'** y **'x'** para producto punto y cruz.

Después **definimos las reglas** que vamos a utilizar para la calculadora:



Donde cada no terminal simboliza:

**list:** Es el no terminal que nos permite agregar diferentes expresiones cada salto de línea.

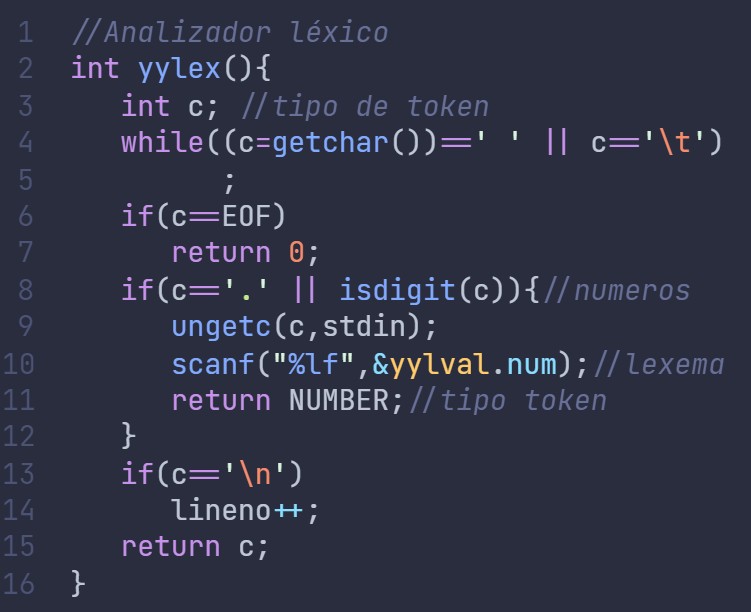
**expVectorial:** Simboliza una expresión vectorial (por lo que a la pila de YACC le va a devolver un Vector\*). Cabe mencionar que en este tipo de expresiones se pone las reglas para poder realizar las operaciones de suma, resta, producto cruz y multiplicación por un escalar *(las cuales son operaciones que nos devuelven un vector)*. Además damos la posibilidad de agregar paréntesis a una expresión vectorial.

**expEscalar:** Simboliza una expresión escalar (por lo que a la pila de YACC le va a devolver un double). Cabe mencionar que en este tipo de expresiones se pone las reglas para poder realizar las operaciones de magnitud y producto punto *(las cuales son operaciones que nos devuelven un escalar)*. Además damos la posibilidad de agregar paréntesis a una expresión escalar.

**vector:** Simboliza el como ingresar un vector, donde dentro de los corchetes puede existir una gran lista de números. Cabe mencionar que en la acción gramática es donde generamos el vector que el usuario escribió con ayuda de un pila *(de la cual hablaremos más adelante)*.

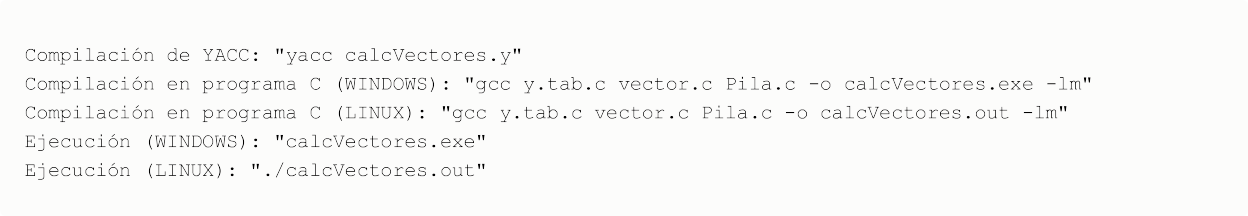
**listnum:** Simboliza todos los números que el usuario puede meter en el vector. Cabe mencionar que en la acción gramática nosotros metemos todos esos números en una pila *(de la cual hablaremos más adelante)* para posteriormente ser puestos en un vector cuando se llegue a la acción gramática de **vector**.

Finalmente en la parte de **código de soporte** vemos lo siguiente:

El analizador **yylex()** es muy parecido al visto en clase en **HOC1.y** donde solo obtenemos puros números de tipo double.

Y después declaramos un método **init()** donde vamos a inicializar una Pila (creada con **Pila.h** y **Pila.c**) la cual nos va a servir para que en la acción gramatical de **listnum** nosotros podamos meter a esa pila todos los números ingresados y después, cuando se ejecute el método **creaInputVector()** de la acción gramatical de **vector**, podamos meter todos los números de la pila en un Vector\* y se puedan utilizar todas las funciones de **vector.h** llamadas desde la acción gramatical correspondiente.

# Compilación y Ejecución

Para poder compilar y ejecutar:

# Programa en Ejecución

**Ejemplos propuestos:**

**Ejemplos propios:**

**Errores posibles:**

# Conclusión

La realización de esta práctica se podría considerar de nivel intermedio, pero realmente si tienes bien fundamentados los conocimientos de cómo es que funciona YACC, los conceptos básicos de las gramáticas y todo lo visto en clase puede que se torne más sencillo. Si bien, considero que la parte que me costó más trabajo pensar fue el cómo crear el vector, sin embargo, al implementar una pila propia realmente me solucionó demasiado el problema.

También algo que de inicio olvide completamente es que podemos darle varios tipos a lo que podemos guardar en la pila de YACC (usando la unión) y una vez recordado esto fue más trivial poder crear las reglas y sobre todo saber que tipo son cada una (si me van a devolver un vector o un escalar) y a partir de ahí se resolvía muy sencillo cada una de las implementaciones de los vectores.

Otra problemática que a lo mejor en el enunciado del problema no encontramos mucha especificación es recordar las reglas en cuanto las operaciones entre vectores ya que debemos de tomar en cuenta que no se pueden hacer operaciones entre vectores que tienen diferente dimensión y, en caso del producto cruz, solo se puede hacer en R3.

Además, recordar reglas como *A* ⋅ *B* × *C* = *A* × *B* ⋅ *C* son valiosísimas para saber si nuestras reglas en la especificación de YACC son válidas y estamos haciendo las cosas bien.