INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO (ESCOM)

COMPILADORES

NOMBRE DEL ALUMNO:

- SANTOS MÉNDEZ ULISES JESÚS

PRÁCTICA:

- TABLA DE SÍMBOLOS

NÚMERO DE PRÁCTICA: 3
OPCIÓN 2:

- CALCULADORA DE VECTORES

FECHA DE ENTREGA:

- 24/04/2023

GRUPO:

- 3CM14

Calculadora de vectores con tabla de símbolos

Introducción:

En esta práctica se busca aplicar lo visto en clase respecto a la tabla de símbolos vista en el HOC3, después de un estudio del código que compone a la tabla de símbolos, ahora se regresa al uso de YACC en C y C++ que se utilizó para la práctica 1, se utilizarán los archivos en C utilizados para la práctica 1 y se le harán las modificaciones respectivas en conjunto con los archivos que están en la carpeta que nos compartió el maestro en el HOC3, se copiarán a la carpeta donde se encuentran los archivos copiados de la práctica 1.
En la carpeta de HOC3 que nos compartió el maestro, se compilo y se ejecutó el programa generado.

Objetivo:
Agregar una tabla de símbolos para permitir nombres de variables de mas de una letra y agregar a la gramática la producción para el operador de asignación. Para la tabla de símbolos use una lista simplemente ligada (en leng C o Java) o una tabla hash (de las librerías de Java).

Desarrollo:

Principalmente se revisa la carpeta de HOC3 y se tienen los siguientes archivos:

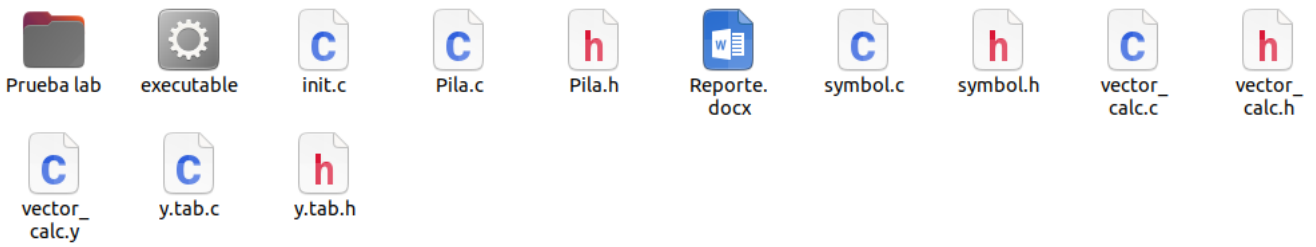
Se abre una terminal donde se compilo el archivo hoc3..y se compilaron los archivos en C

```
ulisespc04@ulisessm04:/media/ulisespc04/Ulises/OCTAVO SEMESTRE/Compiladores/Prác
ticas/Practica 3/Prueba lab$ ls
executable  hoc3.y  hoc.h  init.c  math.c  symbol.c  y.tab.c  y.tab.h
ulisespc04@ulisessm04:/media/ulisespc04/Ulises/OCTAVO SEMESTRE/Compiladores/Prác
ticas/Practica 3/Prueba lab$ yacc hoc3.y
yacc: 1 shift/reduce conflict.
ulisespc04@ulisessm04:/media/ulisespc04/Ulises/OCTAVO SEMESTRE/Compiladores/Prác
ticas/Practica 3/Prueba lab$ gcc y.tab.c init.c math.c symbol.c -lm
math.c: In function ‘errcheck’:
math.c:28:17: warning: implicit declaration of function ‘execerror’ [-Wimplicit-
function-declaration]
   28 |                 execerror(s, "argument out of donain");
      |                 ^~~~~~~~~
ulisespc04@ulisessm04:/media/ulisespc04/Ulises/OCTAVO SEMESTRE/Compiladores/Prác
ticas/Practica 3/Prueba lab$
```

Se ejecuta el programa y tenemos la siguiente salida:

```
ulisespc04@ulisessm04:/media/ulisespc04/Ulises/OCTAVO SEMESTRE/Compiladores/Prác
ticas/Practica 3/Prueba lab$ ./executable
5+5
        10
PI
        3.1415927
E
        2.7182818
GAMMA
        0.57721566
5/2
        2.5
```

Ahora se copiaron archivos de la práctica 1 de la calculadora de vectores, y se implemento la lista de símbolos, los archivos que quedaron en conjunto fueron los siguientes



En la gramática se agrego la tabla de símbolos, en la union se agrego un apuntador a sym de tipo Symbol, donde se agregaron tokens en symbol, como VAR, VARVECTOR, VARESCALAR, INDEF

```
%{
#include "vector_calc.h"
#include "symbol.h"
void yyerror (char *s);
int yylex ();
void warning(char *s, char *t);
%}

%union{
    double num;
    Vector *vect;
    Symbol *sym;
}
%token<num> NUMBER
%token<sym> VAR VARVECTOR VARESCALAR INDEF
%type<vect> vector
%type<vect> expVectorial asgnVector
%type<num> expEscalar asgnEscalar
```

En la siguiente imagen se agregó la parte de las variables donde se le puede agregar nombre y agregar a la asignación el operador de asignación:

```
list:
    | list '\n'
    | list asgnVector '\n'   {imprimeVector($2);}
    | list asgnEscalar '\n'  {printf("\033[0;36m%f\n\033[0;0m", $2);}
    | list expVectorial '\n' {imprimeVector($2); }
    | list expEscalar '\n'   {printf("\033[0;36m%f\n\033[0;0m", $2); }
    ;
asgnVector: VAR '=' expVectorial {$$= $1->u.varVector = $3; $1->type=VARVECTOR;} //Declaracion
    | VARESCALAR '=' expVectorial {$$= $1->u.varVector = $3; $1->type=VARVECTOR;} //Redefiniciones
    | VARVECTOR '=' expVectorial {$$= $1->u.varVector = $3; $1->type=VARVECTOR;}
    ;
asgnEscalar: '#' VAR '=' expEscalar {$$= $2->u.varEscalar = $4; $2->type=VARESCALAR;} //Declaracion
    | '#' VARESCALAR '=' expEscalar {$$= $2->u.varEscalar = $4; $2->type=VARESCALAR;} //Redefiniciones
    | '#' VARVECTOR '=' expEscalar {$$= $2->u.varEscalar = $4; $2->type=VARESCALAR;}
    ;
```

A continuación presento la tabla de símbolos que se implementó

```
#include "vector_calc.h"
typedef struct Symbol{
    char *name;
    short type;
    union {
        double varEscalar;
        Vector *varVector;
    }u;
    struct Symbol *next;
} Symbol;

Symbol *install(char *s,int t, double d), *lookup(char *s);
```

En el fragmento de código anterior se crea la tabla de símbolos donde short type es donde se almacena el tipo de dato que ocuparemos en la gramática, es el tipo que sale en los tokens como VAR, VARVECTOR, VARESCALAR, INDEF, en struct Symbol *next se esta apuntando al siguiente elemento de la lista ligada.

Finalmente se compilan los archivos YACC y C y se procede a ejecutar el programa como se muestra a continuación:

```
ulisespc04@ulisessm04:/media/ulisespc04/Ulises/OCTAVO SEMESTRE/Compiladores/Prác
ticas/Practica 3$ ls
 init.c   Pila.h        Reporte.docx   symbol.h        vector_calc.h
 Pila.c  'Prueba lab'   symbol.c       vector_calc.c   vector_calc.y
ulisespc04@ulisessm04:/media/ulisespc04/Ulises/OCTAVO SEMESTRE/Compiladores/Prác
ticas/Practica 3$ yacc -d vector_calc.y
ulisespc04@ulisessm04:/media/ulisespc04/Ulises/OCTAVO SEMESTRE/Compiladores/Prác
ticas/Practica 3$ gcc y.tab.c init.c Pila.c symbol.c vector_calc.c -o executable -lm
vector_calc.y: In function ‘init’:
vector_calc.y:96:9: warning: implicit declaration of function ‘initConstants’; did you mean ‘initstate’? [-Wimplicit-function-declar
ation]
   96 |         initConstants();
      |         ^~~~~~~~~~~~~
      |         initstate
init.c:16:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
   16 | initConstants( ) {
      | ^~~~~~~~~~~~~
ulisespc04@ulisessm04:/media/ulisespc04/Ulises/OCTAVO SEMESTRE/Compiladores/Prácticas/Practica 3$
```

```
ulisespc04@ulisessm04:/media/ulisespc04/Ulises/OCTAVO SEMESTRE/Compiladores/Prácticas/Practica 3$ ./executable
[1 2 3]+[3 2 1]
[ 4.000000 4.000000 4.000000 ]
a=[1 2 3]
[ 1.000000 2.000000 3.000000 ]
a
[ 1.000000 2.000000 3.000000 ]
# d= 46
46.000000
d
46.000000
d*a
```

Conclusión:

En conclusión se logró visualizar la importancia de una tabla de símbolos para lograr el funcionamiento en el caso que se desee implementar más variables, o algún token que se requiera para optimizar el funcionamiento del programa, fue un tanto complicado entender como implementarlo, se realizó una función que permite inicializar constantes las cuales nos permite asignar a una variable un vector o una constante, esta práctica fue muy complementaria para los temas vistos en clase respecto al HOC3 que tuve muchas dudas sobre el uso de la tabla de símbolos.