



INSTITUTO POLITECNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO (ESCOM)



COMPILADORES

NOMBRE DEL ALUMNO:

- SANTOS MÉNDEZ ULISES JESÚS

PRÁCTICA:

- CALCULADORA DE VECTORES CON MÁQUINA DE PILA

NÚMERO DE PRÁCTICA: 4

FECHA DE ENTREGA:

- 08/05/2023

GRUPO:

- 3CM14

Calculadora de vectores con maquina de pila.

Introducción:

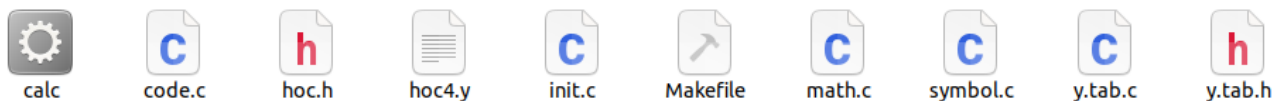
En esta práctica se busca aplicar lo que se aprende durante la clase, respecto a la implementación de una máquina virtual de pila cuando ya se tiene una tabla de símbolos haciendo uso de la práctica anterior y los documentos que se tienen del HOC4, se hace uso del YACC en C y C++ que se utilizó en la práctica 1 y en la práctica 3, se harán algunas modificaciones necesarias relacionadas con lo que el maestro compartió, todo esto va en relación que se compiló y se ejecutó el HOC 4.

Objetivos:

Modificar la especificación de YACC de HOC4 en la práctica 3 agregando los macros con parámetros, tokens y producciones a la gramática, modificando la parte val de %union para que trabaje con vectores, modificar yylex() y main() basándose en HOC4.

Desarrollo:

Principalmente se revisa la carpeta de HOC4 y se tienen los siguientes archivos:



Se abre una terminal donde se compilo el archivo hoc4.y y se compilaron los archivos en C

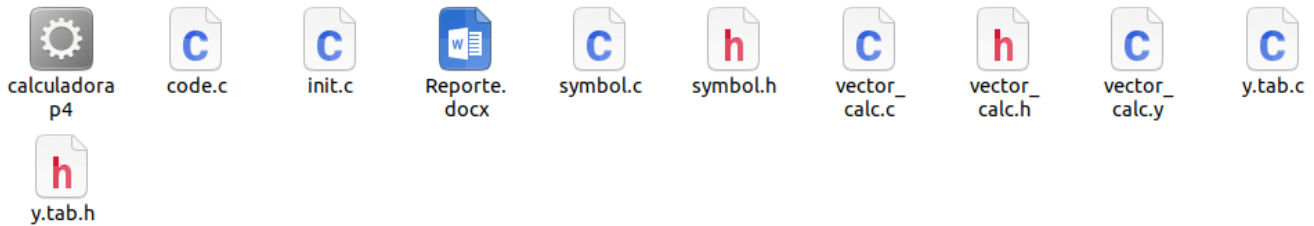
```
ulisespc04@ulisesm04:~/Descargas/compila/compipracticass_codigo1/practica3/hoc4$ gcc y.tab.c code.c init.c math.c symbol.c -o calc -lm
```

Se ejecuta el programa y tenemos la siguiente salida:

```
ulisespc04@ulisesm04:/media/ulisespc04/Ulises/OCTAVO SEMESTRE/Compiladores/Prácticas/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 3 de la calculadora de vectores y se implementó la máquina virtual de pila cuando ya hay una tabla de símbolos, los archivos en conjunto son:



En la gramática de nuestra calculadora de vectores tenemos la definición de funciones que se mandarán a llamar para agregar elementos a la pila y para extraer elementos de la pila:

```
%{
#include "symbol.h"
void yyerror (char *s);
int yylex ();
void warning(char *s, char *t);
#define code2(c1,c2)      code(c1); code(c2);
#define code3(c1,c2,c3); code(c1); code(c2); code(c3);
%}

%union{
    Symbol *sym;
    Inst *inst;
}

%token<sym>  NUMBER VAR VARVECTOR VARESCALAR INDEF
%left '+' '-'
%left '*'
%left 'x'
%%
```

A diferencia de la práctica anterior se omitió en el código la parte relacionada a Vector y a double num., además solo se busca llamar a la tabla de símbolos para ejecutar cada función correspondiente.

```

list:
| list '\n'
| list asgnVector '\n' {code2(printVector, STOP); return 1; }
| list asgnEscalar '\n' {code2(printEscalar, STOP); return 1; }
| list expVectorial '\n' {code2(printVector, STOP); return 1; }
| list expEscalar '\n' {code2(printEscalar, STOP); return 1; }
;
asgnVector: VAR '=' expVectorial {code3(varpush, (Inst)$1, assignVector);} //Declaracion
| VARESCALAR '=' expVectorial {code3(varpush, (Inst)$1, assignVector);} //Redefiniciones
| VARVECTOR '=' expVectorial {code3(varpush, (Inst)$1, assignVector);}
;
asgnEscalar: '#' VAR '=' expEscalar {code3(varpush, (Inst)$2, assignEscalar);} //Declaracion
| '#' VARESCALAR '=' expEscalar {code3(varpush, (Inst)$2, assignEscalar);} //Redefiniciones
| '#' VARVECTOR '=' expEscalar {code3(varpush, (Inst)$2, assignEscalar);}
;
expVectorial:vector
| VAR {code3(varpush, (Inst)$1, evalVector);}
| VARVECTOR {code3(varpush, (Inst)$1, evalVector);}
| expVectorial '+' expVectorial {code(add);}
| expVectorial '-' expVectorial {code(sub);}
| expVectorial 'x' expVectorial {code(cruz);}
| expEscalar '*' expVectorial {code(multiEscalarVect);}
| expVectorial '*' expEscalar {code(multiVectEscalar);}
| '(' expVectorial ')'
;

```

En la asignación se le pasan parámetros a la función code2 que van a permitir llamar a la producción asgnVector o asgnEscalar, expVectorial o expEscalar.

```

Datum pop( ){      /* sacar y retornar de la pila el elemento del tope */
    if (stackp <= stack)
        printf("\x1b[31m%s\n\x1b[0m", "stack underflow");
    return *--stackp;
}

void escalarpush(){
    Datum d;
    d.num = ((Symbol *)*pc++)->u.varEscalar;
    push(d);
}

void numpush(){
    escalarpush();
    dimVector++;
}

void vectorpush(){
    Vector *vec = creaVector(dimVector);
    int i;
    for(i=0; i<dimVector; i++){
        Datum t = pop();
        vec->vec[i]=t.num;
    }
}

```

```

void varpush( ) { /* meter una variable a la pila */
    Datum d;
    d.sym = (Symbol *) (*pc++);
    push(d);
}

void evalEscalar( ) /* evaluar una variable escalar en la pila */
{
    Datum d;
    d = pop();
    if(d.sym->type == INDEF)
        printf("\x1b[31mVariable no definida: %s\x1b[0m\n", d.sym->name);
    d.num = d.sym->u.varEscalar;
    push(d);
}

void evalVector( ) /* evaluar un vector en la pila */
{
    Datum d;
    d = pop();
    if(d.sym->type == INDEF || d.sym->type == VAR){
        printf("\x1b[31mVariable no definida: %s\x1b[0m\n", d.sym->name);
        d.vect=NULL;
    }
}

```

Se agregaron las siguientes llamadas a función que se pide en el formato de lo que se debe de agregar a la práctica, esto se agrego en el archivo code.c para que sea apto para trabajar con vectores.

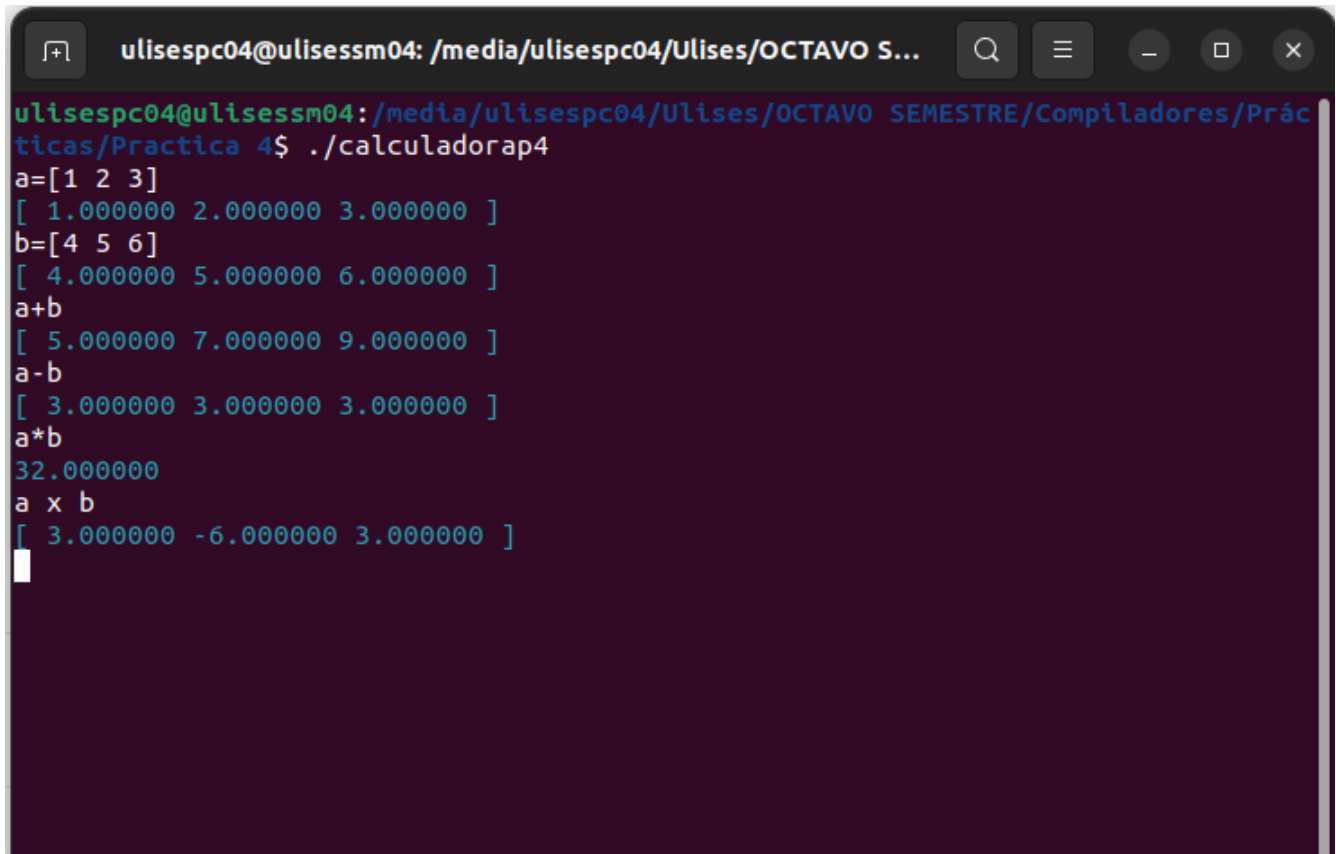
Finalmente se compilan los archivos YACC y C y se procede a ejecutar el programa como se muestra a continuación:

```

ulisespc04@ulisesm04: /media/ulisespc04/ulises/OCTAVO SEMESTRE/Compiladores/Prac
ticas/p4prueba$ yacc -d vector_calc.y
ulisespc04@ulisesm04: /media/ulisespc04/ulises/OCTAVO SEMESTRE/Compiladores/Prac
ticas/p4prueba$ gcc y.tab.c code.c init.c symbol.c vector_calc.c -o calculadora -lm
vector_calc.y: In function 'init':
vector_calc.y:80:9: warning: implicit declaration of function 'initConstants'; did you mean 'initstate'? [-Wimplicit-function-declar
ation]
  80 |         initConstants();
      |         ^~~~~~
      |         initstate
vector_calc.y: In function 'main':
vector_calc.y:86:13: warning: implicit declaration of function 'initcode' [-Wimplicit-function-declaration]
  86 |         for(initcode(); yyparse(); initcode())
      |         ^~~~~~
vector_calc.y:87:17: warning: implicit declaration of function 'execute' [-Wimplicit-function-declaration]
  87 |         execute(prog);
      |         ^~~~~~
vector_calc.y: In function 'yyparse':
vector_calc.y:18:26: warning: implicit declaration of function 'code'; did you mean 'code2'? [-Wimplicit-function-declaration]
  18 | #define code2(c1,c2)    code(c1); code(c2);
      |                          ^~~~~~
vector_calc.y:34:10: note: in expansion of macro 'code2'
  34 |         list assignVector '\n' {code2(printVector, STOP); return 1; }
      |         ^~~~~~
init.c:16:1: warning: return type defaults to 'int' [-Wimplicit-int]
  16 | initConstants( ) {
      | ^~~~~~

```

Se procede a ejecutar el archivo .exe teniendo el siguiente resultado:



```
ulisespc04@ulises-sm04: /media/ulisespc04/Ulises/OCTAVO S...
ulisespc04@ulises-sm04:/media/ulisespc04/Ulises/OCTAVO SEMESTRE/Compiladores/Prácticas/Practica 4$ ./calculadorap4
a=[1 2 3]
[ 1.000000 2.000000 3.000000 ]
b=[4 5 6]
[ 4.000000 5.000000 6.000000 ]
a+b
[ 5.000000 7.000000 9.000000 ]
a-b
[ 3.000000 3.000000 3.000000 ]
a*b
32.000000
a x b
[ 3.000000 -6.000000 3.000000 ]
```

Conclusión:

En conclusión se logró implementar la máquina virtual de pila a la gramática que contenía una tabla de símbolos, esto nos facilita el mandar a llamar y establecer parámetros de STOP para poder ocupar lo menos que se pueda de espacio en la RAM, según lo visto en clase, se logró un buen funcionamiento aunque se tuvieron algunas complicaciones con el HOC4 que se solucionaron para poder tener el preámbulo de esta práctica.