INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO (ESCOM)

COMPILADORES

NOMBRE DEL ALUMNO:

- SANTOS MÉNDEZ ULISES JESÚS

PRÁCTICA:

- FOR EN CALCULADORA DE VECTORES

NÚMERO DE PRÁCTICA: 6

FECHA DE ENTREGA:

- 14/06/2023

GRUPO:

- 3CM14

Calculadora de vectores con condicionales y ciclos.

Introducción:

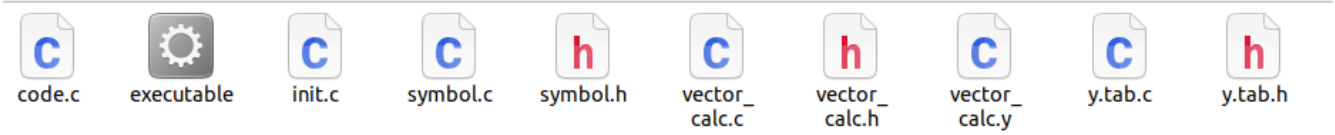
En esta práctica se busca aplicar lo que se aprende durante la clase, respecto a la implementación de condicionales y ciclos haciendo uso de la práctica anterior y los documentos que se encuentran en la carpeta practica 5, se hace uso del YACC en C y C++ que se utilizó en la práctica 1,3,4 y en la práctica 5, se harán algunas modificaciones necesarias relacionadas con lo que el maestro compartió.

Objetivos:

- Agregar una producción a la gramática para generar el ciclo for.
- Dibujar el mapa de memoria del ciclo for.
- Tener en cuenta que el código que se genera en postfijo para dibujar dicho para de memoria.
- Poner la cantidad de STOP's que hagan falta y poner los STOP's en el lugar adecuado.
- Codificar a forcode().

Desarrollo:

Se utilizan los mismos archivos que se tenían en la práctica 5 y son los siguientes:



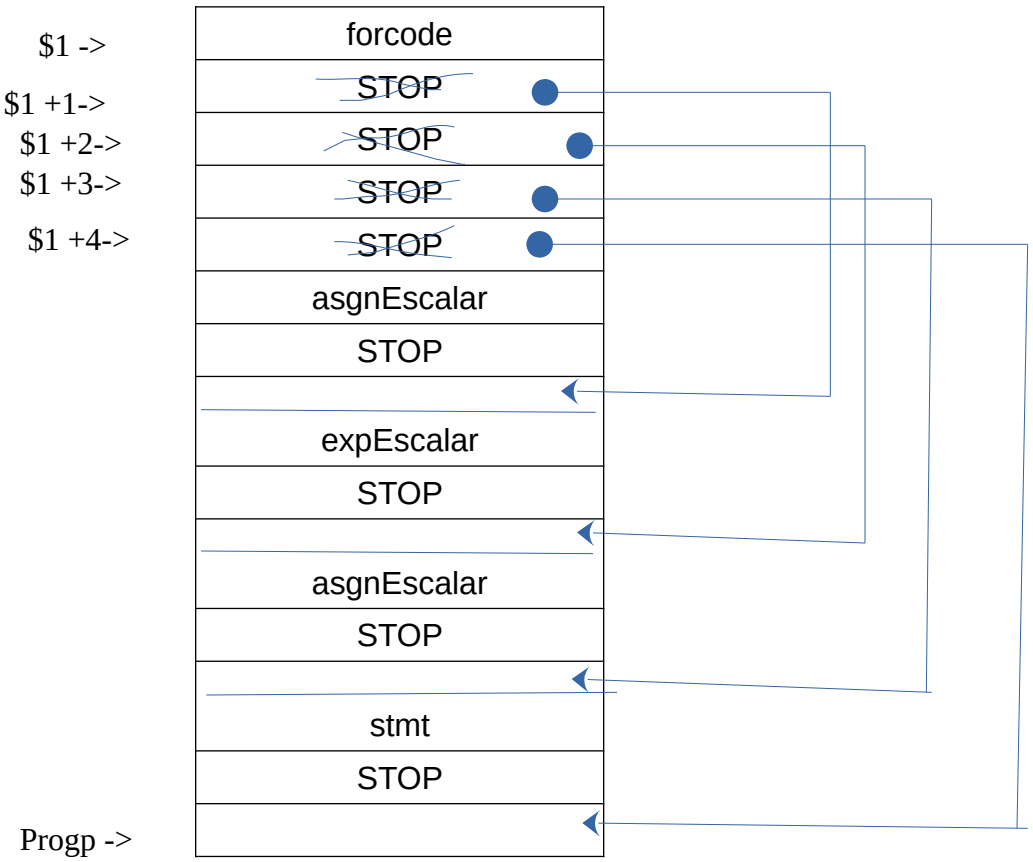
Se implemento en el archivo vector_calc.y algunas producciones que nos van a permitir hacer la construcción del ciclo dor como se muestra a continuación:

```
| for '(' stmtfor ';' condfor ';' stmtfor ')' stmt end {
                                        ($1)[1] = (Inst)$5;
                                        ($1)[2] = (Inst)$7;
                                        ($1)[3] = (Inst)$9;
                                        ($1)[4] = (Inst)$10;
                                    }
```

Las posiciones del arreglo mostradas se refieren a los STOP's que se van a eliminar y a los cuales se va a apuntar después de que se ejecute el STOP.

```
condfor: expEscalar {code(STOP); $$ = $1; }
    ;
stmtfor: asgnEscalar    {code(pop1); code(STOP); }
    | asgnVector                    { code(pop1); code(STOP); }
    | PRINT expEscalar              { code(prexpresc); $$ = $2; code(STOP);}
    | PRINT expVectorial            { code(prexprvec); $$ = $2; code(STOP);}
    ;
while:  WHILE   {   $$ = code3(whilecode,STOP,STOP); }
    ;
if:     IF   { $$=code(ifcode); code3(STOP, STOP, STOP); }
    ;
for: FOR    {$$=code(forcode); code3(STOP, STOP, STOP); code(STOP);}
    ;
end:      /* nada */{ code(STOP); $$ = progp; }
    ;
```

En la gramática anterior se observan las producciones y los terminales a utilizar que podemos elegir para poder efectuar el ciclo así como otras operaciones, además muchos de los STOP's presentes estarán dentro de la pila como se muestra a continuación.



Finalmente tenemos la ejecución de nuestro programa:

```
ulisespc04@ulisessm04:/media/ulisespc04/Ulises/OCTAVO SEMESTRE/Compiladores/Prác
ticas/Practica 6$ ./executable
for(i=[0 0 0]; i<=[10 10 10]; i=i+[1 1 1]){ print i}
[ 0.000000 0.000000 0.000000 ]
[ 1.000000 1.000000 1.000000 ]
[ 2.000000 2.000000 2.000000 ]
[ 3.000000 3.000000 3.000000 ]
[ 4.000000 4.000000 4.000000 ]
[ 5.000000 5.000000 5.000000 ]
[ 6.000000 6.000000 6.000000 ]
[ 7.000000 7.000000 7.000000 ]
[ 8.000000 8.000000 8.000000 ]
[ 9.000000 9.000000 9.000000 ]
[ 10.000000 10.000000 10.000000 ]
```

Conclusión:

En conclusión el implementar la función y la gramática para el ciclo for fue más sencilla gracias a el conocimiento previo de la práctica anterior ya que se sabe con más profundidad como actúa en el mapa de memoria cada STOP y cada instrucción.