



**Instituto Politécnico Nacional**  
**Escuela Superior de Cómputo**  
**Departamento de Ciencias e Ingeniería**  
**de la Computación**



**Compiladores**

## **Práctica 6: Ciclo FOR HOC5**

### **Opción: Calculadora de vectores**

**Grupo:** 5CM2

**Alumno:**

Sandoval Hernández Eduardo

**Profesor:**

Tecla Parra Roberto

**Fecha de entrega:**

10 de junio de 2022

## **Introducción**

Yet Another Compiler Compiler (YACC) es un generador de analizadores. A partir de un archivo fuente en yacc se puede generar un archivo fuente en C el cual contendrá el analizador sintáctico. Este analizador requerirá de un analizador léxico externo para funcionar, esto debido a que el archivo fuente en C que genera yacc contiene llamadas a la función `yylex()` la cual debe estar definida y debe ser capaz de devolver el tipo de lexema encontrado. Adicionalmente es necesario incluir una función `yyerror()` la cual será invocada cuando el analizador sintáctico encuentre un símbolo que no encaje con la gramática [1].

El presente reporte consiste en los resultados obtenidos de la práctica 6 con el tema relacionado a HOC5, para esta práctica se reutilizó el código de la práctica 5 manteniendo la opción de la calculadora de vectores, para esta práctica se complementó el código anterior para que la calculadora pueda realizar un ciclo `for` y así tener dos métodos para realizar ciclos (junto con el `while` de la práctica anterior).

## Desarrollo

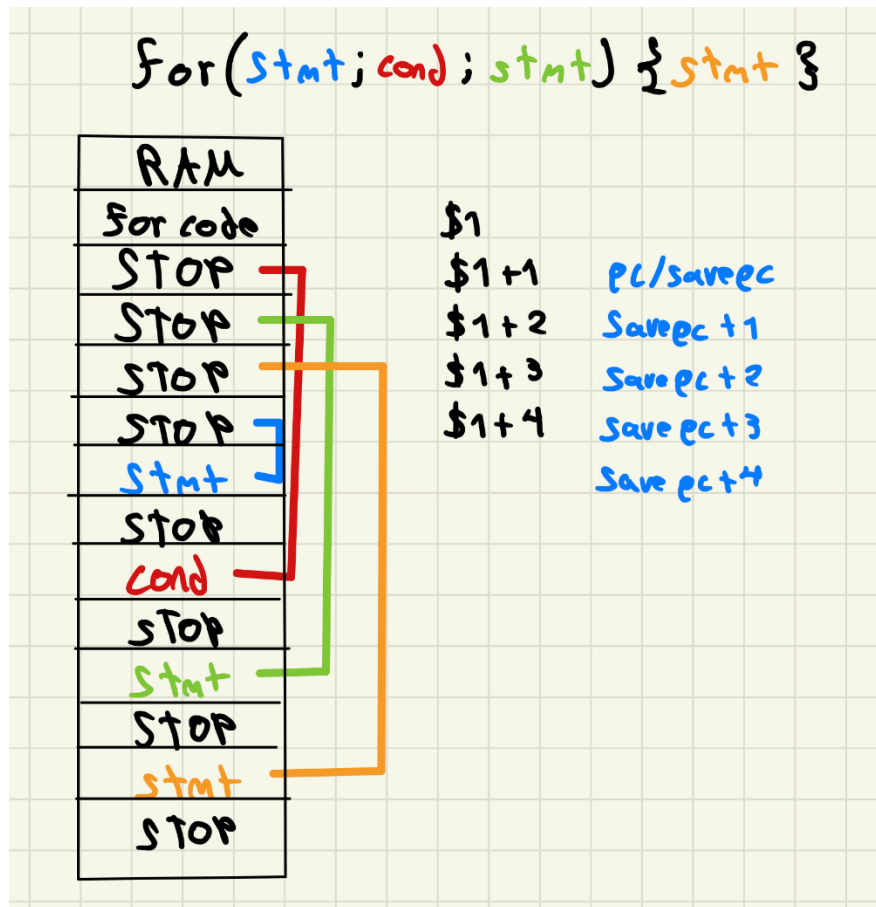
Para la realización de esta práctica se agregó a la estructura keywords la palabra “for” para que pueda reconocer la instrucción del ciclo:

```
static struct {
    char *name;
    int val;
} keywords[] = {
    "if", IF,
    "else", ELSE,
    "while", WHILE,
    "print", PRINT,
    "for", FOR,
    0, 0,
};
```

Manteniendo las funciones de comparación y operaciones booleanas de la práctica anterior, se agregó la función forcode la cual se encargará de realizar el ciclo for:

```
void forcode(){
    Datum d;
    Inst* savepc = pc;
    execute(savepc + 4);
    execute(*((Inst **)(savepc)));
    d = pop();
    while(d.val){
        execute(* ( (Inst **)(savepc + 2)));
        execute(* ( (Inst **)(savepc + 1)));
        execute(*((Inst **)(savepc)));
        d = pop();
    }
    pc = *((Inst **)(savepc + 3));
}
```

Esta función se realizó tomando en cuenta el mapa de memoria necesario para el ciclo for:



Para entender el funcionamiento de la función se puede explicar de la siguiente manera:

1. Se ejecuta el statement que se encuentra en savepc +4 y que viene siendo el statement inicial del ciclo el cual se realiza una sola vez.
2. Posteriormente se ejecuta la instrucción que se encuentra en savepc la cual viene siendo la condición para que se cumpla el ciclo, el resultado se obtiene en un Datum.
3. Si se cumple la condición, se entra al ciclo donde primero se ejecuta la instrucción a la que apunta savepc + 2 que viene siendo el cuerpo del ciclo, posteriormente se ejecuta la instrucción a la que apunta savepc + 1 y que viene siendo el 3er statement del ciclo for que se ejecuta al final. Posteriormente se repite el paso 2.
4. Si no se cumple la condición, el ciclo termina y se le envía a pc la dirección de retorno del ciclo.

En cuanto a la gramática, se agregó un terminal y dos no terminales para la producción de FOR:

```
%token<sym>      FOR
%type<inst>      for exprn
```

Posteriormente se modificó la producción stmt para agregar la producción de for:

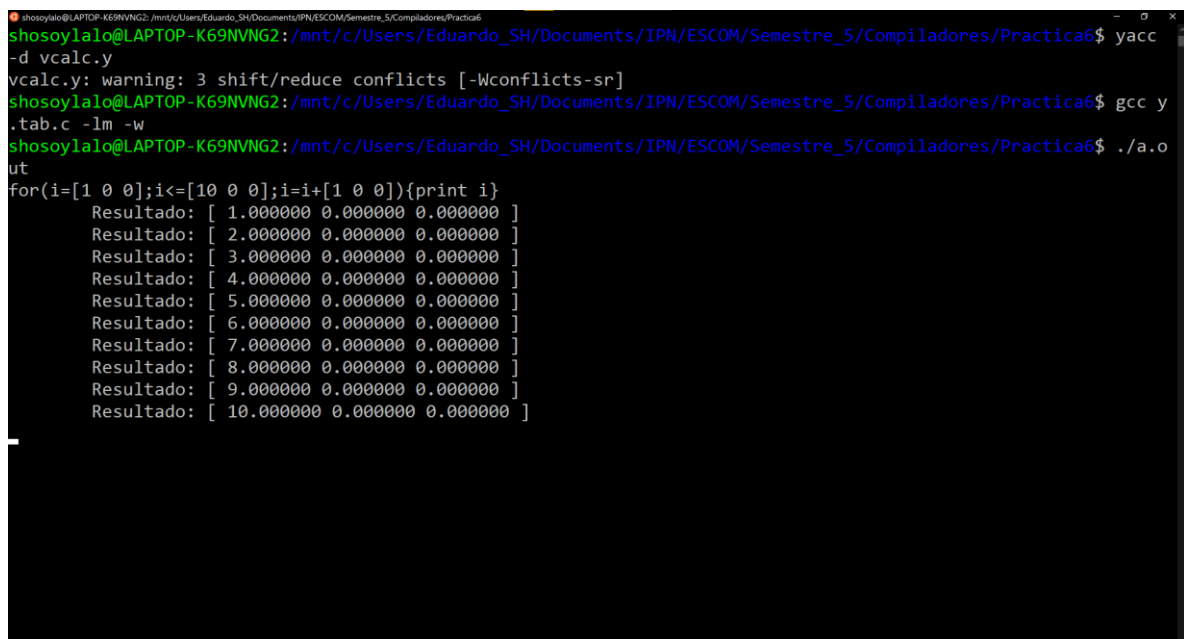
```
stmt: exp                {code((Inst)pop);}
    | PRINT exp          {code(printVector); $$ = $2;}
    | while cond stmt end {
                          ($1)[1] = (Inst)$3;
                          ($1)[2] = (Inst)$4;
                          }
    | if cond stmt end   {
                          ($1)[1] = (Inst)$3;
                          ($1)[3] = (Inst)$4;
                          }
    | if cond stmt end ELSE stmt end {
                          ($1)[1] = (Inst)$3;
                          ($1)[2] = (Inst)$6;
                          ($1)[3] = (Inst)$7;
                          }
    | for '(' exprn ';' exprn ';' exprn ')' stmt end {
                          ($1)[1] = (Inst)$5;
                          ($1)[2] = (Inst)$7;
                          ($1)[3] = (Inst)$9;
                          ($1)[4] = (Inst)$10;
                          }
    | '{' stmtlist '}'   {$$ = $2;}
    ;
```

Finalmente, las producciones de for y esxprn:

```
for: FOR    {$$ = code(forcode); code3(STOP,STOP,STOP); code
(STOP);}
    ;
```

```
exprn: exp                { $$ = $1; code(STOP); }  
      | '{' stmtlist '}' { $$ = $2; }  
      ;
```

El código de soporte se mantuvo sin cambios, para la ejecución y compilación de la práctica se ejecutan los mismos comandos que en la práctica anterior, de igual manera para probar los ciclos se realizará uno que se repetirá 10 veces utilizando el ciclo for:



```
shosoylalo@LAPTOP-K69NVNG2: /mnt/c/Users/Eduardo_SH/Documents/IPN/ESCOM/Semestre_5/Compiladores/Practica6$ yacc  
-d vcalc.y  
vcalc.y: warning: 3 shift/reduce conflicts [-Wconflicts-sr]  
shosoylalo@LAPTOP-K69NVNG2: /mnt/c/Users/Eduardo_SH/Documents/IPN/ESCOM/Semestre_5/Compiladores/Practica6$ gcc y  
.tab.c -lm -w  
shosoylalo@LAPTOP-K69NVNG2: /mnt/c/Users/Eduardo_SH/Documents/IPN/ESCOM/Semestre_5/Compiladores/Practica6$ ./a.o  
ut  
for(i=[1 0 0];i<=[10 0 0];i=i+[1 0 0]){print i}  
Resultado: [ 1.000000 0.000000 0.000000 ]  
Resultado: [ 2.000000 0.000000 0.000000 ]  
Resultado: [ 3.000000 0.000000 0.000000 ]  
Resultado: [ 4.000000 0.000000 0.000000 ]  
Resultado: [ 5.000000 0.000000 0.000000 ]  
Resultado: [ 6.000000 0.000000 0.000000 ]  
Resultado: [ 7.000000 0.000000 0.000000 ]  
Resultado: [ 8.000000 0.000000 0.000000 ]  
Resultado: [ 9.000000 0.000000 0.000000 ]  
Resultado: [ 10.000000 0.000000 0.000000 ]
```

Como se muestra en la evidencia anterior, el programa puede realizar el ciclo for sin complicación alguna, demostrando que tanto la gramática como la función forcode funcionan de manera eficaz.

## Conclusión

En esta ocasión fue un poco más complicado la realización de la practica ya que a pesar de que solo fue necesario modificar ligeramente la gramática y realizar el código para la función forcode, fue esta última parte la que se me dificultó al

comienzo pues era necesario comprender como tenía que ejecutarse dicha función. Considero que aún se pueden realizar cambios en todo el código para mejorarlo.

### **Referencias**

[1] F. Simmross Wattenberg (s.f.). "El generador de analizadores sintácticos yacc". [Internet]. Disponible en <https://www.infor.uva.es/~mluisa/talf/docs/labo/L8.pdf>