



INSTITUTO POLITÉCNICO NACIONAL  
ESCUELA SUPERIOR DE CÓMPUTO

Práctica número 6:  
Calculadora para vectores  
(Con ciclo For)

6 de enero de 2021

Grupo: 3CM7

*Nombre del alumno:*  
Ramos Mesas Edgar Alain

*Número de boleta:*  
2013090243

MATERIA: COMPILADORES

## 1. Introducción

Los ciclos for son lo que se conoce como estructuras de control de flujo cíclicas o simplemente estructuras cíclicas, estos ciclos, como su nombre lo sugiere, nos permiten ejecutar una o varias líneas de código de forma iterativa, conociendo un valor específico inicial y otro valor final, además nos permiten determinar el tamaño del paso entre cada "giro" o iteración del ciclo. En resumen, un ciclo for es una estructura de control iterativa, que nos permite ejecutar de manera repetitiva un bloque de instrucciones, conociendo previamente un valor de inicio, un tamaño de paso y un valor final para el ciclo.

## 2. Desarrollo

Esta práctica consiste en agregar a la calculadora para vectores, las modificaciones necesarias para poder ejecutar un ciclo for al escribirlo en consola.

Primeramente agregamos los nuevos símbolos gramaticales necesarios para el ciclo for.

```
38 //Nuevos símbolos gramaticales para la práctica 6
39 %token<sym>      FOR
40 %type<inst>      for exprn
```

Posteriormente se especifica la sintaxis que seguirá el ciclo for al escribirse en la consola, además de definir qué será de tipo instrucción. La iteración del for está compuesta por una expresión que es la inicialización, un marcador que indica el inicio de la condición, otro marcador que indica el inicio de la expresión que es el incremento y el cuerpo de la función con el final del for.

```
121 | for '(' exprn ';' exprn ';' exprn ')' stmt end    {($1)[1] = (Inst)$5;
122 |                                                    ($1)[2] = (Inst)$7;
123 |                                                    ($1)[3] = (Inst)$9;
124 |                                                    ($1)[4] = (Inst)$10;}
125 | '{' stmtlst '}'                                  {$$ = $2;}
126 ;
127
128 cond: '(' exp ')'                                  {code(STOP); $$ = $2;}
129 ;
130
131 while: WHILE                                        {$$ = code3(whilecode, STOP, STOP);}
132 ;
133
134 if: IF                                               {$$ = code(ifcode);
135 |                                                    code3(STOP, STOP, STOP);}
136 ;
137
138 end: /* NADA */                                     {code(STOP); $$ = progp;}
139 ;
```

```

140
141      stmtlst: /* NADA */                {$$ = prog; }
142      |      stmtlst '\n'
143      |      stmtlst stmt
144      ;
145
146      //PRÁCTICA 6
147      for: FOR                            {$$ = code(forcode); code3(STOP, STOP,
148      ↪   STOP); code(STOP); }
149      ;
150      exprn: exp                          {$$ = $1; code(STOP); }
151      |      '{' stmtlst '}'              {$$ = $2; }
152      ;
153
154      %%

```

Partiendo de lo anterior, el mapa de memoria para el ciclo for queda dibujado de la siguiente manera.

```

RAM
FOR      $1
STOP     $1[2]
STOP     $1[2]
STOP     $1[3]
STOP     $1[4]
COND
STOP
COND
STOP
COND
STOP
STMT
STOP

```

Posteriormente fue necesario modificar el archivo llamado `code.c` pues era necesario codificar el método perteneciente al ciclo for. En la función `forcode` se ejecuta primero la expresión de inicialización y está solo se ejecuta una sola vez, posteriormente se ejecuta la condición y se saca el resultado de la pila, si es verdadero entra al while y se ejecuta el cuerpo del for, posteriormente se ejecuta la expresión incremento, y por último se ejecuta de nuevo la condición y si es cierta se vuelve a entrar al while así hasta que no se cumpla la condición, en caso de que no sea verdadera se ejecuta la siguiente instrucción fuera del for.

```

260  /***** PRÁCTICA 6 *****/
261  void forcode(){
262      Datum d;
263      Inst* savepc = pc;
264      execute(savepc + 4);

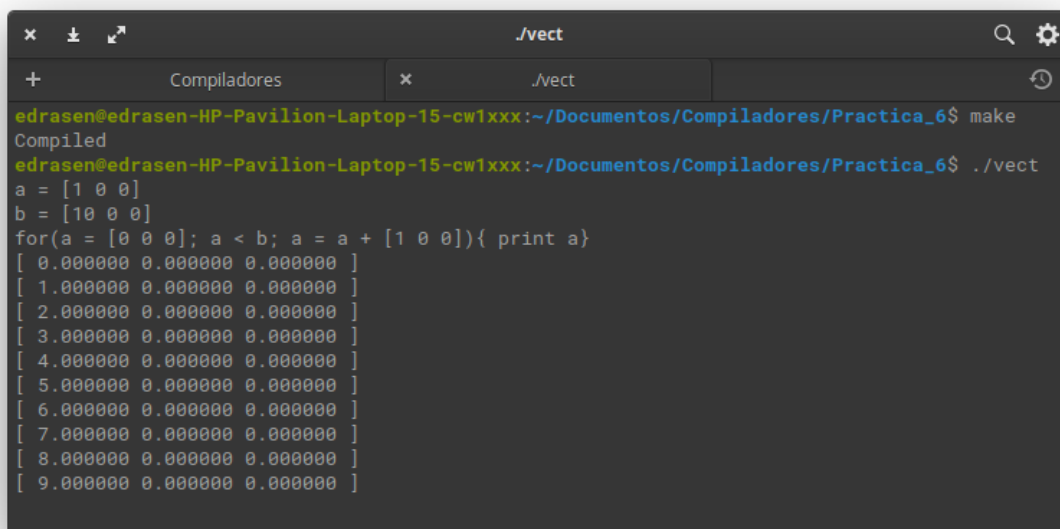
```

```

265     execute(*((Inst **)(savepc)));
266     //Se saca la instrucción
267     d = pop();
268     while(d.val){
269         execute(* ( (Inst **)(savepc + 2))); /* Cuerpo del ciclo*/
270         execute(* ( (Inst **)(savepc + 1))); // Último campo
271         pop();
272         execute(*((Inst **)(savepc))); /* CONDICION */
273         d = pop();
274     }
275     pc = *((Inst **)(savepc + 3)); /*Vamos a la siguiente posicion*/
276 }

```

Finalmente, tras realizar todas las modificaciones mencionadas se compilo todo desde un archivo Makefile y se ejecutó el programa obteniendo los siguientes resultados:



```

edrasen@edrasen-HP-Pavilion-Laptop-15-cw1xxx:~/Documentos/Compiladores/Practica_6$ make
Compiled
edrasen@edrasen-HP-Pavilion-Laptop-15-cw1xxx:~/Documentos/Compiladores/Practica_6$ ./vect
a = [1 0 0]
b = [10 0 0]
for(a = [0 0 0]; a < b; a = a + [1 0 0]){ print a}
[ 0.000000 0.000000 0.000000 ]
[ 1.000000 0.000000 0.000000 ]
[ 2.000000 0.000000 0.000000 ]
[ 3.000000 0.000000 0.000000 ]
[ 4.000000 0.000000 0.000000 ]
[ 5.000000 0.000000 0.000000 ]
[ 6.000000 0.000000 0.000000 ]
[ 7.000000 0.000000 0.000000 ]
[ 8.000000 0.000000 0.000000 ]
[ 9.000000 0.000000 0.000000 ]

```

Comenzamos declarando dos vectores “a” y “b”, donde el vector “a” se declara con todos sus componentes en 0, mientras que el vector “b” se declara con su componente “i” en 10 y el resto en 0. El ciclo for se prueba del siguiente modo, con el vector “a” inicializado todos sus componentes en 0, el ciclo se ejecutará hasta que el vector “a” sea mayor que el vector “b”, en cada vuelta de bucle el vector “a” se incrementará en 1 su componte “i”, mientras que la condición del ciclo for se cumpla, proseguiremos a mostrar el valor de los componentes del vector “a”. Como se puede observar, el bucle se repite 10 veces comenzando desde que el vector “a” todos sus componentes son 0, hasta que llega a que el componente “i” sea 9, cuando el componente “i” del vector “a” alcanza el 10, la condición, del ciclo for, no se cumple por lo cual sale y ya no se vuelve a mostrar el vector “a”.

### 3. Conclusiones

El manejo correcto de las instrucciones stop, stmt y, por supuesto de los tokens desde el archivo y, nos permiten poder realizar correctamente los saltos y el uso del ciclo for. Es bastante importante tomar en cuenta que la generación del código de dicho ciclo se hizo con base en el mapa de memoria, que fue el primer paso para realizar la esta práctica. Saber construir el mapa de memoria correspondiente al ciclo for, implica poder implementar el ciclo de manera correcta. Gracias a esta práctica, fue posible comprender con mayor claridad cómo funcionan los stop y su importancia, además de que ayudo a ver cómo generar ciclos de una manera diferente desde la calculadora.