



INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE COMPUTO



PRACTICA 6 CICLO FOR

Opción calculadora de vectores

COMPILADORES

Grupo: 3CM17

ALUMNO: MORA GUZMAN JOSE ANTONIO

FECHA ENTREGA: Miercoles 3 noviembre
2021

Descripción

En esta practica se añaden los ciclos for, que son ciclos que tienen 3 condiciones las cuales son: inicio, final y cantidad en que va aumentar la variable que controla el ciclo

Codigo

Se modifico la gramatica para añadir mas simbolos gramaticales

```
35 /**Creación de símbolos terminales y no terminales**/  
36 %token<comp> NUMBER  
37 %type<comp> escalar  
38  
39 %token<sym> VAR INDEF VECTOR NUMB  
40 %type<sym> vector number  
41  
42 %type<inst> exp asgn  
43  
44 %token<sym> PRINT WHILE IF ELSE BLTIN  
45 %type<inst> stmt stmtlst cond while if end  
46  
47 //Nuevos símbolos gramaticales para la práctica 6  
48 %token<sym> FOR  
49 %type<inst> for exprn  
50 /**Jerarquía de operadores**/
```

```
67 /**Gramática**/  
68 %%  
69  
70 list:  
71 | list '\n'  
72 | list asgn '\n' {code2(pop, STOP); return 1;}  
73 | list stmt '\n' {code(STOP); return 1;}  
74 | list exp '\n' {code2(print, STOP); return 1;}  
75 | list escalar '\n' {code2(prntd, STOP); return 1;}  
76 | list error '\n' {yyerror;}  
77 ;  
78  
79 asgn: VAR '=' exp {$$ = $3; code3(varpush, (Inst)$1, assign);}  
80 ;  
81  
82 exp: vector {$$ = code2(constpush, (Inst)$1);}  
83 | VAR {$$ = code3(varpush, (Inst)$1, eval);}  
84 | asgn  
85 | BLTIN '(' exp ')' {$$ = $3; code2(bltin, (Inst)$1 -> u.ptr);}  
86 | exp '+' exp {code(add);}  
87 | exp '-' exp {code(sub);}  
88 | escalar '*' exp {code(escalar);}  
89 | exp '*' escalar {code(escalar);}  
90 | exp '#' exp {code(producto_cruz);}  
91 | exp GT exp {code(mayor);}  
92 | exp LT exp {code(menor);}  
93 | exp GE exp {code(mayorIgual);}  
94 | exp LE exp {code(menorIgual);}  
95 | exp EQ exp {code(igual);}  
96 | exp NE exp {code(diferente);}  
97 | exp OR exp {code(or);}  
98 | exp AND exp {code(and);}  
99 | NOT exp {$$ = $2; code(not);}  
100 ;  
101
```

```

118     stmt: exp                                {code(pop);}
119     | PRINT exp                             {code(print); $$ = $2;}
120     | while cond stmt end                   {($1)[1] = (Inst)$3;
121                                             ($1)[2] = (Inst)$4;}
122
123     | if cond stmt end                     {($1)[1] = (Inst)$3;
124                                             ($1)[3] = (Inst)$4;}
125
126     | if cond stmt end ELSE stmt end       {($1)[1] = (Inst)$3;
127                                             ($1)[2] = (Inst)$6;
128                                             ($1)[3] = (Inst)$7;}
129
130     | for '(' exprn ';' exprn ';' exprn ')' stmt end {($1)[1] = (Inst)$5;
131                                                         ($1)[2] = (Inst)$7;
132                                                         ($1)[3] = (Inst)$9;
133                                                         ($1)[4] = (Inst)$10;}
134     | '{' stmtlst '}'                       {$$ = $2;}
135     ;

```

```

155 //PRÁCTICA 6
156 for: FOR                                {$$ = code(forcode); code3(STOP, STOP, STOP); code(STOP);}
157 ;
158
159 exprn: exp                             {$$ = $1; code(STOP);}
160 | '{' stmtlst '}'                     {$$ = $2;}
161 ;
162
163 %%
164

```

Y tambien se añadió en code.c una funcion que ejecuta la instrucción

```

269 /***** PRÁCTICA 6 *****/
270 void forcode(){
271     Datum d;
272     Inst* savepc = pc;
273     execute(savepc + 4);
274     execute(*((Inst **)(savepc)));
275     //Se saca la instrucción
276     d = pop();
277     while(d.val){
278         execute(* ( (Inst **)(savepc + 2))); /* Cuerpo del ciclo*/
279         execute(* ( (Inst **)(savepc + 1))); // Último campo
280         pop();
281         execute(*((Inst **)(savepc)));      /* CONDICION */
282         d = pop();
283     }
284     pc = *((Inst **)(savepc + 3)); /*Vamos a la siguiente posicion*/
285 }

```

Pruebas del programa

```
tony@tony-Aspire-E5-523: ~/Escritorio/compiladores/Practica 6
tony@tony-Aspire-E5-523:~/Escritorio/compiladores/Practica 6$ yacc -d vector_cal.y
vector_cal.y:33 parser name defined to default : "parse"
conflicts: 3 shift/reduce
tony@tony-Aspire-E5-523:~/Escritorio/compiladores/Practica 6$ gcc y.tab.c code.c vector_cal.c hoc.c init.c -lm -w
tony@tony-Aspire-E5-523:~/Escritorio/compiladores/Practica 6$ ./a.out
a=[ 0 0 0]
b=[20 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 ]
[ 10.000000 0.000000 0.000000 ]
[ 11.000000 0.000000 0.000000 ]
[ 12.000000 0.000000 0.000000 ]
[ 13.000000 0.000000 0.000000 ]
[ 14.000000 0.000000 0.000000 ]
[ 15.000000 0.000000 0.000000 ]
[ 16.000000 0.000000 0.000000 ]
[ 17.000000 0.000000 0.000000 ]
[ 18.000000 0.000000 0.000000 ]
[ 19.000000 0.000000 0.000000 ]
```

Conclusiones

Al igual que la practica anterior esta al ser una continuacion se vuelve cada vez mas sencillo el trabajar en las practicas y esta vez toco agregar un ciclo que personalmente yo uso demasiado y es el ciclo for y se le puede ver muchisima mas forma a nuestro compilador puesto que ya tiene muchas funcionalidades

Link del video

<https://youtu.be/oT1kFkwsZc0>