**Instituto Politécnico Nacional.**

**Escuela Superior De Cómputo.**





**Materia:**

**Compiladores.**

**Tema:**

**Reporte.**

**(Práctica 6).**

**Profesor:**

**Roberto Tecla Parra.**

**Alumno:**

**Mario Alberto Miranda Sandoval.**

**Grupo:**

**3CM7**

**Descripción.**

En esta sexta practica se ha implementado el ciclo for.

**Modificaciones al código.**

En primera instancia se añadieron los nuevos símbolos gramaticales para esta práctica.

1. %token<sym>     FOR
2. %type<inst>     **for** exprn

Posteriormente declaramos la gramática para el for, donde especificamos la estructura y funcionamiento del for, además de colocar las acciones gramaticales que pertenecen a el for.

Además, colocamos la gramática para el nuevo símbolo gramatical “exprn” que nos servirá para guardar “exp” de cada parte del for.

1. | **for** '(' exprn ';' exprn ';' exprn ')' stmt end    {($1)[1] = (Inst)$5;
2. ($1)[2] = (Inst)$7;
3. ($1)[3] = (Inst)$9;
4. ($1)[4] = (Inst)$10;}
5. **for**: FOR                            {$$ = code(forcode); code3(STOP, STOP, STOP); code(STOP);}
6. ;
8. exprn: exp                          {$$ = $1; code(STOP);}
9. |   '{' stmtlst '}'                 {$$ = $2;}
10. ;

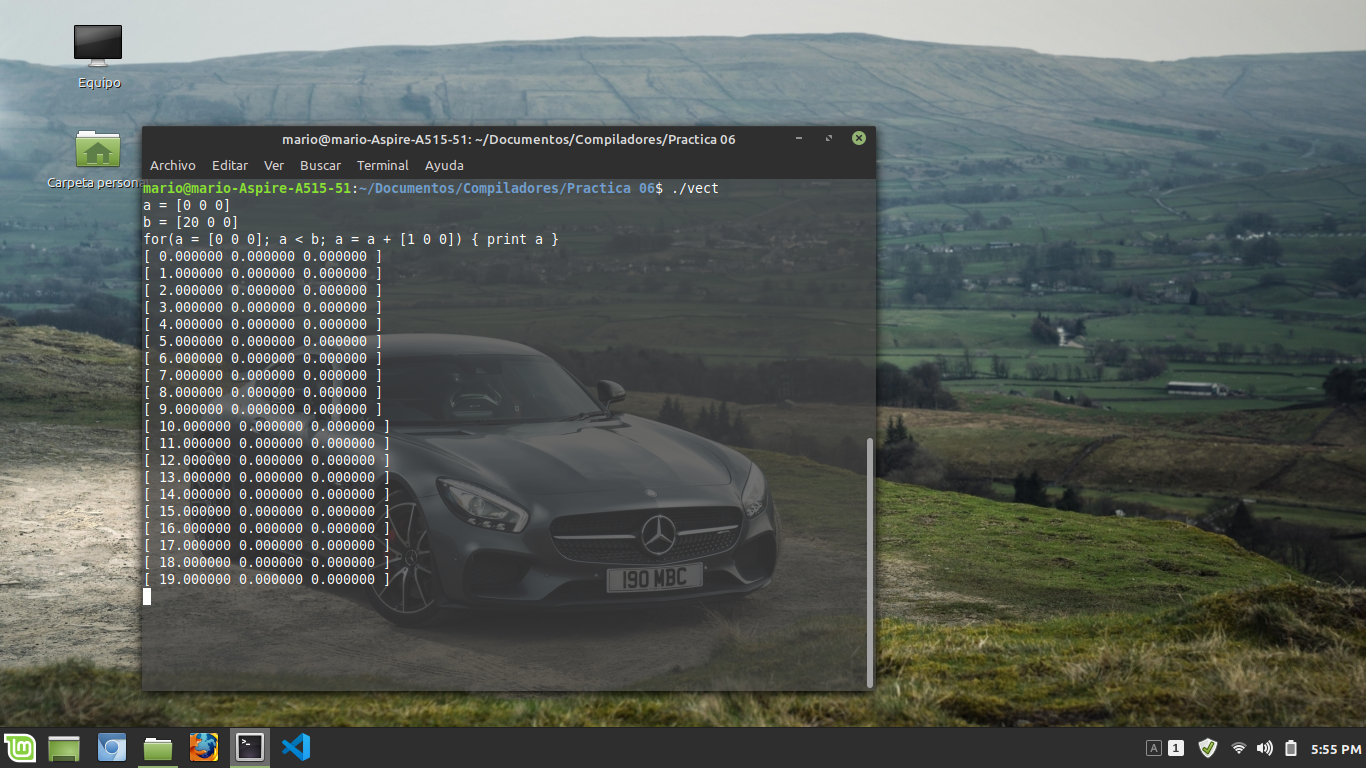
Y en el code.c colocamos la nueva función que es el forcode, donde primero sacamos la instrucción de la pila, ejecutamos el cuerpo del ciclo, al termino de este, ejecutamos el ultimo campo, al hacer esto, volvemos a hacer pop a la pila, por último, checamos la condición de paro y después sacamos otra instrucción de la pila, para finalizar, el contador del programa lo pasamos a la siguiente posición.

1. **void** forcode(){
2. Datum d;
3. Inst\* savepc = pc;
4. execute(savepc + 4);
5. execute(\*((Inst \*\*)(savepc)));
6. d = pop();
7. **while**(d.val){
8. execute(\* ( (Inst \*\*)(savepc + 2)));
9. execute(\* ( (Inst \*\*)(savepc + 1)));
10. pop();
11. execute(\*((Inst \*\*)(savepc)));
12. d = pop();
13. }
14. pc = \*((Inst \*\*)(savepc + 3));
15. }

**Pruebas.**

Se comienza 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 20 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 20 veces comenzando desde que el vector “a” todos sus componentes son 0, hasta que llega a que el componente “i” sea 19, cuando el componente “i” del vector “a” alcanza el 20, la condición, del ciclo for, no se cumple por lo cual sale y ya no se vuelve a mostrar el vector “a”.

**Código.**

**Vec\_cal.y**

1. %{
2. #include "hoc.h"
3. #include <math.h>
4. #include <stdio.h>
5. #define MSDOS
6. #define code2(c1, c2) code(c1); code(c2);
7. #define code3(c1, c2, c3) code(c1); code(c2); code(c3);
9. **void** yyerror(**char**\* s);
10. **int** yylex();
11. **void** warning(**char**\* s, **char**\* t);
12. **void** fpecatch();
13. **void** execerror(**char**\*s, **char**\* t);
14. **extern** **void** init();
15. %}
17. %**union**{
18. **double** comp;
19. Vector\* vec;
20. Symbol\* sym;
21. Inst\* inst;
22. **int** eval;
23. }
25. %token<comp> NUMBER
26. %type<comp> escalar
28. %token<sym> VAR INDEF VECTOR NUMB
29. %type<sym> vector number
31. %type<inst> exp asgn
33. %token<sym>     PRINT WHILE IF ELSE BLTIN
34. %type<inst>     stmt stmtlst cond **while** **if** end
36. %token<sym>     FOR
37. %type<inst>     **for** exprn
39. %right '='
40. %left OR AND
41. %left GT GE LT LE EQ NE
42. %left '+' '-'
43. %left '\*'
44. %left '#' '.' '|'
45. %left UNARYMINUS NOT
47. %%
49. list:
50. | list '\n'
51. | list asgn '\n'    {code2(pop, STOP); **return** 1;}
52. | list stmt '\n'    {code(STOP); **return** 1;}
53. | list exp '\n'     {code2(print, STOP); **return** 1;}
54. | list escalar '\n' {code2(printd, STOP); **return** 1;}
55. | list error '\n'   {yyerror;}
56. ;
58. asgn: VAR '=' exp   {$$ = $3; code3(varpush, (Inst)$1, assign);}
59. ;
61. exp: vector             {$$ = code2(constpush, (Inst)$1);}
62. | VAR                 {$$ = code3(varpush, (Inst)$1, eval);}
63. | asgn
64. | BLTIN '(' exp ')'   {$$ = $3; code2(bltin, (Inst)$1 -> u.ptr);}
65. | exp '+' exp         {code(add);}
66. | exp '-' exp         {code(sub);}
67. | escalar '\*' exp     {code(escalar);}
68. | exp '\*' escalar     {code(escalar);}
69. | exp '#' exp         {code(producto\_cruz);}
70. | exp GT exp          {code(mayor);}
71. | exp LT exp          {code(menor);}
72. | exp GE exp          {code(mayorIgual);}
73. | exp LE exp          {code(menorIgual);}
74. | exp EQ exp          {code(igual);}
75. | exp NE exp          {code(diferente);}
76. | exp OR exp          {code(or);}
77. | exp AND exp         {code(and);}
78. | NOT exp             {$$ = $2; code(not);}
79. ;
81. escalar: number         {code2(constpushd, (Inst)$1);}
82. | exp '.' exp         {code(producto\_punto);}
83. | '|' exp '|'         {code(magnitud);}
84. ;
86. vector: '[' NUMBER NUMBER NUMBER ']'   {  Vector\* v = creaVector(3);
87. v -> vec[0] = $2;
88. v -> vec[1] = $3;
89. v -> vec[2] = $4;
90. $$ = install("", VECTOR, v);}
91. ;

94. number: NUMBER                      {$$ = installd("", NUMB, $1);}
95. ;
97. stmt: exp                                           {code(pop);}
98. |   PRINT exp                                       {code(print); $$ = $2;}
99. |   **while** cond stmt end                             {($1)[1] = (Inst)$3;
100. ($1)[2] = (Inst)$4;}
102. | **if** cond stmt end                                  {($1)[1] = (Inst)$3;
103. ($1)[3] = (Inst)$4;}
105. | **if** cond stmt end ELSE stmt end                    {($1)[1] = (Inst)$3;
106. ($1)[2] = (Inst)$6;
107. ($1)[3] = (Inst)$7;}
109. | **for** '(' exprn ';' exprn ';' exprn ')' stmt end    {($1)[1] = (Inst)$5;
110. ($1)[2] = (Inst)$7;
111. ($1)[3] = (Inst)$9;
112. ($1)[4] = (Inst)$10;}
113. | '{' stmtlst '}'                                   {$$ = $2;}
114. ;
116. cond: '(' exp ')'                   {code(STOP); $$ = $2;}
117. ;
119. **while**: WHILE                        {$$ = code3(whilecode, STOP, STOP);}
120. ;
122. **if**: IF                              {$$ = code(ifcode);
123. code3(STOP, STOP, STOP);}
124. ;
126. end:                                {code(STOP); $$ = progp;}
127. ;
129. stmtlst:                            {$$ = progp;}
130. |   stmtlst '\n'
131. |   stmtlst stmt
132. ;
134. **for**: FOR                            {$$ = code(forcode); code3(STOP, STOP, STOP); code(STOP);}
135. ;
137. exprn: exp                          {$$ = $1; code(STOP);}
138. |   '{' stmtlst '}'                 {$$ = $2;}
139. ;
141. %%
143. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
144. \*                            Código en C                                    \*
145. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
146. #include <stdio.h>
147. #include <ctype.h>
148. #include <signal.h>
149. #include <setjmp.h>
151. **jmp\_buf** begin;
152. **char** \* progname;
153. **int** lineno = 1;
155. **void** main(**int** argc, **char** \* argv[]) {
156. progname = argv[0];
157. init();
158. setjmp(begin);
159. signal(SIGFPE, fpecatch);
160. **for**(initcode(); yyparse (); initcode())
161. execute(prog);
162. }
164. **void** execerror(**char** \* s, **char** \* t){
165. warning(s, t);
166. longjmp(begin, 0);
167. }
169. **void** fpecatch(){
170. execerror("Excepcion de punto flotante", (**char** \*)0);
171. }
173. **int** yylex(){
174. **int** c;
175. **while** ((c = getchar()) == ' ' || c == '\t');
177. **if** (c == EOF)
178. **return** 0;
180. **if** (isdigit(c) ) {
181. ungetc(c, stdin);
182. scanf("%lf\n", &yylval.comp);
183. **return** NUMBER;
184. }
186. **if** (isalpha(c)) {
187. Symbol\* s;
188. **char** sbuf[200];
189. **char**\* p = sbuf;
190. **do** {
191. \*p++=c;
192. } **while**((c = getchar()) != EOF && isalnum(c));
194. ungetc(c, stdin);
195. \*p = '\0';
196. **if** ((s = lookup(sbuf)) == (Symbol \*)NULL)
197. s = install(sbuf, INDEF, NULL);
198. yylval.sym = s;
200. **if** (s->type == INDEF)
201. **return** VAR;
202. **else**
203. **return** s->type;
204. }
206. **switch**(c){
207. **case** '>': **return** follow('=', GE, GT);
208. **case** '<': **return** follow('=', LE, LT);
209. **case** '=': **return** follow('=', EQ, '=');
210. **case** '!': **return** follow('=', NE, NOT);
211. **case** '|': **return** follow('|', OR, '|');
212. **case** '&': **return** follow('&', AND, '&');
213. **case** '\n': lineno++; **return** '\n';
214. **default**: **return** c;
215. }
216. lineno++;
217. **return** c;
218. }
220. **int** follow(**int** expect, **int** ifyes, **int** ifno){
221. **int** c  = getchar();
222. **if**  (c  ==  expect)
223. **return** ifyes;
224. ungetc(c,   stdin);
225. **return**  ifno;
226. }
228. **void** yyerror(**char** \* s){
229. warning(s, (**char** \*)0);
230. }
232. **void** warning(**char** \* s, **char** \* t) {
233. fprintf(stderr, "%s: %s",progname,s);
234. **if** (t)
235. fprintf(stderr, "%s",t);
236. fprintf(stderr, "Cerca de la linea %d\n",lineno);
237. }

**Code.c**

1. #include "hoc.h"
2. #include "y.tab.h"
3. #include <stdio.h>
4. #define NSTACK 256
5. **static** Datum stack[NSTACK];
6. **static** Datum\* stackp;
7. #define NPROG 2000
8. Inst prog[NPROG];
9. Inst\* progp;
10. Inst\* pc;
12. **void** initcode(){
13. stackp = stack;
14. progp = prog;
16. }
18. **void** push(d)
19. Datum d;
20. {     /\* Se mete d en la pila\*/
21. **if**( stackp >= &stack[NSTACK] )
22. execerror("stack overflow", (**char** \*) 0);
23. \*stackp++ = d;
24. }
26. Datum pop(){
27. **if**( stackp <= stack )
28. execerror("stack underflow", (**char** \*) 0);
29. **return** \*--stackp;
30. }
32. **void** constpush(){
33. Datum d;
34. d.val = ((Symbol \*)\*pc++)->u.vec;
36. push(d);
37. }
39. **void** constpushd(){
40. Datum d;
41. d.num = ((Symbol  \*)\*pc++)->u.comp;
43. push(d);
44. }
46. **void** varpush(){
47. Datum d;
48. d.sym = (Symbol  \*)(\*pc++);
50. push(d);
51. }
53. **void** eval( ){
54. Datum d;
55. d = pop();
56. **if**( d.sym->type == INDEF )
57. execerror("undefined variable",d.sym->name);
58. d.val = d.sym->u.vec;
59. push(d);
60. }
62. **void** add(){
63. Datum d1, d2;
64. d2 = pop();
65. d1 = pop();
66. d1.val = sumaVector(d1.val, d2.val);
67. push(d1);
68. }
70. **void** sub(){
71. Datum d1, d2;
72. d2 = pop();
73. d1 = pop();
74. d1.val = restaVector(d1.val, d2.val);
75. push(d1);
76. }
78. **void** escalar(){
79. Datum d1, d2;
80. d2 = pop();
81. d1 = pop();
82. d1.val = escalarVector(d1.num, d2.val);
83. push(d1);
84. }
86. **void** producto\_punto(){
87. Datum d1, d2;
88. **double** d3;
89. d2 = pop();
90. d1 = pop();
91. d3 = productoPunto(d1.val, d2.val);
92. push((Datum)d3);
93. }
95. **void** producto\_cruz(){
96. Datum d1, d2;
97. d2 = pop();
98. d1 = pop();
99. d1.val = productoCruz(d1.val, d2.val);
100. push(d1);
101. }
103. **void** magnitud(){
104. Datum d1;
105. d1 = pop();
106. d1.num = vectorMagnitud(d1.val);
107. push(d1);
108. }
110. **void** assign( ){
111. Datum d1, d2;
112. d1 = pop();
113. d2 = pop();
114. **if**(d1.sym->type != VAR && d1.sym->type != INDEF)
115. execerror("assignment to non-variable", d1.sym->name);
116. d1.sym->u.vec = d2.val;
117. d1.sym->type = VAR;
118. push(d2);
119. }
121. **void** print(){
122. Datum d;
123. d = pop();
125. imprimeVector(d.val);
126. }
128. **void** printd(){
129. Datum d;
130. d = pop();
131. printf("%lf\n",d.num);
132. }
134. Inst \*code(Inst f){
135. Inst \*oprogp = progp;
136. **if** (progp >= &prog [ NPROG ])
137. execerror("program too big", (**char** \*) 0);
138. \*progp++ = f;
140. **return** oprogp;
141. }
143. **void** execute( Inst\* p){
144. **for**( pc = p; \*pc != STOP; )
145. (\*(\*pc++))();
147. }
149. **void** mayor(){
150. Datum d1, d2;
151. d2 = pop();
152. d1 = pop();
153. d1.num = (**int**)( vectorMagnitud(d1.val) > vectorMagnitud(d2.val) );
154. push(d1);
155. }
157. **void** menor(){
158. Datum d1, d2;
159. d2 = pop();
160. d1 = pop();
161. d1.num = (**int**)( vectorMagnitud(d1.val) < vectorMagnitud(d2.val) );
162. push(d1);
163. }
165. **void** mayorIgual(){
166. Datum d1, d2;
167. d2 = pop();
168. d1 = pop();
169. d1.num = (**int**)( vectorMagnitud(d1.val) >= vectorMagnitud(d2.val) );
170. push(d1);
171. }
173. **void** menorIgual(){
174. Datum d1, d2;
175. d2 = pop();
176. d1 = pop();
177. d1.num = (**int**)( vectorMagnitud(d1.val) <= vectorMagnitud(d2.val) );
178. push(d1);
179. }
181. **void** igual(){
182. Datum d1, d2;
183. d2 = pop();
184. d1 = pop();
185. d1.num = (**int**)( vectorMagnitud(d1.val) == vectorMagnitud(d2.val) );
186. push(d1);
187. }
189. **void** diferente(){
190. Datum d1, d2;
191. d2 = pop();
192. d1 = pop();
193. d1.num = (**int**)( vectorMagnitud(d1.val) != vectorMagnitud(d2.val) );
194. push(d1);
195. }
197. **void** and(){
198. Datum d1, d2;
199. d2 = pop();
200. d1 = pop();
201. d1.num = (**int**)( vectorMagnitud(d1.val) && vectorMagnitud(d2.val) );
202. push(d1);
203. }
205. **void** or(){
206. Datum d1, d2;
207. d2 = pop();
208. d1 = pop();
209. d1.num = (**int**)( vectorMagnitud(d1.val) || vectorMagnitud(d2.val) );
210. push(d1);
211. }
213. **void** not(){
214. Datum d1;
215. d1 = pop();
216. d1.num = (**int**)( vectorMagnitud(d1.val) == (**double**)0.0);
217. push(d1);
218. }
220. **void** whilecode(){
221. Datum d;
222. Inst\* savepc = pc;
223. execute(savepc + 2);
224. d = pop();
225. **while**(d.val){
226. execute(\* ( (Inst \*\*)(savepc) ));
227. execute(savepc + 2);
228. d = pop();
229. }
230. pc = \*((Inst \*\*)(savepc + 1));
231. }
233. **void** ifcode(){
234. Datum d;
235. Inst\* savepc = pc;
236. execute(savepc + 3);
237. d = pop();
238. **if**(d.val)
239. execute(\*((Inst \*\*)(savepc)));
240. **else** **if**(\*((Inst \*\*)(savepc + 1)))
241. execute(\*((Inst \*\*)(savepc + 1)));
242. pc = \*((Inst \*\*)(savepc + 2));
243. }
245. **void** bltin(){
246. Datum d;
247. d = pop();
248. d.val = (\*(Vector \* (\*)() )(\*pc++))(d.val);
249. push(d);
250. }
252. **void** forcode(){
253. Datum d;
254. Inst\* savepc = pc;
255. execute(savepc + 4);
256. execute(\*((Inst \*\*)(savepc)));
257. d = pop();
258. **while**(d.val){
259. execute(\* ( (Inst \*\*)(savepc + 2)));
260. execute(\* ( (Inst \*\*)(savepc + 1)));
261. pop();
262. execute(\*((Inst \*\*)(savepc)));
263. d = pop();
264. }
265. pc = \*((Inst \*\*)(savepc + 3));
266. }