



Instituto Politécnico Nacional.
Escuela Superior De Cómputo.



Materia:
Compiladores.

Tema:
Reporte.
(Práctica 5).

Profesor:
Roberto Tecla Parra.

Alumno:
Mario Alberto Miranda Sandoval.

Grupo:
3CM7.

Descripción.

En esta quinta práctica se han añadido pequeños fragmentos de código, en concreto se han añadido sentencias IF, ELSE, WHILE, obviamente operadores lógicos, como:

- Mayor
- Mayor igual
- Menor
- Menor igual
- Diferente
- OR
- AND
- NOT

Para poder aplicar esto a vectores calculamos las magnitudes de los vectores.

Modificaciones al código.

Primero añadimos los nuevos símbolos gramaticales.

```
1. %token<sym>      PRINT WHILE IF ELSE BLTIN
2. %type<inst>      stmt stmtlst cond while if end
```

Posteriormente colocamos la precedencia de operadores que nos permitirá hacer las comparaciones y las operaciones binarias, resaltando el unaryminus que nos da prioridad más alta.

```
1. %left OR AND
2. %left GT GE LT LE EQ NE
3. %left UNARYMINUS NOT
```

Ahora añadimos las producciones gramaticales pertinentes para la práctica 5.

```
1. | exp GT exp          {code(mayor);}
2. | exp LT exp          {code(menor);}
3. | exp GE exp          {code(mayorIgual);}
4. | exp LE exp          {code(menorIgual);}
5. | exp EQ exp          {code(igual);}
6. | exp NE exp          {code(diferente);}
7. | exp OR exp          {code(or);}
8. | exp AND exp         {code(and);}
9. | NOT exp             {$$ = $2; code(not);}
10. stmt: exp            { code(pop); }
11. | PRINT exp          {code(print); $$ = $2;}
12. | while cond stmt end { ($1)[1] = (Inst)$3;
```

```

13.                                     ($1)[2] = (Inst)$4;}
14. | if cond stmt end                 { ($1)[1] = (Inst)$3;
15.                                     ($1)[3] = (Inst)$4;}
16. | if cond stmt end ELSE stmt end  {($1)[1] = (Inst)$3;
17.                                     ($1)[2] = (Inst)$6;
18.                                     ($1)[3] = (Inst)$7;}
19. | '{' stmtlst '}'                 {$$ = $2;}
20. ;
21.
22. cond: '(' exp ')'                   {code(STOP); $$ = $2;}
23. ;
24.
25. while: WHILE                        {$$ = code3(whilecode, STOP, STOP);}
26. ;
27.
28. if: IF                              {$$ = code(ifcode);
29.                                     code3(STOP, STOP, STOP);}
30. ;

```

Ahora como se va a evaluar operadores lógicos a yylex se han añadido el siguiente código.

```

1. switch(c){
2.     case '>': return follow('=', GE, GT);
3.     case '<': return follow('=', LE, LT);
4.     case '=': return follow('=', EQ, '=');
5.     case '!': return follow('=', NE, NOT);
6.     case '|': return follow('|', OR, '|');
7.     case '&': return follow('&', AND, '&');
8.     case '\n': lineno++; return '\n';
9.     default: return c;
10. }

```

Seguido se añadió una nueva función que será la encargada de buscar operadores.

```

1. int follow(int expect, int ifyes, int ifno){
2.     int c = getchar();
3.     if (c == expect)
4.         return ifyes;
5.     ungetc(c, stdin);
6.     return ifno;
7. }

```

En code.c también se han añadido nuevas funciones, estas son para los condicionales.

```

1. void mayor(){
2.     Datum d1, d2;
3.     d2 = pop();
4.     d1 = pop();
5.     d1.num = (int)( vectorMagnitud(d1.val) > vectorMagnitud(d2.val) );
6.     push(d1);
7. }
8.
9. void menor(){
10.     Datum d1, d2;

```

```

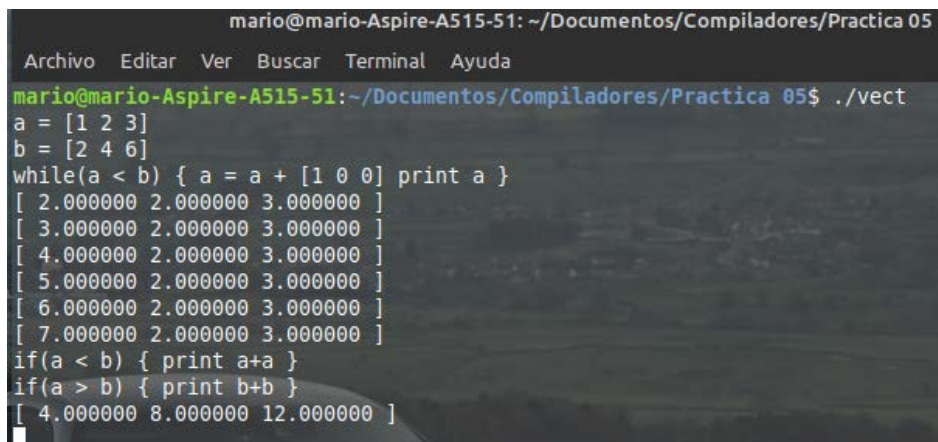
11.     d2 = pop();
12.     d1 = pop();
13.     d1.num = (int)( vectorMagnitud(d1.val) < vectorMagnitud(d2.val) );
14.     push(d1);
15. }
16.
17. void mayorIgual(){
18.     Datum d1, d2;
19.     d2 = pop();
20.     d1 = pop();
21.     d1.num = (int)( vectorMagnitud(d1.val) >= vectorMagnitud(d2.val) );
22.     push(d1);
23. }
24.
25. void menorIgual(){
26.     Datum d1, d2;
27.     d2 = pop();
28.     d1 = pop();
29.     d1.num = (int)( vectorMagnitud(d1.val) <= vectorMagnitud(d2.val) );
30.     push(d1);
31. }
32.
33. void igual(){
34.     Datum d1, d2;
35.     d2 = pop();
36.     d1 = pop();
37.     d1.num = (int)( vectorMagnitud(d1.val) == vectorMagnitud(d2.val) );
38.     push(d1);
39. }
40.
41. void diferente(){
42.     Datum d1, d2;
43.     d2 = pop();
44.     d1 = pop();
45.     d1.num = (int)( vectorMagnitud(d1.val) != vectorMagnitud(d2.val) );
46.     push(d1);
47. }
48.
49. void and(){
50.     Datum d1, d2;
51.     d2 = pop();
52.     d1 = pop();
53.     d1.num = (int)( vectorMagnitud(d1.val) && vectorMagnitud(d2.val) );
54.     push(d1);
55. }
56.
57. void or(){
58.     Datum d1, d2;
59.     d2 = pop();
60.     d1 = pop();
61.     d1.num = (int)( vectorMagnitud(d1.val) || vectorMagnitud(d2.val) );
62.     push(d1);
63. }
64.
65. void not(){
66.     Datum d1;
67.     d1 = pop();
68.     d1.num = (int)( vectorMagnitud(d1.val) == (double)0.0 );
69.     push(d1);
70. }

```

Y para el while e if.

```
1. void whilecode(){
2.     Datum d;
3.     Inst* savepc = pc;
4.     execute(savepc + 2);
5.     d = pop();
6.     while(d.val){
7.         execute(* ( (Inst **)(savepc) ));
8.         execute(savepc + 2);
9.         d = pop();
10.    }
11.    pc = *((Inst **)(savepc + 1));
12. }
13.
14. void ifcode(){
15.     Datum d;
16.     Inst* savepc = pc;
17.     execute(savepc + 3);
18.     d = pop();
19.     if(d.val)
20.         execute(*((Inst **)(savepc)));
21.     else if(*((Inst **)(savepc + 1)))
22.         execute(*((Inst **)(savepc + 1)));
23.     pc = *((Inst **)(savepc + 2));
24. }
25.
26. void bltin(){
27.     Datum d;
28.     d = pop();
29.     d.val = (*(Vector * (*)() )(*pc++))(d.val);
30.     push(d);
31. }
```

Pruebas .



```
mario@mario-Aspire-A515-51: ~/Documentos/Compiladores/Practica 05
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
mario@mario-Aspire-A515-51:~/Documentos/Compiladores/Practica 05$ ./vect
a = [1 2 3]
b = [2 4 6]
while(a < b) { a = a + [1 0 0] print a }
[ 2.000000 2.000000 3.000000 ]
[ 3.000000 2.000000 3.000000 ]
[ 4.000000 2.000000 3.000000 ]
[ 5.000000 2.000000 3.000000 ]
[ 6.000000 2.000000 3.000000 ]
[ 7.000000 2.000000 3.000000 ]
if(a < b) { print a+a }
if(a > b) { print b+b }
[ 4.000000 8.000000 12.000000 ]
```

Como se puede ver, primero declaramos dos vectores, después vamos incrementando y mostrando el vector "a" mientras este

sea menor en magnitud que "b", notamos que el vector "a" va actualizando su valor.

Para probar los condicionales, primero probamos si el vector "a" es menor al vector "b", si es así mostramos una suma del vector "a" consigo mismo, vemos que como actualizamos (debido a los incrementos en el while) el valor del vector "a", la condición no se cumple por lo que no se ejecuta la acción.

Mientras que si probamos al revés vemos que ejecuta la instrucción que se encuentra dentro del if, que en este caso es mostrar la suma del vector "b", consigo mismo.

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);
8.
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.}%
16.
17.%union{
18.    double comp;
19.    Vector* vec;
20.    Symbol* sym;
21.    Inst* inst;
22.    int eval;
23.}
24.
25.%token<comp> NUMBER
26.%type<comp> escalar
27.
28.%token<sym> VAR INDEF VECTOR NUMB
29.%type<sym> vector number
30.
31.%type<inst> exp asgn
32.
33.%token<sym>      PRINT WHILE IF ELSE BLTIN
34.%type<inst>      stmt stmtlst cond while if end
35.
36.%right '='
37.%left OR AND
38.%left GT GE LT LE EQ NE
39.%left '+' '-'
40.%left '*'
```

```

41. %left '#' '.' ' ' '|'
42. %left UNARYMINUS NOT
43.
44. %%
45.
46. list:
47. | list '\n'
48. | list asgn '\n' {code2(pop, STOP); return 1;}
49. | list stmt '\n' {code(STOP); return 1;}
50. | list exp '\n' {code2(print, STOP); return 1;}
51. | list escalar '\n' {code2(printd, STOP); return 1;}
52. | list error '\n' {yyerror;}
53. ;
54.
55. asgn: VAR '=' exp {$$ = $3; code3(varpush, (Inst)$1, assign);}
56. ;
57.
58. exp: vector {code2(constpush, (Inst)$1);}
59. | VAR {code3(varpush, (Inst)$1, eval);}
60. | asgn
61. | BLTIN '(' exp ')' {$$ = $3; code2(bltin, (Inst)$1 -> u.ptr);}
62. | exp '+' exp {code(add);}
63. | exp '-' exp {code(sub);}
64. | escalar '*' exp {code(escalar);}
65. | exp '*' escalar {code(escalar);}
66. | exp '#' exp {code(producto_cruz);}
67. | exp GT exp {code(mayor);}
68. | exp LT exp {code(menor);}
69. | exp GE exp {code(mayorIgual);}
70. | exp LE exp {code(menorIgual);}
71. | exp EQ exp {code(igual);}
72. | exp NE exp {code(diferente);}
73. | exp OR exp {code(or);}
74. | exp AND exp {code(and);}
75. | NOT exp {$$ = $2; code(not);}
76. ;
77.
78. escalar: number {code2(constpushd, (Inst)$1);}
79. | exp '.' exp {code(producto_punto);}
80. | '|' exp '|' {code(magnitud);}
81. ;
82.
83. vector: '[' NUMBER NUMBER NUMBER ']' { Vector* v = creaVector(3);
84. v -> vec[0] = $2;
85. v -> vec[1] = $3;
86. v -> vec[2] = $4;
87. $$ = install("", VECTOR, v);}
88. ;
89.
90. number: NUMBER {$$ = installd("", NUMB, $1);}
91. ;
92.
93. //Para la práctica 5
94. stmt: exp { code(pop); }
95. | PRINT exp {code(print); $$ = $2;}
96. | while cond stmt end { ($1)[1] = (Inst)$3;
97. ($1)[2] = (Inst)$4;}
98. | if cond stmt end { ($1)[1] = (Inst)$3;
99. ($1)[3] = (Inst)$4;}
100. | if cond stmt end ELSE stmt end {($1)[1] = (Inst)$3;
101. ($1)[2] = (Inst)$6;

```

```

102.                                     ($1)[3] = (Inst)$7;}
103.         | '{' stmtlst '}'         {$$ = $2;}
104.         ;
105.
106.         cond: '(' exp ')'         {code(STOP); $$ = $2;}
107.         ;
108.
109.         while: WHILE               {$$ = code3(whilecode, STOP, STOP)}
        ;}
110.         ;
111.
112.         if: IF                     {$$ = code(ifcode);
113.                                     code3(STOP, STOP, STOP);}
114.         ;
115.
116.         end:                       {code(STOP); $$ = prog;}
117.         ;
118.
119.         stmtlst:                   {$$ = prog;}
120.         | stmtlst '\n'
121.         | stmtlst stmt
122.         ;
123.
124.         %%
125.
126.         /*****
***
127.         *                               Código en C
        *
128.         *****/
**/
129.         #include <stdio.h>
130.         #include <ctype.h>
131.         #include <signal.h>
132.         #include <setjmp.h>
133.
134.         jmp_buf begin;
135.         char * progname;
136.         int lineno = 1;
137.
138.         void main(int argc, char * argv[]) {
139.             progname = argv[0];
140.             init();
141.             setjmp(begin);
142.             signal(SIGFPE, fpecatch);
143.             for(initcode(); yyparse (); initcode())
144.                 execute(prog);
145.         }
146.
147.         void execerror(char * s, char * t){
148.             warning(s, t);
149.             longjmp(begin, 0);
150.         }
151.
152.         void fpecatch(){
153.             execerror("Excepcion de punto flotante", (char *)0);
154.         }
155.
156.         int yylex(){
157.             int c;
158.             while ((c = getchar()) == ' ' || c == '\t')

```



```

159.         /**SALTA BLANCOS**/;
160.
161.         if (c == EOF)
162.             return 0;
163.
164.         if (isdigit(c) ) {
165.             ungetc(c, stdin);
166.             scanf("%lf\n", &yyval.comp);
167.             return NUMBER;
168.         }
169.
170.         if (isalpha(c)) {
171.             Symbol* s;
172.             char sbuf[200];
173.             char* p = sbuf;
174.             do {
175.                 *p++=c;
176.             } while((c = getchar()) != EOF && isalnum(c));
177.
178.             ungetc(c, stdin);
179.             *p = '\0';
180.             if ((s = lookup(sbuf)) == (Symbol *)NULL)
181.                 s = install(sbuf, INDEF, NULL);
182.             yyval.sym = s;
183.
184.             if (s->type == INDEF)
185.                 return VAR;
186.             else
187.                 return s->type;
188.         }
189.
190.         switch(c){
191.             case '>': return follow('=', GE, GT);
192.             case '<': return follow('=', LE, LT);
193.             case '=': return follow('=', EQ, '=');
194.             case '!': return follow('=', NE, NOT);
195.             case '|': return follow('|', OR, '|');
196.             case '&': return follow('&', AND, '&');
197.             case '\n': lineno++; return '\n';
198.             default: return c;
199.         }
200.     }
201.
202.     int follow(int expect, int ifyes, int ifno){
203.         int c = getchar();
204.         if (c == expect)
205.             return ifyes;
206.         ungetc(c, stdin);
207.         return ifno;
208.     }
209.
210.     void yyerror(char * s){
211.         warning(s, (char *)0);
212.     }
213.
214.     void warning(char * s, char * t) {
215.         fprintf(stderr, "%s: %s", progname, s);
216.         if (t)
217.             fprintf(stderr, "%s", t);
218.         fprintf(stderr, "Cerca de la linea %d\n", lineno);
219.     }

```

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.
11. Inst* pc;
12.
13. void initcode(){
14.     stackp = stack;
15.     progp = prog;
16. }
17.
18. void push(d) {
19.     Datum d;
20.     if( stackp >= &stack[NSTACK] )
21.         execerror("stack overflow", (char *) 0);
22.     *stackp++ = d;
23. }
24.
25. Datum pop(){
26.     if( stackp <= stack )
27.         execerror("stack underflow", (char *) 0);
28.     return *--stackp;
29. }
30.
31. void constpush(){
32.     Datum d;
33.     d.val = ((Symbol *)*pc++)->u.vec;
34.
35.     push(d);
36. }
37.
38. void constpushd(){
39.     Datum d;
40.     d.num = ((Symbol *)*pc++)->u.comp;
41.
42.     push(d);
43. }
44.
45. void varpush(){
46.     Datum d;
47.
48.     d.sym = (Symbol *)(*pc++);
49.
50.     push(d);
51. }
52.
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. }
61.
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. }
69.
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. }
77.
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. }
85.
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. }
94.
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.    }
102.
103.    void magnitud(){
104.        Datum d1;
105.        d1 = pop();
106.        d1.num = vectorMagnitud(d1.val);
107.        push(d1);
108.    }
109.
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.    }

```

```

120.
121.     void print(){
122.         Datum d;
123.         d = pop();
124.
125.         imprimeVector(d.val);
126.     }
127.
128.     void printd(){
129.         Datum d;
130.         d = pop();
131.         printf("%lf\n",d.num);
132.     }
133.
134.     Inst *code(Inst f){
135.         Inst *oprogp = progp;
136.         if (progp >= &prog [ NPROG ])
137.             execerror("program too big", (char *) 0);
138.         *progp++ = f;
139.
140.         return oprogp;
141.     }
142.
143.     void execute( Inst* p){
144.         for( pc = p; *pc != STOP; )
145.
146.             ((*pc++))();
147.     }
148.
149.
150.
151.     void mayor(){
152.         Datum d1, d2;
153.         d2 = pop();
154.         d1 = pop();
155.         d1.num = (int)( vectorMagnitud(d1.val) > vectorMagnitud(d2.val) );
156.         push(d1);
157.     }
158.
159.     void menor(){
160.         Datum d1, d2;
161.         d2 = pop();
162.         d1 = pop();
163.         d1.num = (int)( vectorMagnitud(d1.val) < vectorMagnitud(d2.val) );
164.         push(d1);
165.     }
166.
167.     void mayorIgual(){
168.         Datum d1, d2;
169.         d2 = pop();
170.         d1 = pop();
171.         d1.num = (int)( vectorMagnitud(d1.val) >= vectorMagnitud(d2.val) );
172.         push(d1);
173.     }
174.
175.     void menorIgual(){
176.         Datum d1, d2;
177.         d2 = pop();
178.         d1 = pop();
179.         d1.num = (int)( vectorMagnitud(d1.val) <= vectorMagnitud(d2.val) );
180.         push(d1);

```

```

181.     }
182.
183.     void igual(){
184.         Datum d1, d2;
185.         d2 = pop();
186.         d1 = pop();
187.         d1.num = (int)( vectorMagnitud(d1.val) == vectorMagnitud(d2.val) );
188.         push(d1);
189.     }
190.
191.     void diferente(){
192.         Datum d1, d2;
193.         d2 = pop();
194.         d1 = pop();
195.         d1.num = (int)( vectorMagnitud(d1.val) != vectorMagnitud(d2.val) );
196.         push(d1);
197.     }
198.
199.     void and(){
200.         Datum d1, d2;
201.         d2 = pop();
202.         d1 = pop();
203.         d1.num = (int)( vectorMagnitud(d1.val) && vectorMagnitud(d2.val) );
204.         push(d1);
205.     }
206.
207.     void or(){
208.         Datum d1, d2;
209.         d2 = pop();
210.         d1 = pop();
211.         d1.num = (int)( vectorMagnitud(d1.val) || vectorMagnitud(d2.val) );
212.         push(d1);
213.     }
214.
215.     void not(){
216.         Datum d1;
217.         d1 = pop();
218.         d1.num = (int)( vectorMagnitud(d1.val) == (double)0.0);
219.         push(d1);
220.     }
221.
222.     void whilecode(){
223.         Datum d;
224.         Inst* savepc = pc;
225.         execute(savepc + 2);
226.         d = pop();
227.         while(d.val){
228.             execute(* ( (Inst **)(savepc) ));
229.             execute(savepc + 2);
230.             d = pop();
231.         }
232.         pc = *((Inst **)(savepc + 1));
233.     }
234.
235.     void ifcode(){
236.         Datum d;
237.         Inst* savepc = pc;
238.         execute(savepc + 3);
239.         d = pop();
240.         if(d.val)
241.             execute(*((Inst **)(savepc)));

```

```
242.         else if(*((Inst **)(savepc + 1)))
243.             execute(*((Inst **)(savepc + 1)));
244.         pc = *((Inst **)(savepc + 2));
245.     }
246.
247.     void bltin(){
248.         Datum d;
249.         d = pop();
250.         d.val = (*(Vector * (*)() )(*pc++))(d.val);
251.         push(d);
252.     }
```