Compiladores

- Práctica 05: if-else-while -

Grupo 3CM7

Vargas Romero Erick Efraín
Prof. Tecla Parra Roberto

Instituto Politécnico Nacional
Escuela Superior de Cómputo
Juan de Dios Bátiz, nueva industrial Vallejo
07738 ciudad de México

Chapter 1

Práctica 05

1.1 IF-ELSE-WHILE

1.1.1 Descripción

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

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

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

1.1.2 Ejemplos

A continuación muestro una captura de pantalla, la cual muestra la compilación del código en yacc, y también la compilación del código que es generado en c y finalmente la ejecución del programa.

Figure 1.1: Ejemplo ciclos y condiciones

```
[erick@erick-pc Práctica 05]$ ./a.out
a = [1 0 0]
b = [0 10 0]
if(a < b) { print a+a }
[ 2.000000 0.000000 0.000000 ]
while(a < b) { a = a + [1 0 0] print a }
[ 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 ]
```

1.1.3 Código

Nuevamente se ha modificado la gramática, se han añadido más símbolos gramaticales, y además se ha añadido un elemento a la definición de tipos de dato de YACC a continuación se muestran esas modificaciones

```
%union{
     double comp;
     Vector* vec;
     Symbol* sym;
     Inst* inst;
     //ñAadida en la áprctica 5
     int eval;
}

/**óCreacin de ísmbolos terminales y no terminales**/
%token<comp> NUMBER
%type<comp> escalar

%token<sym> VAR INDEF VECTOR NUMB
%type<sym> vector number

%type<inst> exp asgn

//NUEVOS ÍSMBOLOS GRAMATICALES PARA LA ÁPRCTICA 5
%token<sym>     PRINT WHILE IF ELSE BLTIN
%type<inst>     stmt stmtlst cond while if end

/**íJerarqua de operadores**/

//Para áprctica 3
%right '='
//Para la áprctica 5
%left OR AND
%left GT GE LT LE EQ NE
//íSmbolos gramaticales de la áprctica 1
//Suma y resta de vectores
```

```
32 %left '+' '-'
33 //Escalar por un vector
34 %left '*'
35 //Producto cruz y producto punto
36 %left '#' '.' '|'
37 //Para la áprctica 5
38 %left UNARYMINUS NOT
39
40 /**áGramtica**/
41 %%
42
43     list:
44       | list '\n'
45       | list asgn '\n'     {code2(pop, STOP); return 1;}
46       | list stmt '\n'     {code(STOP); return 1;}    //ñAadida en la
           áprctica 5
47       | list exp '\n'      {code2(print, STOP); return 1;}
48       | list escalar '\n' {code2(printd, STOP); return 1;}
49       | list error '\n'    {yyerror;}
50       ;
51
52     asgn: VAR '=' exp    {$$ = $3; code3(varpush, (Inst)$1, assign);}
53     ;
54
55     exp: vector              {code2(constpush, (Inst)$1);}
56       | VAR                  {code3(varpush, (Inst)$1, eval);}
57       | asgn
58       | BLTIN '(' exp ')'    {$$ = $3; code2(bltin, (Inst)$1 -> u.ptr);}
59       | exp '+' exp          {code(add);}
60       | exp '-' exp          {code(sub);}
61       | escalar '*' exp      {code(escalar);}
62       | exp '*' escalar      {code(escalar);}
63       | exp '#' exp          {code(producto_cruz);}
64       //Para la áprctica 5
65       | exp GT exp           {code(mayor);}
66       | exp LT exp           {code(menor);}
67       | exp GE exp           {code(mayorIgual);}
68       | exp LE exp           {code(menorIgual);}
69       | exp EQ exp           {code(igual);}
70       | exp NE exp           {code(diferente);}
71       | exp OR exp           {code(or);}
72       | exp AND exp          {code(and);}
73       | NOT exp              {$$ = $2; code(not);}
74     ;
75
76     escalar: number          {code2(constpushd, (Inst)$1);}
77       | exp '.' exp          {code(producto_punto);}
78       | '|' exp '|'          {code(magnitud);}
79       ;
80
81     vector: '[' NUMBER NUMBER NUMBER ']'    {  Vector* v = creaVector(3);
82                                                v -> vec[0] = $2;
83                                                v -> vec[1] = $3;
```

```
84                                                  v -> vec[2] = $4;
85                                                  $$ = install("", VECTOR, v)
                                                        ;}
86          ;
87
88      //Para la áprctica 4
89      number: NUMBER                              {$$ = installd("", NUMB, $1);}
90      ;
91
92      //Para la áprctica 5
93      stmt: exp                                   { code(pop); }
94      |    PRINT exp                              {code(print); $$ = $2;}
95      |    while cond stmt end                    { ($1)[1] = (Inst)$3;
96                                                    ($1)[2] = (Inst)$4;}
97      | if cond stmt end                          { ($1)[1] = (Inst)$3;
98                                                    ($1)[3] = (Inst)$4;}
99      | if cond stmt end ELSE stmt end            {($1)[1] = (Inst)$3;
100                                                 ($1)[2] = (Inst)$6;
101                                                 ($1)[3] = (Inst)$7;}
102     | '{' stmtlst '}'                           {$$ = $2;}
103     ;
104
105     cond: '(' exp ')'                           {code(STOP); $$ = $2;}
106     ;
107
108     while: WHILE                                {$$ = code3(whilecode, STOP,
            STOP);}
109     ;
110
111     if: IF                                      {$$ = code(ifcode);
112                                                  code3(STOP, STOP, STOP);}
113     ;
114
115     end: /* NADA */                            {code(STOP); $$ = progp;}
116     ;
117
118     stmtlst: /* NADA */                         {$$ = progp;}
119     |    stmtlst '\n'
120     |    stmtlst stmt
121     ;
122
123 %%
```

Como se van a evaluar operadores lógicos también se ha añadido algo de código a yylex

```
1       //ñAadido para la áprctica 5
2       switch(c){
3           case '>': return follow('=', GE, GT);
4           case '<': return follow('=', LE, LT);
5           case '=': return follow('=', EQ, '=');
6           case '!': return follow('=', NE, NOT);
7           case '|': return follow('|', OR, '|');
8           case '&': return follow('&', AND, '&');
```

```
9           case '\n': lineno++; return '\n';
10          default: return c;
11      }
```

Y también se ha añadido una función más

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

También se han añadido algunas funciones a code.c

```
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  }
71  /********* Ciclos *********/
72  void whilecode(){
73      Datum d;
74      Inst* savepc = pc;      /* Cuerpo de la óiteracin */
75      execute(savepc + 2);     /* óCondicin */
76      d = pop();
77      while(d.val){
78          execute(* ( (Inst **)(savepc) ));    /* Cuerpo del ciclo*/
79          execute(savepc + 2);
80          d = pop();
81      }
82      pc = *((Inst **)(savepc + 1)); /*Vamos a la siguiente posicion*/
83  }
84
85  void ifcode(){
86      Datum d;
87      Inst* savepc = pc;      /* Parte then */
88      execute(savepc + 3);     /*condicion*/
89      d = pop();
90      if(d.val)
```

```
91          execute(*((Inst **)(savepc)));
92      else if(*((Inst **)(savepc + 1)))       /*Parte del else*/
93          execute(*((Inst **)(savepc + 1)));
94      pc = *((Inst **)(savepc + 2)); /*Vamos a la siguiente posicion de la
            pila*/
95  }
96
97  void bltin(){    /*Evaluar un predefinido en el tope de la pila */
98      Datum d;
99      d = pop();
100     d.val = (*(Vector * (*)() )(*pc++))(d.val);
101     push(d);
102 }
```