INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

REPORTE DE PRÁCTICA 06:

Ciclos for y decisiones

Esquivel Valdez Alberto
Profesor: Tecla Parra Roberto
Compiladores
Grupo: 3CM7
Fecha de entrega: 03 de diciembre del 2018

1. Descripción de la práctica

Agregar ciclo for
Empezar por dibujar el mapa de memoria
Tener en cuenta que el codigo se genera en postfijo para dibujar dicho mapa de memoria

2. Código

2.1. vectorcal.y

```
    Inst* inst;
    int eval;
}

/**Creaci n de s mbolos terminales y no terminales**/
%token<comp> NUMBER
%type<comp> escalar

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

%type<inst> exp asgn

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

//Nuevos s mbolos gramaticales para la pr ctica 6
%token<sym>     FOR
%type<inst>     for exprn
/**Jerarqu a de operadores**/

//Para pr ctica 3
%right '='
//Para la pr ctica 5
%left OR AND
%left GT GE LT LE EQ NE
//S mbolos gramaticales de la pr ctica 1
//Suma y resta de vectores
%left '+' '-'
//Escalar por un vector
%left '*'
//Producto cruz y producto punto
%left '#' '.' '|'
//Para la pr ctica 5
```

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

```
80                                                  v -> vec[2] = $4;
81                                                  $$ = install("",
                                                        VECTOR, v);}
82          ;
83
84
85      number: NUMBER                          {$$ = installd("", NUMB,
            $1);}
86      ;
87
88      stmt: exp                                                   {code(
            pop);}
89      |    PRINT exp                                              {code(
            print); $$ = $2;}
90      |    while cond stmt end                                    {($1)[1]
            = (Inst)$3;
91                                                                  ($1)[2]
                                                                        = (
                                                                    Inst
                                                                    )$4
                                                                    ;}
92
93      | if cond stmt end                                          {($1)[1]
            = (Inst)$3;
94                                                                  ($1)[3]
                                                                        = (
                                                                    Inst
                                                                    )$4
                                                                    ;}
95
96      | if cond stmt end ELSE stmt end                            {($1)[1]
            = (Inst)$3;
97                                                                  ($1)[2]
                                                                        = (
                                                                    Inst
                                                                    )$6;
98                                                                  ($1)[3]
                                                                        = (
                                                                    Inst
                                                                    )$7
                                                                    ;}
99
100     | for '(' exprn ';' exprn ';' exprn ')' stmt end            {($1)[1]
            = (Inst)$5;
101                                                                 ($1)[2]
                                                                        = (
                                                                    Inst
                                                                    )$7;
```

```
102                                                          ($1)[3]
                                                                 = (
                                                             Inst
                                                             )$9;
103                                                          ($1)[4]
                                                                 = (
                                                             Inst
                                                             )$10
                                                             ;}
104     | '{' stmtlst '}'                                  {$$ = $2
            ;}
105     ;
106
107     cond: '(' exp ')'                       {code(STOP); $$ = $2;}
108     ;
109
110     while: WHILE                            {$$ = code3(whilecode,
            STOP, STOP);}
111     ;
112
113     if: IF                                  {$$ = code(ifcode);
114                                              code3(STOP, STOP, STOP)
                                                     ;}
115     ;
116
117     end: /* NADA */                         {code(STOP); $$ = progp
            ;}
118     ;
119
120     stmtlst: /* NADA */                     {$$ = progp;}
121     |    stmtlst '\n'
122     |    stmtlst stmt
123     ;
124
125     //PRCTICA 6
126     for: FOR                                {$$ = code(forcode);
            code3(STOP, STOP, STOP); code(STOP);}
127     ;
128
129     exprn: exp                              {$$ = $1; code(STOP);}
130     |    '{' stmtlst '}'                    {$$ = $2;}
131     ;
132
133 %%
134
135 /*
        ************************************************************************
```

```
136 *                                  C digo en C
                                                  *
137 ************************************************************************
    */
138 #include <stdio.h>
139 #include <ctype.h>
140 #include <signal.h>
141 #include <setjmp.h>
142
143 jmp_buf begin;
144 char * progname;
145 int lineno = 1;
146
147 void main(int argc, char * argv[]) {
148     progname = argv[0];
149     init();
150     setjmp(begin);
151     signal(SIGFPE, fpecatch);
152     for(initcode(); yyparse (); initcode())
153                 execute(prog);
154 }
155
156 void execerror(char * s, char * t){
157     warning(s, t);
158     longjmp(begin, 0);
159 }
160
161 void fpecatch(){
162     execerror("Excepcion de punto flotante", (char *)0);
163 }
164
165 int yylex(){
166     int c;
167     while ((c = getchar()) == ' ' || c == '\t')
168         /**SALTA BLANCOS**/;
169
170     if (c == EOF)
171       return 0;
172
173     if (isdigit(c) ) {
174         ungetc(c, stdin);
175         scanf("%df\n", &yylval.comp);
176         return NUMBER;
177     }
178
179     if (isalpha(c)) {
```

2.2. hoc.c

Lookup sirve para encontrar s en la tabla de símbolos:

```
1  #include "hoc.h"
2  #include "y.tab.h"
3  #include <string.h>
4  #include <stdlib.h>
5
6  static Symbol *symlist=0;     /* tabla de simbolos: lista ligada
       */
7
8  Symbol *lookup(char *s)     /* encontrar s en la tabla de
       s mbolos   */
9  {
10 Symbol  *sp;
11         for (sp = symlist; sp != (Symbol *)0; sp = sp->next)
12                 if (strcmp(sp->name, s)== 0)
13                         return sp;
14         return 0;       /* 0 ==> no se encontr  */
15 }
16
17 Symbol *install(char *s,int t, Vector *vec) /* instalar s en la
       tabla de   s mbolos   */
18 {
19         Symbol *sp;
20         char *emalloc();
21         sp = (Symbol *) emalloc(sizeof(Symbol));
22         sp->name = emalloc(strlen(s)+ 1) ; /* +1 para '\0' */
23         strcpy(sp->name, s);
24         sp->type = t;
25         sp->u.vec = vec;
26         sp->next = symlist;    /*  poner al frente de la lista
               */
27         symlist = sp;
28     return sp;
29 }
30
31 char  *emalloc(unsigned n)       /*    revisar el regreso desde
       malloc   */
32 {
33         char *p;
34         p = malloc(n);
35         return p;
36 }
```

2.3. code.c

```
1  #include "hoc.h"
2  #include "y.tab.h"
3  #include <string.h>
4  #include <stdlib.h>
5
6  static Symbol *symlist=0;     /* tabla de simbolos: lista ligada
       */
7
8  Symbol *lookup(char *s)     /* encontrar s en la tabla de
       s mbolos   */
9  {
10 Symbol  *sp;
11         for (sp = symlist; sp != (Symbol *)0; sp = sp->next)
12                 if (strcmp(sp->name, s)== 0)
13                         return sp;
14         return 0;       /* 0 ==> no se encontr  */
15 }
16
17 Symbol *install(char *s,int t, Vector *vec) /* instalar s en la
       tabla de  s mbolos   */
18 {
19         Symbol *sp;
20         char *emalloc();
21         sp = (Symbol *) emalloc(sizeof(Symbol));
22         sp->name = emalloc(strlen(s)+ 1) ; /* +1 para '\0' */
23         strcpy(sp->name, s);
24         sp->type = t;
25         sp->u.vec = vec;
26         sp->next = symlist;    /*  poner al frente de la lista
               */
27         symlist = sp;
28     return sp;
29 }
30
31 char  *emalloc(unsigned n)       /*   revisar el regreso desde
       malloc  */
32 {
33         char *p;
34         p = malloc(n);
35         return p;
36 }
37
38 Symbol * installd(char * s, int t, double d){  //Se instala 's'
39     Symbol * sp;                                  //en la tabla de
           simbolos
40     char * emalloc();
41     sp = (Symbol *)emalloc(sizeof(Symbol));
```

```
42
43      sp->name = emalloc( strlen(s) + 1);    /* '\0' es +1 */
44      strcpy( sp->name, s);
45      sp->type = t;
46      sp->u.comp = d;
47      sp->next = symlist;          /* Se pone al frente de la lista
            */
48      symlist = sp;
49
50      return sp;
51  }
```

2.4. Definición de las variables x,y e imprimir en un ciclo for el valor de x, aumentando x de 1 en 1

```
[alberto@alberto-pc Práctica 06]$ ./a.out
x=[0 0 0]
y=[20 0 0]
for(x; x<[20 0 0]; x=x+[1 0 0]){ print x}
forcode[ 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 ]
```