INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

COMPILADORES

3CM7

PRÁCTICA 06

Alumno:
Díaz Medina Jesús Kaimorts

Profesor:
Tecla Parra Roberto

Ciclo FOR.

29 de noviembre 2018

Introducción.

Todas las estructuras de control tienen un único punto de entrada. Las estructuras de control se pueden clasificar en: secuenciales, iterativas y de control avanzadas. Esta es una de las cosas que permiten que la programación se rija por los principios de la programación estructurada. Los lenguajes de programación modernos tienen estructuras de control similares. Básicamente lo que varía entre las estructuras de control de los diferentes lenguajes es su sintaxis; cada lenguaje tiene una sintaxis propia para expresar la estructura.

Las decisiones son estructuras de control que realizan una pregunta la cual retorna verdadero o falso (evalúa una condición) y selecciona la siguiente instrucción a ejecutar dependiendo la respuesta o resultado, por otra parte, un ciclo es una sentencia que se realiza repetidas veces un bloque aislado de código, hasta que la condición asignada a dicho bucle deje de cumplirse.

Desarrollo.

En esta práctica se utilizó el HOC5 que ya incorpora secciones de condiciones y ciclos dentro de su gramática permitiéndonos generar ciclos con >, <, >=, <=, ==, != los cuales podremos generar ciclos y condiciones de la manera que más nos convenga

```
%token <num> NUMBER
%token <sym> VAR INDEF VECT NUMB PRINT WHILE IF ELSE BLTIN FOR
%type <inst>  stmt asgn expr stmtlist cond while if begin end for exprn
%type <sym> vector numero
%type <num> escalar

%right '='
%left   OR
%left   AND
%left   GT GE LT LE EQ NE
%left '+' '-'
%left '*' '#' '|' ':'
%left   UNARYMINUS NOT
%% /* A continuaci□as reglas gramaticales y las acciones */
list:
    | list '\n'
    | list asgn '\n'  { code2(pop1, STOP); return 1; }
    | list stmt '\n'    { code(STOP); return 1; }
    | list expr '\n'   { code2(print, STOP); return 1;}
    | list escalar '\n'  { code2(printd, STOP); return 1;}
    | list error '\n' { yyerrok; }
    ;
asgn:   VAR '=' expr { $$=$3; code3(varpush, (Inst)$1, assign);}
    ;
stmt:     expr   { code(pop1); }
```

```
    | PRINT expr    { code(print); $$ = $2;}
    | while cond stmt end {
        ($1)[1] = (Inst)$3;     /* cuerpo de la iteraci□/
        ($1)[2] = (Inst)$4; }   /* terminar si la condici□o se cumple */
    | if cond stmt end {   /* proposici□f que no emplea else */
        ($1)[1] = (Inst)$3;     /* parte then */
        ($1)[3] = (Inst)$4; }   /* terminar si la condici□o se cumple */
    | if cond stmt end ELSE stmt end {  /* proposici□f con parte else */
        ($1)[1]   =   (Inst)$3; /*  parte then  */
        ($1)[2]   =   (Inst)$6; /* parte else   */
        ($1)[3]   =   (Inst)$7;   } /*   terminar si la condici□o se cumple
*/
    |for '(' exprn ';' exprn ';' exprn ')' stmt end {
        ($1)[1] = (Inst)$5;
        ($1)[2] = (Inst)$7;
        ($1)[3] = (Inst)$9;
        ($1)[4] = (Inst)$10; }
    |   '{'   stmtlist   '}' {   $$  =  $2;   }
    ;
    exprn:  expr   {  $$=$1; code(STOP); }
    |   '{'   stmtlist   '}' {   $$  =  $2;   }
    ;
    cond:   '('   expr   ')'      {   code(STOP);  $$=  $2;   }
    ;
while:  WHILE   {$$ = code3(whilecode,STOP,STOP); }
    ;
if:     IF      { $$=code(ifcode); code3(STOP, STOP, STOP); }
    ;
for: FOR {$$=code(forcode);code3(STOP,STOP,STOP);code(STOP);}
;
begin:      { $$ = progp; }
;
end:      /* nada */{ code(STOP); $$ = progp; }
    ;
stmtlist: /* nada */    { $$ = progp; }
    | stmtlist '\n'
    | stmtlist stmt
    ;
expr:   vector {/*puts("xprvector"); imprimeVector($1->u.val);*/
                $$ = code2(constpush, (Inst)$1);}
    | VAR  { $$ = code3(varpush, (Inst)$1, eval);}
    | asgn
    |   BLTIN '(' expr ')' { $$ = $3; code2(bltin,(Inst)$1->u.ptr); }
    | expr '+' expr     { code(add); }
    | expr '-' expr     { code(sub); }
    | escalar '*' expr  { code(escalar);}
    | expr '#' expr     { code(productoc); }
    | expr GT expr      { code(gt); }
    | expr GE expr      { code(ge); }
    | expr LT expr      { code(lt); }
    | expr LE expr      { code(le); }
    | expr EQ expr      { code (eq); }
    | expr NE expr      { code(ne); }
    | expr AND expr     { code(and); }
    | expr OR expr      { code(or); }
    | NOT expr          { $$ = $2; code(not); }
```

```
    ;
escalar: numero { /*printf("numero: %lf",$1->u.num);*/ code2(constpushd,
(Inst)$1);}
        | expr ':' expr         { code(productop); }
        | '|' expr '|'          { code(magnitud); }
        ;
vector: '[' NUMBER NUMBER NUMBER ']' {Vector *vector1= creaVector(3);
vector1->vec[0] = $2; vector1->vec[1] = $3; vector1->vec[2] = $4; $$ =
install("", VECT , vector1);}
        ;

numero: NUMBER { $$ = installd("", NUMB,$1); }
        ;
%%

#include "hoc.h"
#include "y.tab.h"
#include <math.h>
static struct{
    char * name;    /* Keywords */
    int kval;
}keywords[] = {
    "if", IF,
    "else", ELSE,
    "while", WHILE,
    "print", PRINT,
    0,0,
};
int init(){ /* Se instalan las constantes y predefinidos en la tabla */
    int i;
    Symbol * s;
    for (i = 0; keywords[i].name; i++)
        install(keywords[i].name, keywords[i].kval, NULL);
}
/* Ciclo WHILE */
void whilecode(){
    Datum d;
    Inst * savepc = pc;     /* Cuerpo de la iteración */
    execute(savepc + 2);    /* Condición */
    d = pop();

    while(d.val){
        execute(* ( (Inst **)(savepc) ));   /* Cuerpo del ciclo*/
        execute(savepc + 2);
        d = pop();
    }
    pc = *((Inst **)(savepc + 1));  /*Vamos a la siguiente posicion*/
}
/* Condición IF */
void ifcode(){
    Datum d;
    Inst * savepc = pc;     /* Parte then */
    execute(savepc + 3);    /*condicion*/
    d = pop();
    if(d.val)
        execute(*((Inst **)(savepc)));
```

```
    else if(*((Inst **)(savepc + 1)));      /*Parte del else*/
        execute(*((Inst **)(savepc + 1)));

    pc = *((Inst **)(savepc + 2)); /*Vamos a la siguiente posicion de la
pila*/
}
void  forcode() {
        Datum d;
        Inst  *savepc  = pc;    /*  dir cond  */
        execute(savepc+4);  /*  expr1  */
        execute(*((Inst  **)(savepc)));     /*   condición   */
        d  =  pop();
        while   (d.eval)   {
                execute(*((Inst  **)(savepc+2)));     /*  cuerpo  */
                execute(*((Inst  **)(savepc+1)));
                pop();
                execute(*((Inst  **)(savepc)));     /*   condición   */
                d  = pop();
        }
        pc  =  *((Inst  **)(savepc+3));     /*   siguiente proposición   */
}
switch(c){
        case '>':   return follow('=', GE, GT);
        case '<':   return follow('=', LE, LT);
        case '=':   return follow('=', EQ, '=');
        case '!':   return follow('=', NE, NOT);
        case '|':   return follow('|', OR, '|');
        case '&':   return follow('&', AND, '&');
        case '\n':  lineno++; return '\n';
        default:    return c;
    }
    if( c == '\n')  lineno++;
    return c;
```

```
x=[1 0 0]
y=[0 1 0]

if(x==y){x=x+y print x}

LibreOffice Writer

for(z; z<[9 9 9]; z=z+y) {print z}
[1.000000, 1.000000, 0.000000]
[1.000000, 1.000000, 0.000000]
[1.000000, 2.000000, 0.000000]
[1.000000, 3.000000, 0.000000]
[1.000000, 4.000000, 0.000000]
[1.000000, 5.000000, 0.000000]
[1.000000, 6.000000, 0.000000]
[1.000000, 7.000000, 0.000000]
[1.000000, 8.000000, 0.000000]
[1.000000, 9.000000, 0.000000]
[1.000000, 10.000000, 0.000000]
[1.000000, 11.000000, 0.000000]
[1.000000, 12.000000, 0.000000]
[1.000000, 13.000000, 0.000000]
[1.000000, 14.000000, 0.000000]
[1.000000, 15.000000, 0.000000]
```

Conclusión.

Un ciclo nos permite controlar diversos procesos o repetirlos de una manera más eficiente mientras que una condicional nos permite manejar y controlar los procesos en ejecución para obtener el resultado deseado.