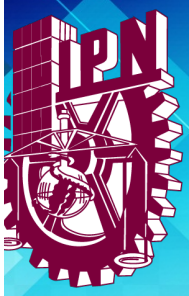

INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

PRÁCTICA 5

Calculadora científica con condiciones y bucle.

Profesor:

- Tecla Parra Roberto

Alumno:

- Pacheco Delgado José Jacobeth

Grupo:

- 3CM8

Materia:

- Compiladores

Introducción

Esta práctica consiste en agregar condiciones y bucles a la practica que hemos estado construyendo a lo largo del semestre, por medio de la máquina de pila anteriormente construida y la tabla de símbolos.

Objetivos

- Agregar condiciones y bucles a la calculadora científica de racionales.

Desarrollo

Realizamos modificaciones a nuestra gramática:

Racional_calc.y

```
%right  '='
%left   OR
%left   AND
%left   GT GE LT LE EQ NE
%left '+' '-'
%left '*' '/'
%left   UNARYMINUS NOT
%right ';'
%%
list:
      | list '\n'
      | list asigna '\n'  { code2(pop, STOP); return 1; }
      | list exp '\n'   { code2(print, STOP); return 1;}
      | list stmt '\n' { code(STOP); return 1; }
      | list error '\n' {yyerrok;}
      ;
asigna : var '=' exp  {$$=$3; code3(varpush, (Inst)$1, assign); }
       ;
stmt:     exp  { code(pop); }
       | PRINT exp    { code(prexpr); $$ = $2;}
       | while cond stmt end {
```

```
                ($1)[1] = (Inst)$3;     /* cuerpo de la iteración */
                ($1)[2] = (Inst)$4; }   /* terminar si la condición no se
cumple */
        | if cond stmt end {    /* proposición if que no emplea else */
                ($1)[1] = (Inst)$3;     /* parte then */
                ($1)[3] = (Inst)$4; }   /* terminar si la condición no se
cumple */
        | if cond stmt end ELSE stmt end {  /* proposición if con parte
else */
                ($1)[1]   =   (Inst)$3;       /*  parte then  */
                ($1)[2]   =   (Inst)$6;       /* parte else   */
                ($1)[3]   =   (Inst)$7;   } /*   terminar si la condición
no s cumple  */
        | for '(' asigna ';' cond';'exp')' stmt end {
                ($1)[1] = (Inst)$3;
                ($1)[2] = (Inst)$7;
                ($1)[3] = (Inst)$9;
                ($1)[4] = (Inst)$10;
        }
        |   '{'   stmtlist   '}' {   $$   =   $2;   }
        ;
cond:   '('   exp   ')'       {   code(STOP);  $$=$2;   }
;
while:  WHILE   {   $$   =   code3(whilecode,STOP,STOP); }
        ;
if:IF   { $$=code(ifcode); code3(STOP, STOP, STOP);  }
        ;
for:FOR {

        $$=code(forcode);
        code3(STOP,STOP,STOP);
        code(STOP);
        }
;
end:      /* nada */{ code(STOP); $$ = progp; }
        ;
stmtlist: /* nada */   { $$ = progp; }
        | stmtlist '\n'
        | stmtlist stmt
        ;
exp:    racionalnum { code2(constpush, (Inst)$1);}
        | asigna
        | bltin  '(' exp ')' { $$=$3;
                code2(BLTIN, (Inst)$1->u.ptr);}
        | var { code3(varpush, (Inst)$1, eval);}
```

```
        | exp '+' exp {code(add);}
        | exp '-' exp {code(sub);}
        | exp '*' exp {code(mul);}
        | exp '/' exp {code(divi);}
        | '(' exp ')' { $$ = $2; }
        |exp GT exp  { code(gt); }
        |exp GE exp  { code(ge); }
        |exp LT exp  { code(lt); }
        |exp LE exp  { code(le); }
        |exp EQ exp  { code (eq); }
        |exp NE exp  { code(ne); }
        |exp AND exp { code(and); }
        |exp OR exp  { code(or); }
        |NOT exp      { $$ = $2; code(not); }
        ;
  ;
%%
```

Así como a nuestro archivo racional_calc.l

Donde añadimos el **mayor que** > , mejor llamado **GT.**

```
fr [+]
GT[>]
GE[>][=]
LT[<]
LE[<][=]
EQ[=][=]
NE[!][=]
AND[&][&]
OR[|][|]
NOT[!]

racionalnum {ws}*[-]*{ws}*{number}{ws}*{fr}{ws}*{number}{ws}*

%%
{var} {
        Symbol *s;
        lookup(yytext);
        if ((s=lookup(yytext)) == 0 )
                s= install(yytext,undef,0.0);
        yylval.sim = s;
        return s->tipo == undef ? var: s -> tipo;
}
{GT} {
```

```
        return GT;
}
{GE} {

        return GE;
}
{LT} {

        return LT;
}
{LE} {

        return LE;
}
{EQ} {

        return EQ;
}
{NE} {
        return NE;
}
{AND} {
        return AND;
}
{OR} {

        return GE;
}
{NOT} {
        return NOT;
}
```

Por último modificamos nuestro archivo code.c donde vamos a operar nuestra pila, donde manejamos la memoria a modo de diseñar un **bucle**, y un **IF e IF-ELSE,** también comparamos cuando se mandan a llamar los operadores: **>, <, >=, <=,** not, and, or

```
whilecode() {
Datum d;
Inst  *savepc  = pc;   /*  cuerpo de la iteraci */
execute(savepc+2);     /*   condici  */
d  =  pop();
while   (d.val->dec)   {
execute(*((Inst  **)(savepc)));     /*  cuerpo  */
execute(savepc+2);
d  = pop();
}
pc  =  *((Inst  **)(savepc+1));     /*   siguiente proposici  */
}
ifcode()
{
Datum d;
Inst  *savepc  = pc;   /* parte then */
execute(savepc+3);     /*  condici  */
d  =  pop();
if(d.val->dec)
        execute(*((Inst **)(savepc)));
else if (*((Inst **)(savepc+1)))
execute(*(( Inst  **) (savepc+1)));
pc = *((Inst **)(savepc+2));
}
gt() {
Datum d1,  d2,d3;
d2 = pop();
d1 = pop();
d3.val = creaRacional(0, 1, 0.0, 0);
d3.val->dec =   (double)(d1.val->dec  > d2.val->dec);
push(d3);
}

lt()
{
Datum d1,  d2;
d2 = pop();
d1 = pop();
d1.val->dec  =   (double)(d1.val->dec  < d2.val->dec);
push(d1);
```

```
}

ge( ) {
Datum d1,  d2;
d2  = pop();
d1  = pop();
d1.val->dec   =   (double)(d1.val->dec  >= d2.val->dec);
push(d1) ;
}

le() {
Datum d1,  d2;
d2   =  pop();
d1   =  pop();
d1.val->dec  =   (double)(d1.val->dec   <=  d2.val->dec);
push(d1);
}

eq( ) {
Datum d1,  d2;
d2  = pop();
d1  = pop();
d1.val->dec  =   (double) (d1.val->dec  ==  d2.val->dec);
push(d1);
}

ne(){
Datum d1, d2;
d2 = pop();
d1 = pop();
d1.val->dec = (double)(d1.val->dec != d2.val->dec);
push(d1);
}

and()
{
Datum d1,   d2;
d2   = pop();
d1   = pop();
d1.val->dec = (double)(d1.val->dec   !=   0.0 && d2.val->dec   !=  0.0);
push(d1);
}
```

Conclusión

Es interesante aprovechar la máquina de pila para comenzar a crear mapas de memoria para distintas estructuras de control que nos permite diseñar esta. Cada vez la calculadora científica se torna más compleja conteniendo más y más funciones.