NSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE COMPUTO

PRACTICA 5 Decisiones y Ciclos

Opción calculadora vectores

COMPILADORES

Grupo: 3CM17

ALUMNO: MORA GUZMÁN JOSE ANTONIO

FECHA ENTREGA: Miércoles 3 noviembre 2021

Descripción

Para la presente practica se deben agregar sentencias IF, ELSE, EHILE y tambien operadores logicos como lo son:

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

Para que todo esto se pueda aplicar con los vectores se deben calcular las magnitudes de los mismos

Código

Se modifico la gramatica, y se añadieron mas simbolos gramaticales y se añadio un elemento a la definicion de tipos de dato de YACC

```
%}
%union{
    double comp;
    Vector* vec;
    Symbol* sym;
    Inst* inst;
    //P5
    int eval;
}

%token<comp> NUMBER
%type<comp> escalar

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

%type<inst> exp asgn

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


%right '='
//P5
%left OR AND
%left GT GE LT LE EQ NE
%left '+' '-'
%left '*'
%left '#' '.' '|'
//Para la práctica 5
%left UNARYMINUS NOT

```

```
/**Gramática**/
%%

    list:
      | list '\n'
      | list asgn '\n'    {code2(pop, STOP); return 1;}
      | list stmt '\n'    {code(STOP); return 1;}   //Añadida en la práctica 5
      | list exp '\n'     {code2(print, STOP); return 1;}
      | list escalar '\n' {code2(printd, STOP); return 1;}
      | list error '\n'   {yyerror;}
      ;

    asgn: VAR '=' exp   {$$ = $3; code3(varpush, (Inst)$1, assign);}
    ;

    exp: vector             {code2(constpush, (Inst)$1);}
      | VAR                 {code3(varpush, (Inst)$1, eval);}
      | asgn
      | BLTIN '(' exp ')'   {$$ = $3; code2(bltin, (Inst)$1 -> u.ptr);}
      | exp '+' exp         {code(add);}
      | exp '-' exp         {code(sub);}
      | escalar '*' exp     {code(escalar);}
      | exp '*' escalar     {code(escalar);}
      | exp '#' exp         {code(producto_cruz);}
      //Para la práctica 5
      | exp GT exp          {code(mayor);}
      | exp LT exp          {code(menor);}
      | exp GE exp          {code(mayorIgual);}
      | exp LE exp          {code(menorIgual);}
      | exp EQ exp          {code(igual);}
      | exp NE exp          {code(diferente);}
      | exp OR exp          {code(or);}
      | exp AND exp         {code(and);}
      | NOT exp             {$$ = $2; code(not);}
    ;
    escalar: number         {code2(constpushd, (Inst)$1);}
      | exp '.' exp         {code(producto_punto);}
      | '|' exp '|'         {code(magnitud);}
      ;

    vector: '[' NUMBER NUMBER NUMBER ']'   {  Vector* v = creaVector(3);
                                              v -> vec[0] = $2;
                                              v -> vec[1] = $3;
                                              v -> vec[2] = $4;
                                              $$ = install("", VECTOR, v);}
      ;

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

    //P5
    stmt: exp                         { code(pop); }
    |   PRINT exp                     {code(print); $$ = $2;}
    |   while cond stmt end           { ($1)[1] = (Inst)$3;
                                        ($1)[2] = (Inst)$4;}
    | if cond stmt end                { ($1)[1] = (Inst)$3;
                                        ($1)[3] = (Inst)$4;}
    | if cond stmt end ELSE stmt end  {($1)[1] = (Inst)$3;
                                      ($1)[2] = (Inst)$6;
                                      ($1)[3] = (Inst)$7;}
    | '{' stmtlst '}'                 {$$ = $2;}
    ;

    cond: '(' exp ')'                 {code(STOP); $$ = $2;}
    ;

    while: WHILE                      {$$ = code3(whilecode, STOP, STOP);}
    ;

    if: IF                            {$$ = code(ifcode);
                                       code3(STOP, STOP, STOP);}
    ;

      end: /* NADA */                       {code(STOP); $$ = progp;}
      ;

      stmtlst: /* NADA */                   {$$ = progp;}
      |   stmtlst '\n'
      |   stmtlst stmt
      ;

%%
```

Para evaluar operadores logicos se añadio codigo al yylex

```
204
205     //Añadido
206     switch(c){
207         case '>': return follow('=', GE, GT);
208         case '<': return follow('=', LE, LT);
209         case '=': return follow('=', EQ, '=');
210         case '!': return follow('=', NE, NOT);
211         case '|': return follow('|', OR, '|');
212         case '&': return follow('&', AND, '&');
213         case '\n': lineno++; return '\n';
214         default: return c;
215     }
216 }
```

y se añadio otra funcion

```
218 int follow(int expect, int ifyes, int ifno){ //Buscar operadores
219     int c  = getchar();
220     if  (c  ==  expect)
221         return ifyes;
222     ungetc(c,   stdin);
223     return  ifno;
224 }
```

Y finalmente en code.c se agregaron funciones

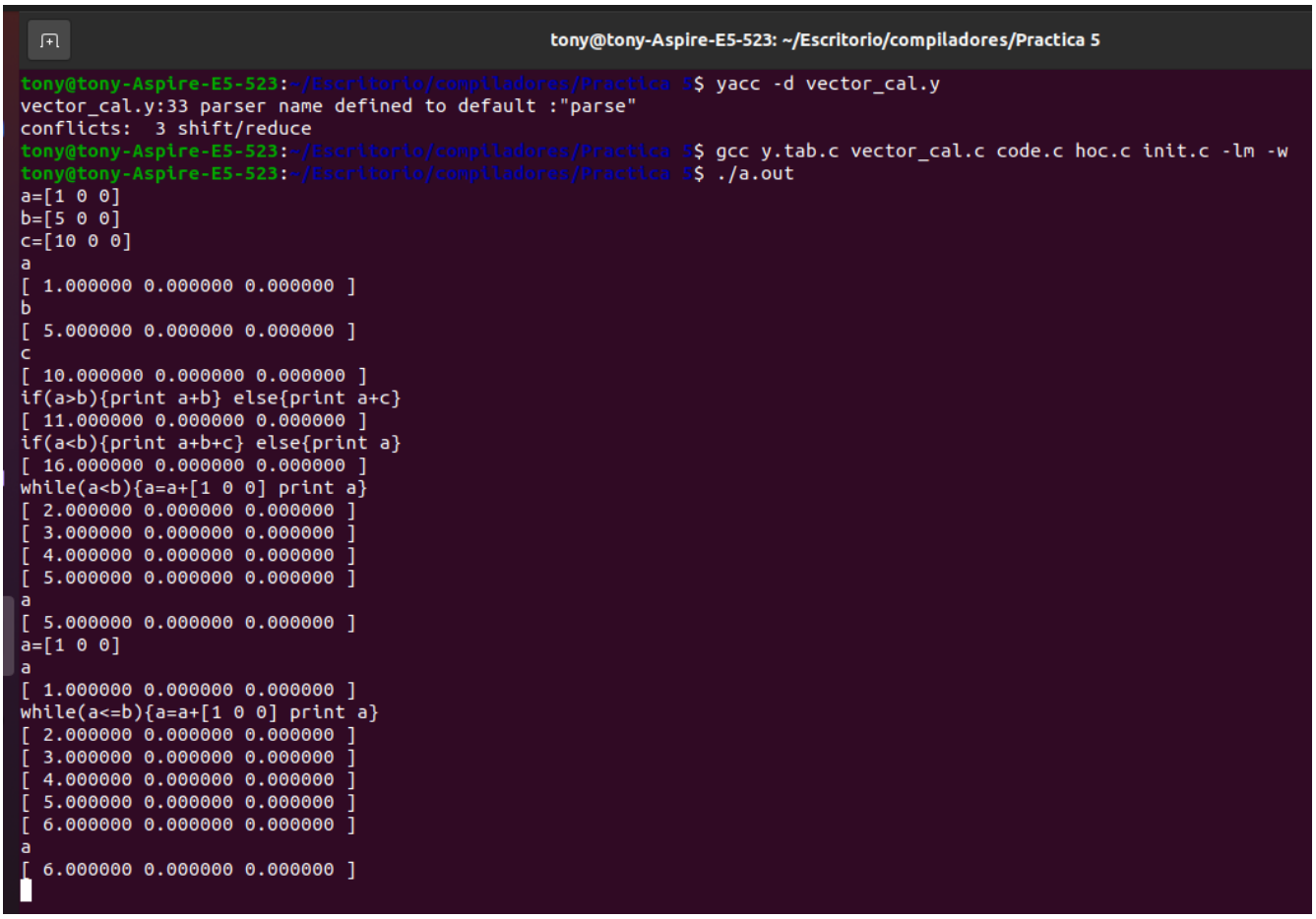
```
163 /******** Funciones de comparacion ********/
164 void mayor(){
165     Datum d1, d2;
166     d2 = pop();
167     d1 = pop();
168     d1.num = (int)( vectorMagnitud(d1.val) > vectorMagnitud(d2.val) );
169     push(d1);
170 }
171
172 void menor(){
173     Datum d1, d2;
174     d2 = pop();
175     d1 = pop();
176     d1.num = (int)( vectorMagnitud(d1.val) < vectorMagnitud(d2.val) );
177     push(d1);
178 }
179
180 void mayorIgual(){
181     Datum d1, d2;
182     d2 = pop();
183     d1 = pop();
184     d1.num = (int)( vectorMagnitud(d1.val) >= vectorMagnitud(d2.val) );
185     push(d1);
186 }
187
188 void menorIgual(){
189     Datum d1, d2;
190     d2 = pop();
191     d1 = pop();
192     d1.num = (int)( vectorMagnitud(d1.val) <= vectorMagnitud(d2.val) );
193     push(d1);
194 }
```

```
196 void igual(){
197     Datum d1, d2;
198     d2 = pop();
199     d1 = pop();
200     d1.num = (int)( vectorMagnitud(d1.val) == vectorMagnitud(d2.val) );
201     push(d1);
202 }
203
204 void diferente(){
205     Datum d1, d2;
206     d2 = pop();
207     d1 = pop();
208     d1.num = (int)( vectorMagnitud(d1.val) != vectorMagnitud(d2.val) );
209     push(d1);
210 }
211
212 void and(){
213     Datum d1, d2;
214     d2 = pop();
215     d1 = pop();
216     d1.num =(double)(d1.num != 0.0 && d2.num != 0.0);
217     push(d1);
218 }
219
220 void or(){
221     Datum d1, d2;
222     d2 = pop();
223     d1 = pop();
224     d1.num = (double)(d1.num != 0.0 || d2.num != 0.0);
225     push(d1);
226 }
227
```

```
228 void not(){
229     Datum d1;
230     d1 = pop();
231     d1.val = (double)(d1.val != 0.0);
232     push(d1);
233 }
234 /********* Ciclos *********/
235 void whilecode(){
236     Datum d;
237     Inst* savepc = pc;
238     execute(savepc + 2);
239     d = pop();
240     while(d.val){
241         execute(* ( (Inst **)(savepc) ));
242         execute(savepc + 2);
243         d = pop();
244     }
245     pc = *((Inst **)(savepc + 1));
246 }
247
248 void ifcode(){
249     Datum d;
250     Inst* savepc = pc;
251     execute(savepc + 3);
252     d = pop();
253     if(d.val)
254         execute(*((Inst **)(savepc)));
255     else if(*((Inst **)(savepc + 1)))
256         execute(*((Inst **)(savepc + 1)));
257     pc = *((Inst **)(savepc + 2));
258 }
259
```

```
260 void bltin(){
261     Datum d;
262     d = pop();
263     d.val = (*(Vector * (*)() )(*pc++))(d.val);
264     push(d);
265 }
```

Pruebas del programa



Conclusiones

Esta practica se me hizo muy interesante debido a que es donde mas se puede identificar el uso del compilador debido a que ya tiene sentencias if y ciclos while, se me hizo un tanto sencilla puesto que al ir trabajando las practicas anteriores en esta solo es añadir unas cuantas lineas de codigo

Link del Video

https://youtu.be/16Jjw_ag8hs