



INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

Práctica número 5:
Calculadora para vectores
(Con ciclo while y operadores lógicos)

5 de enero de 2021

Grupo: 3CM7

Nombre del alumno:
Ramos Mesas Edgar Alain

Número de boleta:
2013090243

MATERIA: COMPILADORES

1. Introducción

Las sentencias de decisión o condición, 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. En nuestros algoritmos, muchas veces tenemos que tomar una decisión en cuanto a que se debe ejecutar basándonos en una condición. Los ciclos while son también una estructura cíclica, que nos permite ejecutar una o varias líneas de código de manera repetitiva sin necesidad de tener un valor inicial e incluso a veces sin siquiera conocer cuándo se va a dar el valor final que esperamos, los ciclos while, no dependen directamente de valores numéricos, sino de valores booleanos, es decir su ejecución depende del valor de una condición dada.

2. Desarrollo

Esta práctica consiste en agregar a la calculadora para vectores, los ciclos y sentencias de decisión, while e if, respectivamente.

Para el desarrollo de esta práctica se tomó en cuenta el mapa de memoria que se genera en las sentencias if y while, pues es esencial para poder realizar correctamente la adición de estas sentencias a la calculadora que se ha ido desarrollando a lo largo de las prácticas previas. Además, dado que se trabajará con ciclos y condicionales, se requiere de la implementación de operadores lógicos como:

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

En la gramática se definen los nuevos simbolos gramaticales necesarios para realizar el ciclo while o las condiciones if o if-else.

```
36 //NUEVOS SÍMBOLOS GRAMATICALES PARA LA PRÁCTICA 5
37 %token<sym>      PRINT WHILE IF ELSE BLTIN
38 %type<inst>      stmt stmtlst cond while if end
```

Posteriormente colocamos la precedencia de operadores que nos permitirá hacer las comparaciones y las operaciones binarias, resaltando el unaryminus que nos da prioridad más alta.

```

44 //Para la práctica 5
45 %left OR AND
46 %left GT GE LT LE EQ NE

```

En la declaración while se especifica la condición de la iteración para saber hasta donde termina y continua la siguiente instrucción. En el caso de la sentencia if el mapa de memoria se comporta de manera similar guardando el cuerpo de la condición, el final y en el caso del if-else, el conjunto de instrucciones si la condición no se cumple. Finalmente, con end se indica el final de la iteración o condición indicando el siguiente espacio para la instrucción.

```

81 //Para la práctica 5
82 | exp GT exp      {code(mayor);}
83 | exp LT exp      {code(menor);}
84 | exp GE exp      {code(mayorIgual);}
85 | exp LE exp      {code(menorIgual);}
86 | exp EQ exp      {code(igual);}
87 | exp NE exp      {code(diferente);}
88 | exp OR exp      {code(or);}
89 | exp AND exp     {code(and);}
90 | NOT exp         {$$ = $2; code(not);}
91 ;
92
93 escalar: number    {code2(constpushd, (Inst)$1);}
94 | exp '.' exp      {code(producto_punto);}
95 | '|' exp '|'      {code(magnitud);}
96 ;
97
98 vector: '[' NUMBER NUMBER NUMBER ']' { Vector* v = creaVector(3);
99                                     v -> vec[0] = $2;
100                                    v -> vec[1] = $3;
101                                    v -> vec[2] = $4;
102                                    $$ = install("", VECTOR, v);}
103 ;
104
105 //Para la práctica 4
106 number: NUMBER     {$$ = installd("", NUMB, $1);}
107 ;
108
109 //Para la práctica 5
110 stmt: exp           { code(pop); }
111 | PRINT exp         {code(print); $$ = $2;}
112 | while cond stmt end { ($1)[1] = (Inst)$3;
113                       ($1)[2] = (Inst)$4;}
114 | if cond stmt end   { ($1)[1] = (Inst)$3;
115                       ($1)[3] = (Inst)$4;}
116 | if cond stmt end ELSE stmt end {($1)[1] = (Inst)$3;
117                                   ($1)[2] = (Inst)$6;
118                                   ($1)[3] = (Inst)$7;}

```

```

119         | '{' stmtlst '}'          { $$ = $2; }
120         ;
121
122     cond: '(' exp ')'              { code(STOP); $$ = $2; }
123     ;
124
125     while: WHILE                    { $$ = code3(whilecode, STOP, STOP); }
126     ;
127
128     if: IF                          { $$ = code(ifcode);
129                                     code3(STOP, STOP, STOP); }
130     ;
131
132     end: /* NADA */                 { code(STOP); $$ = prog; }
133     ;
134
135     stmtlst: /* NADA */              { $$ = prog; }
136     |      stmtlst '\n'
137     |      stmtlst stmt
138     ;
139
140 %%

```

Dado que evaluaremos operadores lógicos a yylex se añadió el siguiente código:

```

206     //Añadido para la práctica 5
207     switch(c){
208         case '>': return follow('=', GE, GT);
209         case '<': return follow('=', LE, LT);
210         case '=': return follow('=', EQ, '=');
211         case '!': return follow('=', NE, NOT);
212         case '|': return follow('|', OR, '|');
213         case '&': return follow('&', AND, '&');
214         case '\n': lineno++; return '\n';
215         default: return c;
216     }

```

Finalmente se añadió también una función auxiliar que se encarga de buscar operadores.

```

219 int follow(int expect, int ifyes, int ifno){ /* buscar operadores. */
220     int c = getchar();
221     if (c == expect)
222         return ifyes;
223     ungetc(c, stdin);
224     return ifno;
225 }

```

Posteriormente se modificó el archivo code.c debido a que en el se especifica el código para la ejecución del while e if así como el código de las condicionales.

En whilecode primero se salva la posición donde empieza el while y después se ejecuta la condición indicando la posición de dicha condición en este caso, savepc+2. Posteriormente se saca el resultado de la pila y se empieza la iteración si es verdadera, ejecutando el cuerpo de la iteración que se encuentra en el primer STOP al que este momento apunta savepc, terminando la ejecución del cuerpo se vuelve a ejecutar la condición y se obtiene el resultado con pop de la pila y se somete al while de nuevo si es cierta, en caso contrario se ejecuta la instrucción que se encuentra al termino del while y se guardó en el segundo STOP. Igualmente en ifcode se guarda la posición del if y se ejecuta la condición, se obtiene el resultado con pop de la pila y si es verdadera entra al if ejecutando el cuerpo del if guardada en el primer STOP, si no es cierta la condición se ejecuta la parte else que esta guarda en el segundo STOP.

```
156  /***** PRÁCTICA 5 *****/
157  /***** Condicionales *****/
158  void mayor(){
159      Datum d1, d2;
160      d2 = pop();
161      d1 = pop();
162      d1.num = (int)( vectorMagnitud(d1.val) > vectorMagnitud(d2.val) );
163      push(d1);
164  }
165
166  void menor(){
167      Datum d1, d2;
168      d2 = pop();
169      d1 = pop();
170      d1.num = (int)( vectorMagnitud(d1.val) < vectorMagnitud(d2.val) );
171      push(d1);
172  }
173
174  void mayorIgual(){
175      Datum d1, d2;
176      d2 = pop();
177      d1 = pop();
178      d1.num = (int)( vectorMagnitud(d1.val) >= vectorMagnitud(d2.val) );
179      push(d1);
180  }
181
182  void menorIgual(){
183      Datum d1, d2;
184      d2 = pop();
185      d1 = pop();
186      d1.num = (int)( vectorMagnitud(d1.val) <= vectorMagnitud(d2.val) );
187      push(d1);
188  }
189
190  void igual(){
191      Datum d1, d2;
192      d2 = pop();
```

```
193     d1 = pop();
194     d1.num = (int)( vectorMagnitud(d1.val) == vectorMagnitud(d2.val) );
195     push(d1);
196 }
197
198 void diferente(){
199     Datum d1, d2;
200     d2 = pop();
201     d1 = pop();
202     d1.num = (int)( vectorMagnitud(d1.val) != vectorMagnitud(d2.val) );
203     push(d1);
204 }
205
206 void and(){
207     Datum d1, d2;
208     d2 = pop();
209     d1 = pop();
210     d1.num = (int)( vectorMagnitud(d1.val) && vectorMagnitud(d2.val) );
211     push(d1);
212 }
213
214 void or(){
215     Datum d1, d2;
216     d2 = pop();
217     d1 = pop();
218     d1.num = (int)( vectorMagnitud(d1.val) || vectorMagnitud(d2.val) );
219     push(d1);
220 }
221
222 void not(){
223     Datum d1;
224     d1 = pop();
225     d1.num = (int)( vectorMagnitud(d1.val) == (double)0.0 );
226     push(d1);
227 }
228 ***** Ciclos *****
229 void whilecode(){
230     Datum d;
231     Inst* savepc = pc;    /* Cuerpo de la iteración */
232     execute(savepc + 2);  /* Condición */
233     d = pop();
234     while(d.val){
235         execute(* ( (Inst **)(savepc) )); /* Cuerpo del ciclo*/
236         execute(savepc + 2);
237         d = pop();
238     }
239     pc = *((Inst **)(savepc + 1)); /*Vamos a la siguiente posicion*/
240 }
241
```

```

242 void ifcode(){
243     Datum d;
244     Inst* savepc = pc;    /* Parte then */
245     execute(savepc + 3);  /*condicion*/
246     d = pop();
247     if(d.val)
248         execute(*((Inst **)(savepc)));
249     else if(*((Inst **)(savepc + 1)))    /*Parte del else*/
250         execute(*((Inst **)(savepc + 1)));
251     pc = *((Inst **)(savepc + 2)); /*Vamos a la siguiente posicion de la pila*/
252 }
253
254 void bltin(){    /*Evaluar un predefinido en el tope de la pila */
255     Datum d;
256     d = pop();
257     d.val = (*(Vector * (*)() )(*pc++))(d.val);
258     push(d);
259 }

```

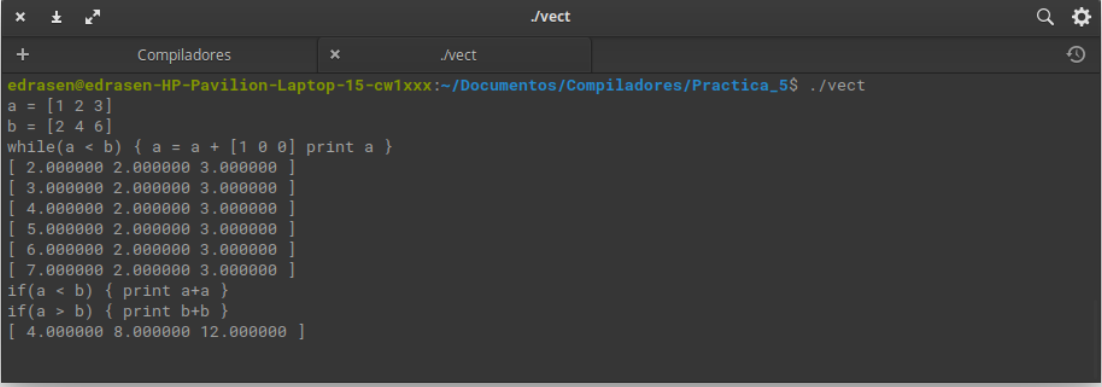
Es importante tomar en cuenta el uso de funciones externas en hoc.h, y la adición de una nueva estructura de palabras “keywords”, en init.c, que serán las que nos permitirán reconocer si hacer un while o un if (o if-else) a partir de la sintaxis.

```

1  #include "hoc.h"
2  #include "y.tab.h"
3  #include <math.h>
4
5  static struct {
6      char    *name;    /* Palabras clave */
7      int     kval;
8  } keywords[] = {
9      "if",      IF,
10     "else" ,    ELSE,
11     "while",    WHILE,
12     "print",    PRINT,
13     0,          0,
14 };
15
16 int init(){ /* Se instalan las constantes y predefinidos en la tabla */
17     int i;
18     Symbol * s;
19     for (i = 0; keywords[i].name; i++)
20         install(keywords[i].name, keywords[i].kval, NULL);
21 }

```

Finalmente, tras realizar todas las modificaciones mencionadas se compiló todo desde un archivo Makefile y se ejecutó el programa obteniendo los siguientes resultados:



```
edrasen@edrasen-HP-Pavilion-Laptop-15-cw1xxx:~/Documentos/Compiladores/Practica_5$ ./vect
a = [1 2 3]
b = [2 4 6]
while(a < b) { a = a + [1 0 0] print a }
[ 2.000000 2.000000 3.000000 ]
[ 3.000000 2.000000 3.000000 ]
[ 4.000000 2.000000 3.000000 ]
[ 5.000000 2.000000 3.000000 ]
[ 6.000000 2.000000 3.000000 ]
[ 7.000000 2.000000 3.000000 ]
if(a < b) { print a+a }
if(a > b) { print b+b }
[ 4.000000 8.000000 12.000000 ]
```

Como se puede ver, primero declaramos dos vectores, después vamos incrementando y mostrando el vector “a” mientras este sea menor en magnitud que “b”, notamos que el vector “a” va actualizando su valor. Para probar los condicionales, primero probamos si el vector “a” es menor al vector “b”, si es así mostramos una suma del vector “a” consigo mismo, vemos que como actualizamos (debido a los incrementos en el while) el valor del vector “a”, la condición no se cumple por lo que no se ejecuta la acción. Mientras que si probamos al revés vemos que ejecuta la instrucción que se encuentra dentro del if, que en este caso es mostrar la suma del vector “b”, consigo mismo.

3. Conclusiones

El manejo correcto de las instrucciones stop, stmt y, por supuesto de los tokens desde el archivo “.y”, nos permiten poder realizar correctamente los saltos y el uso de los ciclos que implica el uso de Hoc 5. La adición de nuevos tipos de sentencias a la calculadora fue bastante interesante ya que aprendimos a manejar el programa de manera interna, tomando en cuenta cómo funcionan internamente (en la pila de memoria) el ciclo while y la condicional if.