**Instituto Politécnico Nacional.**

**Escuela Superior De Cómputo.**





**Materia:**

**Compiladores.**

**Tema:**

**Práctica 04.**

**(Reporte).**

**Profesor:**

**Roberto Tecla Parra.**

**Alumno:**

**Mario Alberto Miranda Sandoval.**

**Grupo:**

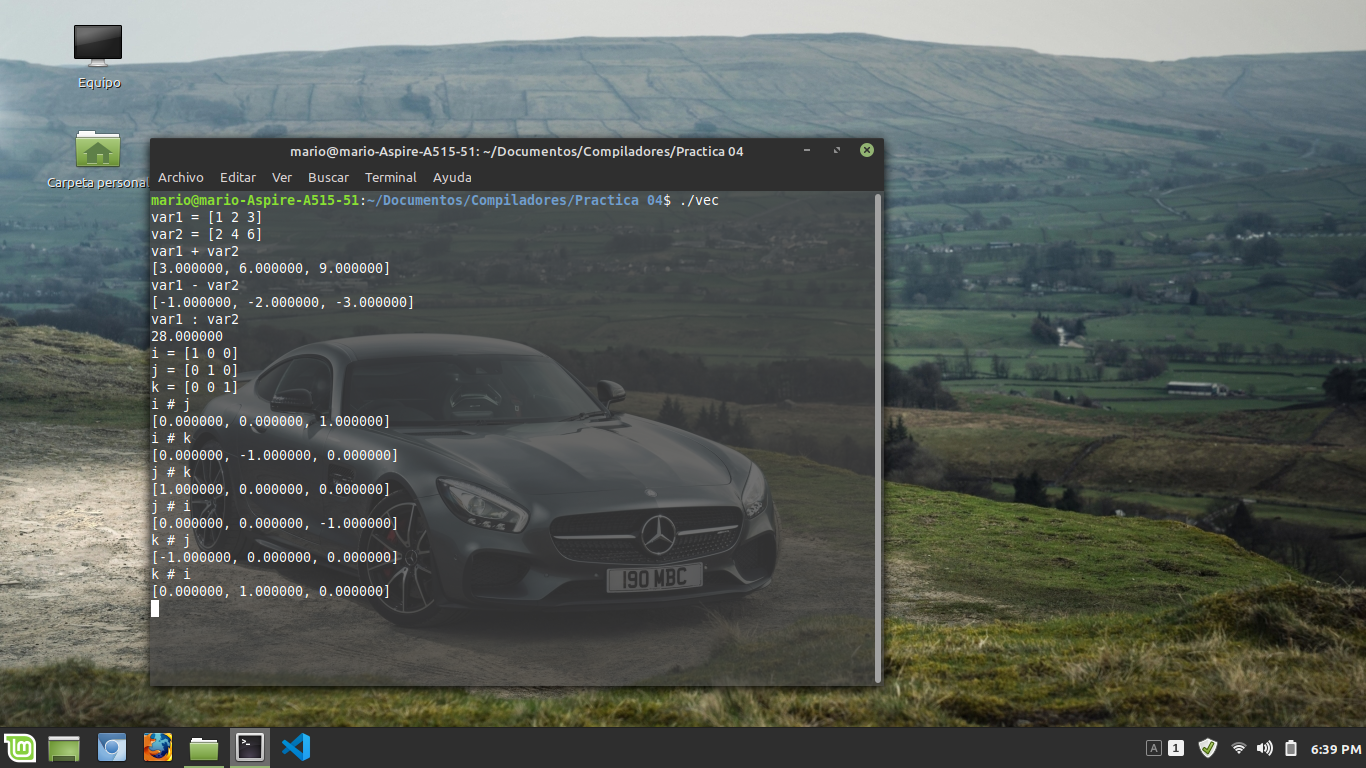
**3CM7.**

**Objetivos.**

En esta cuarta práctica se ha añadido la máquina virtual de pila. Para poder añadir la máquina virtual de pila es necesario crear un arreglo el cual nos servirá para simular nuestra pila.

Además, se han añadido algunas macros las cuales nos ayudarán al funcionamiento del programa. También se han añadido algunas funciones las cuales nos sirven al momento de la ejecución del código del programa.

**Pruebas.**



**Modificación al código.**

1. %%
2. list:
3. | list '\n'
4. | list asgn '\n'  { code2(pop, STOP); **return** 1; }
5. | list expr '\n'   { code2(print, STOP); **return** 1;}
6. | list escalar '\n'  { code2(printd, STOP); **return** 1;}
7. | list error '\n' { yyerrok; }
8. ;
9. asgn:   VAR '=' expr { code3(varpush, (Inst)$1, assign);}
10. ;
11. expr:   vector {code2(constpush, (Inst)$1);}
12. | VAR  {code3(varpush, (Inst)$1, eval);}
13. | asgn
14. | expr '+' expr     { code(add); }
15. | expr '-' expr     { code(sub); }
16. | escalar '\*' expr   { code(escalar);}
17. | expr '\*' escalar {code(escalar);}
18. | expr '#' expr     { code(productoc); }
19. ;
21. escalar: numero {code2(constpushd, (Inst)$1);}
22. | expr ':' expr         { code(productop); }
23. | '|' expr '|'          { code(magnitud); }
24. ;

Las acciones gramaticales fueron cambiadas por las macros dependiendo su caso.

1. **void** add( )     /\*   sumar los dos elementos superiores de la pila   \*/
2. {
3. Datum d1,   d2;
4. d2  =  pop();
5. d1  =  pop();
7. d1.val=  sumaVector(d1.val, d2.val);
8. push(d1);
9. }
11. **void** sub()
12. {
13. Datum d1,  d2;
14. d2  = pop();
15. d1  = pop();
16. d1.val=  restaVector(d1.val, d2.val);
17. push(d1);
18. }
20. **void** escalar()
21. {
22. Datum d1, d2;
23. d2 = pop();
24. d1 = pop();
25. d1.val = escalarVector(d1.num, d2.val);
26. push(d1);
27. }

30. **void** productop( )
31. {
32. Datum d1, d2;
33. **double** d3;
34. d2 = pop();
36. d1 = pop();
38. d3 = productoPuntoVector(d1.val, d2.val);
39. push((Datum)d3);
40. }
42. **void** productoc( )
43. {
44. Datum d1, d2;
45. d2 = pop();
47. d1 = pop();
49. d1.val = productoCruzVector(d1.val, d2.val);
50. push(d1);
51. }
53. **void** magnitud( )
54. {
55. Datum d1;
57. d1 = pop();
59. d1.num = magnitudVector(d1.val);
60. push(d1);
61. }
63. **void** assign( )        /\* asignar el valor superior al siguientevalor \*/
64. {
65. Datum d1, d2;
66. d1 = pop();
67. d2 = pop();
68. **if** (d1.sym->type != VAR && d1.sym->type != INDEF)
69. execerror("assignment to non-variable", d1.sym->name);
70. d1.sym->u.val = d2.val;
71. d1.sym->type = VAR;
72. push(d2);
73. }
75. **void** print( )  /\* sacar el valor superior de la pila e imprimirlo \*/
76. {
77. Datum d;
78. d = pop();
79. imprimeVector(d.val);
80. }
81. **void** printd( )  /\* sacar el valor superior de la pila e imprimirlo \*/
82. {
83. Datum d;
84. d = pop();
85. printf("%lf",d.num);
86. }

Y de aquí se des empilan los vectores de la pila y se hacen sus operaciones correspondientes.

**Código.**

**Vectores.y**

1. %{
3. #include “hoc.h”
4. #include <math.h>
5. #include <string.h>
6. #define MSDOS
8. #define code2(c1, c2)     code(c1); code(c2);
9. #define code3(c1, c2, c3) code(c1); code(c2); code(c3);
11. **void** yyerror (**char** \*s);
12. **int** yylex ();
13. **void** warning(**char** \*s, **char** \*t);
14. **void** execerror(**char** \*s, **char** \*t);
15. **void** fpecatch();
16. **extern** **void** init();
18. %}
19. %**union** {
20. **double** num;
21. Vector \*val;
22. Inst \*inst;
23. Symbol \*sym;
24. }
26. %token <num> NUMBER
27. %token <sym> VAR INDEF VECT NUMB
28. %type <sym> vector numero
29. %type  <val> expr asgn
30. %type <num> escalar
32. %right ‘=’
33. %left ‘+’ ‘-‘
34. %left ‘\*’ ‘#’ ‘|’ ‘:’
36. %%
37. list:
38. | list ‘\n’
39. | list asgn ‘\n’  { code2(pop, STOP); **return** 1; }
40. | list expr ‘\n’   { code2(print, STOP); **return** 1;}
41. | list escalar ‘\n’  { code2(printd, STOP); **return** 1;}
42. | list error ‘\n’ { yyerrok; }
43. ;
44. asgn:   VAR ‘=’ expr { code3(varpush, (Inst)$1, assign);}
45. ;
46. expr:   vector {code2(constpush, (Inst)$1);}
47. | VAR  {code3(varpush, (Inst)$1, eval);}
48. | asgn
49. | expr ‘+’ expr     { code(add); }
50. | expr ‘-‘ expr     { code(sub); }
51. | escalar ‘\*’ expr   { code(escalar);}
52. | expr ‘\*’ escalar {code(escalar);}
53. | expr ‘#’ expr     { code(productoc); }
54. ;
56. escalar: numero {code2(constpushd, (Inst)$1);}
57. | expr ‘:’ expr         { code(productop); }
58. | ‘|’ expr ‘|’          { code(magnitud); }
59. ;
61. vector: ‘[‘ NUMBER NUMBER NUMBER ‘]’ {Vector \*vector1= creaVector(3); vector1->vec[0] = $2; vector1->vec[1] = $3; vector1->vec[2] = $4; $$ = install(“”, VECT , vector1);}
62. ;
63. numero: NUMBER { $$ = installd(“”, NUMB,$1); }
65. %%
67. #include <stdio.h>
68. #include <ctype.h>
69. #include <signal.h>
70. #include <setjmp.h>
72. **jmp\_buf** begin;
74. **char** \*progname;
75. **int** lineno = 1;
77. **void** main (**int** argc, **char** \*argv[]){
78. progname=argv[0];
80. setjmp(begin);
81. signal(SIGFPE, fpecatch);
82. **for**(initcode(); yyparse (); initcode())
83. execute(prog);
85. }
87. **void** execerror(**char** \*s, **char** \*t){
88. warning(s, t);
89. longjmp(begin, 0);
90. }
92. **void** fpecatch(){
93. execerror(“íneaión de punto flotante”, (**char** \*)0);
94. }
96. **int** yylex (){
97. **int** c;
98. **while** ((c = getchar ()) == ‘ ‘ || c == ‘\t’)
99. ;
100. **if** (c == EOF)
101. **return** 0;
102. **if** (c == ‘.’ || isdigit ©){
103. **double** d;
104. ungetc (c, stdin);
105. scanf (“%lf”, &yylval.num);
106. **return** NUMBER;
107. /\*ungetc (c, stdin);
108. scanf (“%lf”, &yylval.num);
109. return NUMBER;\*/
110. }
111. **if**(isalpha©){
112. Symbol \*s;
113. **char** sbuf[200], \*p=sbuf;
114. **do** {
115. \*p++=c;
116. } **while** ((c=getchar())!=EOF && isalnum©);
117. ungetc(c, stdin);
118. \*p=’\0’;
119. **if**((s=lookup(sbuf))==(Symbol \*)NULL)
120. s=install(sbuf, INDEF,NULL);
121. yylval.sym=s;
122. **if**(s->type == INDEF){
123. **return** VAR;
124. } **else** {
125. //printf(“func=(%s) tipo=(%d) \n”, s->name, s->type);
126. **return** s->type;
127. }
128. }
129. **if**(c == ‘\n’)
130. lineno++;
131. **return** c;
132. }
134. **void** yyerror (**char** \*s)  /\* Llamada por yyparse ante un error \*/
135. {
136. warning(s, (**char** \*) 0);
137. }
139. **void** warning(**char** \*s, **char** \*t)
140. {
141. fprintf (stderr, “%s: %s”, progname, s);
142. **if**(t)
143. fprintf (stderr, “ %s”, t);
144. fprintf (stderr, “cerca de la ínea %d\n”, lineno);
145. }

**Code.c**

1. #include "hoc.h"
2. #include "y.tab.h"
3. #define NSTACK  256
4. **static**  Datum  stack[NSTACK];  /\* la pila \*/
5. **static**  Datum   \*stackp;       /\* siguiente lugar libre en la pila \*/
6. #define NPROG   2000
7. Inst    prog[NPROG];    /\* la máquina \*/
8. Inst    \*progp;         /\* siguiente lugar libre para la generación de código \*/
9. Inst    \*pc;            /\* contador de programa durante la ejecución \*/

12. **void** initcode()      /\* inicialización para la generación de código \*/
13. {
14. stackp = stack;
15. progp = prog;
16. }
18. **void** push(d)        /\*  meter d en la pila  \*/
19. Datum d;
20. {
21. **if** (stackp >= &stack[NSTACK])
22. execerror("stack overflow", (**char** \*) 0);
23. \*stackp++ = d;
24. }
26. Datum pop( )         /\* sacar y retornar de la pila el elemento del tope \*/
27. {
28. **if** (stackp <= stack)
29. execerror("stack underflow", (**char** \*) 0);
30. **return**  \*--stackp;
31. }

34. **void** constpush( )   /\* meter una constante a la pila  \*/
35. {
36. Datum d;
37. d.val = ((Symbol  \*)\*pc++)->u.val;
38. push(d);
39. }
41. **void** constpushd( )  /\* meter una constante a la pila  \*/
42. {
43. Datum d;
44. d.num = ((Symbol  \*)\*pc++)->u.num;
45. push(d);
46. }
47. **void** varpush()  /\* meter una variable a la pila   \*/
48. {
49. Datum d;
50. d.sym  =  (Symbol   \*)(\*pc++);
51. push(d);
52. }
54. **void** eval( )    /\*  evaluar una variable en la pila   \*/
55. {
56. Datum  d;
57. d   =  pop();
58. **if**   (d.sym->type   ==   INDEF)
59. execerror("undefined variable",d.sym->name);
60. d.val   =  d.sym->u.val;
61. push(d);
62. }
64. **void** add( )     /\*   sumar los dos elementos superiores de la pila   \*/
65. {
66. Datum d1,   d2;
67. d2  =  pop();
68. d1  =  pop();
70. d1.val=  sumaVector(d1.val, d2.val);
71. push(d1);
72. }
74. **void** sub()
75. {
76. Datum d1,  d2;
77. d2  = pop();
78. d1  = pop();
79. d1.val=  restaVector(d1.val, d2.val);
80. push(d1);
81. }
83. **void** escalar()
84. {
85. Datum d1, d2;
86. d2 = pop();
87. d1 = pop();
88. d1.val = escalarVector(d1.num, d2.val);
89. push(d1);
90. }

93. **void** productop( )
94. {
95. Datum d1, d2;
96. **double** d3;
97. d2 = pop();
99. d1 = pop();
101. d3 = productoPuntoVector(d1.val, d2.val);
102. push((Datum)d3);
103. }
105. **void** productoc( )
106. {
107. Datum d1, d2;
108. d2 = pop();
110. d1 = pop();
112. d1.val = productoCruzVector(d1.val, d2.val);
113. push(d1);
114. }
116. **void** magnitud( )
117. {
118. Datum d1;
120. d1 = pop();
122. d1.num = magnitudVector(d1.val);
123. push(d1);
124. }
126. **void** assign( )        /\* asignar el valor superior al siguientevalor \*/
127. {
128. Datum d1, d2;
129. d1 = pop();
130. d2 = pop();
131. **if** (d1.sym->type != VAR && d1.sym->type != INDEF)
132. execerror("assignment to non-variable", d1.sym->name);
133. d1.sym->u.val = d2.val;
134. d1.sym->type = VAR;
135. push(d2);
136. }
138. **void** print( )  /\* sacar el valor superior de la pila e imprimirlo \*/
139. {
140. Datum d;
141. d = pop();
142. imprimeVector(d.val);
143. }
144. **void** printd( )  /\* sacar el valor superior de la pila e imprimirlo \*/
145. {
146. Datum d;
147. d = pop();
148. printf("%lf",d.num);
149. }

152. Inst   \*code(Inst f) /\*   instalar una instrucción u operando   \*/
153. {
154. Inst \*oprogp = progp;
155. **if** (progp >= &prog [ NPROG ])
156. execerror("program too big", (**char** \*) 0);
157. \*progp++ = f;
158. **return** oprogp;
159. }
161. **void** execute(Inst p)    /\*   ejecución con la máquina   \*/
162. {
163. **for**  (pc  =  p;   \*pc != STOP; )
164. (\*(\*pc++))();
165. }