Compiladores

- Práctica 04: Máquina virtual de pila -

Grupo 3CM7

Vargas Romero Erick Efraín
Prof. Tecla Parra Roberto

Instituto Politécnico Nacional
Escuela Superior de Cómputo
Juan de Dios Bátiz, nueva industrial Vallejo
07738 ciudad de México

Chapter 1

Práctica 04

1.1 Maquina virtual de pila

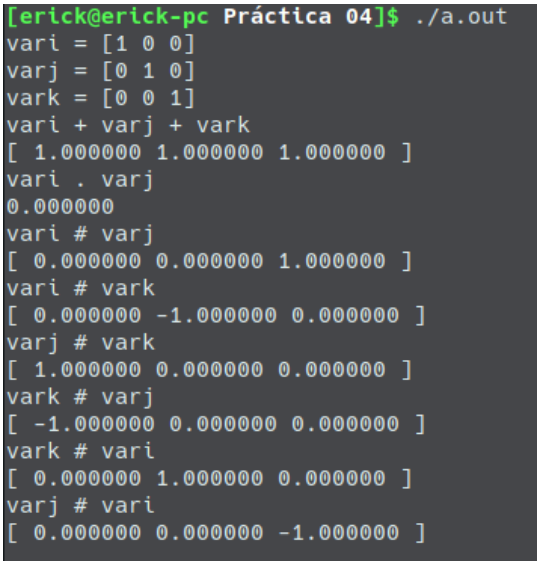
1.1.1 Descripción

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.

1.1.2 Ejemplos

A continuación muestro una captura de pantalla, la cual muestra la compilación del código en yacc, y también la compilación del código que es generado en c y finalmente la ejecución del programa.

Figure 1.1: Ejemplo



1.1.3 Código

Primeramente se han modificado todas las acciones gramaticales de nuestro programa anterior, esto con la finalidad de generar código que será ejecutado más adelante, el código se va a nuestra máquina virtual de pila. Además se han añadido algunos elementos a la union donde tenemos la definición de tipos de dato de la pila de YACC

```
1  //óDefinicin de tipos de dato de la pila de yacc
2  %union{
3      double comp;
4      Vector* vec;
5      //ñAadida en la áprctica 3
6      Symbol* sym;
7      //ñAadida en la áprctica 4
8      Inst* inst;
9  }
10
11 /**óCreacin de ísmbolos terminales y no terminales**/
12 %token<comp>     NUMBER      //íSmbolo terminal
13 %type<vec>       exp         //íSmbolo no terminal
14 %type<sym>       vect        //íSmbolo no terminal
15 %type<sym>      number      //íSmbolo no terminal
16 //NUEVOS ÍSMBOLOS GRAMATICALES PARA LA ÁPRCTICA 3
17 %token<sym>      VAR         //íSmbolo terminal
18 %token<sym>      INDEF       //íSmbolo terminal
19 %type<vec>       asgn        //íSmbolo no terminal
20 //NUEVOS ÍSMBOLOS GRAMATICALES PARA LA ÁPRCTICA 4
21 %type<comp>      escalar
22 %token<sym>      VECT
```

```
23 %token<sym>      NUMB
24
25 /**íJerarqua de operadores**/
26
27 //Para áprctica 1
28 %right '='
29 //Suma y resta de vectores
30 %left '+' '-'
31 //Escalar por un vector
32 %left '*'
33 //Producto cruz y producto punto
34 %left '#' '.' '|'
35 /**áGramtica**/
36 %%
37
38     list:
39       | list '\n'
40       | list asgn '\n'    {code2(pop, STOP); return 1;}
41       | list exp '\n'     {code2(print, STOP); return 1;}
42       | list escalar '\n'  {code2(printd, STOP); return 1;}
43       | list error '\n'   {yyerror;}
44       ;
45
46     asgn: VAR '=' exp     {code3(varpush, (Inst)$1, assign);}
47     ;
48
49     exp: vect             {code2(constpush, (Inst)$1);}
50       | VAR               {code3(varpush, (Inst)$1, eval);}
51       | asgn
52       | exp '+' exp       {code(add);}
53       | exp '-' exp       {code(sub);}
54       | escalar '*' exp    {code(escalar);}
55       | exp '*' escalar    {code(escalar);}
56       | exp '#' exp       {code(producto_cruz);}
57     ;
58
59     escalar: number  {code2(constpushd, (Inst)$1);}
60       | exp '.' exp {code(producto_punto);}
61       | '|' exp '|' {code(magnitud);}
62       ;
63
64     vect: '[' NUMBER NUMBER NUMBER ']'    {  Vector* v = creaVector(3);
65                                              v -> vec[0] = $2;
66                                              v -> vec[1] = $3;
67                                              v -> vec[2] = $4;
68                                              $$ = install("", VECT, v);}
69       ;
70
71     number: NUMBER {$$ = installd("", NUMB, $1);}
72     ;
73 %%
```

Después tendriamos todo lo relacionado con la máquina virtual de pila, lo cual

se encuentra en el archivo code.c

```
1  #include "hoc.h"
2  #include "y.tab.h"
3  #include <stdio.h>
4  #define NSTACK 256
5  static Datum stack[NSTACK];      /* Pila */
6  static Datum *stackp;            /* Tope de la Pila*/
7  #define NPROG 2000
8  Inst prog[NPROG];    /* La ámquina virtual - RAM: Se guardan las
       instrucciones */
9  Inst *progp;         /* Siguiente lugar libre para la ógeneracin de
       ócdigo:
10                         Dice en donde se guarda nuestra proxima
                               instruccion */
11 Inst *pc;            /* Contador del programa durante la óejecucin */
12
13 void initcode(){
14     stackp = stack;      /* Apunta al inicio del arreglo */
15     progp = prog;        /* Se guarda la ódireccin del primer
16                             elemento del arreglo. */
17 }
18
19 void push(d)
20     Datum d;
21 {      /* Se mete d en la pila*/
22     if( stackp >= &stack[NSTACK] )
23         execerror("stack overflow", (char *) 0);
24     *stackp++ = d;
25 }
26
27 Datum pop(){             /* Sacar y retornar de la pila el elemento */
28     if( stackp <= stack )
29         execerror("stack underflow", (char *) 0);
30     return *--stackp;
31 }
32
33 void constpush(){        /* Meter una constante en la pila*/
34     Datum d;
35     d.val = ((Symbol *)*pc++)->u.vec;    /* Apunta a la entarda de
36                                             la tabla de simbolos */
37     push(d);
38 }
39
40 void constpushd(){        /* Meter una constante en la pila*/
41     Datum d;
42     d.num = ((Symbol  *)*pc++)->u.comp;    /* Apunta a la entarda de
43                                               la tabla de simbolos */
44     push(d);
45 }
46
47 void varpush(){      /* Meter una variable a la pila */
48     Datum d;         /* Los elementos de la maquina virtual
49                         de la pila son de tipo Datum */
```

```
50      d.sym = (Symbol   *)(*pc++);   /* Convertirmos a Symbol, lo guardamos
51                                        en .sym y lo metemos en la pila */
52      push(d);
53  }
54
55  void eval( ){
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.vec;
61      push(d);
62  }
63
64  /* ==== FUNCIONES PARA LOS VECTORES ==== */
65  void add(){
66      Datum d1, d2;
67      d2 = pop();
68      d1 = pop();
69      d1.val = sumaVector(d1.val, d2.val);
70      push(d1);
71  }
72
73  void sub(){
74      Datum d1, d2;
75      d2 = pop();
76      d1 = pop();
77      d1.val = restaVector(d1.val, d2.val);
78      push(d1);
79  }
80
81  void escalar(){
82      Datum d1, d2;
83      d2 = pop();
84      d1 = pop();
85      d1.val = escalarVector(d1.num, d2.val);
86      push(d1);
87  }
88
89  void producto_punto(){
90      Datum d1, d2;
91      double d3;
92      d2 = pop();
93      d1 = pop();
94      d3 = productoPunto(d1.val, d2.val);
95      push((Datum)d3);
96  }
97
98  void producto_cruz(){
99      Datum d1, d2;
100     d2 = pop();
101     d1 = pop();
102     d1.val = productoCruz(d1.val, d2.val);
```

```
103     push(d1);
104 }
105
106 void magnitud(){
107     Datum d1;
108     d1 = pop();
109     d1.num = vectorMagnitud(d1.val);
110     push(d1);
111 }
112
113 void assign( ){       /* Asigna el valor superior al siguiente valor */
114     Datum d1, d2;
115     d1 = pop();
116     d2 = pop();
117     if(d1.sym->type != VAR && d1.sym->type != INDEF)
118         execerror("assignment to non-variable", d1.sym->name);
119     d1.sym->u.vec = d2.val;
120     d1.sym->type = VAR;
121     push(d2);
122 }
123
124 void print(){        /* Se saca el valor del tope de la pila y se imprime
     */
125     Datum d;
126     d = pop();
127
128     imprimeVector(d.val);
129 }
130
131 void printd(){        /* Se saca el valor del tope de la pila y se
    imprime */
132     Datum d;
133     d = pop();
134     printf("%lf\n",d.num);
135 }
136
137 Inst *code(Inst f){     /* Recibe una instruccion u operando y la guarda
    en la RAM */
138     Inst *oprogp = progp;     //EL valor actual se guarda en oprogp
139     if (progp >= &prog [ NPROG ])     //Verifica si hay espacio en la RAM
140     execerror("program too big", (char *) 0);
141     *progp++ = f;            /* La instruccion f se guarda en la RAM
142                                 y avanzamos progp*/
143     return oprogp;
144 }
145
146 void execute( Inst* p){       /*Ejecuta instrucciones de la ámquina/RAM
    */
147     for( pc = p; *pc != STOP; ) /*Especificamos donde inicia la
        óejecucin y se
148                                   detiene en donde haya un STOP */
149         (*(*pc++))();              /*Ejecutamos una ófuncin */
150
```

```
151 }
```

Además se ha modificado hoc.h añadiendo una estructura llamada Datum la cual contiene un apuntador a Vector, un valor double y un apuntador a Symbol

```
1  #include "vector_cal.h"
2  #include <string.h>
3  typedef struct Symbol { /* entrada de la tabla de simbolos */
4    char* name;
5    short type;    /* VAR, BLTIN, UNDEF */
6    union {
7      double comp;          /* si es VAR */
8      double  (*ptr)();       /* si es BLTIN */
9      Vector* vec;
10   } u;
11   struct Symbol* next;  /* para ligarse a otro */
12 } Symbol;
13
14 Symbol* install(char* s, int t, Vector* vec);
15 Symbol* lookup(char* s);
16
17 /****************** ÁPRCTICA CUATRO ********************/
18 typedef union Datum{
19   Vector* val;
20   double num;
21   Symbol* sym;    /*Apunta a una entrada en la tabla de simbolos */
22 }Datum;
23
24 extern Datum pop();
25 typedef int (*Inst)(); /* Instruccion de maquina:
26                           Es un apuntador a funcion*/
27
28 #define STOP (Inst) 0
29 extern Inst prog[];
30
31 extern void assign();
32 extern void varpush();
33 extern void constpush();
34 extern void print();
35 extern void printd();
36 extern void constpushd();
37
38 extern void eval();
39 extern void add();
40 extern void sub();
41 extern void producto_cruz();
42 extern void producto_punto();
43 extern void magnitud();
44 extern void escalar();
```