NSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE COMPUTO

PRACTICA 4 MV de Pila

Opción vectores

COMPILADORES

Grupo: 3CM17

ALUMNO: MORA GUZMAN JOSE ANTONIO

FECHA ENTREGA: Lunes 1 noviembre 2021

Descripción

En esta cuarta práctica añade 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.

Código

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 unión donde tenemos la definición de tipos de dato de la pila de YACC

```
26      extern void init();
27  %}
28  //Definición de tipos de dato de la pila de yacc
29  %union{
30      double comp;
31      Vector* vec;
32      //Añadida en la práctica 3
33      Symbol* sym;
34      //Añadida en la práctica 4
35      Inst* inst;
36  }
37
38
39  %token<comp>    NUMBER
40  %type<vec>      exp
41  %type<sym>      vect
42  %type<sym>     number
43
44  %token<sym>     VAR
45  %token<sym>     INDEF
46  %type<vec>      asgn
47
48  %type<comp>     escalar
49  %token<sym>     VECT
50  %token<sym>     NUMB
51
52  /**Jerarquía de operadores**/
53
54  //Para práctica 1
55  %right '='
56  //Suma y resta de vectores
57  %left '+' '-'
58  //Escalar por un vector
59  %left '*'
60  //Producto cruz y producto punto
61  %left '#' '.' '|'
62  /**Gramática**/
63  %%
64
```

```
list:
  | list '\n'
  | list asgn '\n'    {code2(pop, STOP); return 1;}
  | list exp '\n'     {code2(print, STOP); return 1;}
  | list escalar '\n'  {code2(printd, STOP); return 1;}
  | list error '\n'   {yyerror;}
  ;

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

exp: vect           {code2(constpush, (Inst)$1);}
  | VAR             {code3(varpush, (Inst)$1, eval);}
  | asgn
  | exp '+' exp     {code(add);}
  | exp '-' exp     {code(sub);}
  | escalar '*' exp  {code(escalar);}
  | exp '*' escalar  {code(escalar);}
  | exp '#' exp     {code(producto_cruz);}
;

escalar: number  {code2(constpushd, (Inst)$1);}
  | exp '.' exp {code(producto_punto);}
  | '|' exp '|' {code(magnitud);}
  ;

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

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

Todo lo relacionado con la máquina virtual de pila se encuentra en el archivo code.c

```
#include "hoc.h"
#include "y.tab.h"
#include <stdio.h>
#define NSTACK 256
static Datum stack[NSTACK];
static Datum *stackp;
#define NPROG 2000
Inst prog[NPROG];
Inst *progp;

Inst *pc;

void initcode(){
    stackp = stack;
    progp = prog;

}

void push(d)
    Datum d;
{
    if( stackp >= &stack[NSTACK] )
        execerror("stack overflow", (char *) 0);
    *stackp++ = d;
}

Datum pop(){
    if( stackp <= stack )
        execerror("stack underflow", (char *) 0);
    return *--stackp;
}

void constpush(){
    Datum d;
    d.val = ((Symbol *)*pc++)->u.vec;

    push(d);
}

void constpushd(){
    Datum d;
    d.num = ((Symbol  *)*pc++)->u.comp;

    push(d);
}

void varpush(){
    Datum d;

    d.sym = (Symbol  *)(*pc++);

    push(d);
}

void eval( ){
    Datum d;
    d = pop();
    if( d.sym->type == INDEF )
        execerror("undefined variable",d.sym->name);
    d.val = d.sym->u.vec;
    push(d);
}


void add(){
    Datum d1, d2;
    d2 = pop();
    d1 = pop();
    d1.val = sumaVector(d1.val, d2.val);
    push(d1);
}

void sub(){
    Datum d1, d2;
    d2 = pop();
    d1 = pop();
    d1.val = restaVector(d1.val, d2.val);
    push(d1);
}
```

```
91  void escalar(){
92      Datum d1, d2;
93      d2 = pop();
94      d1 = pop();
95      d1.val = escalarVector(d1.num, d2.val);
96      push(d1);
97  }
98
99  void producto_punto(){
100     Datum d1, d2;
101     double d3;
102     d2 = pop();
103     d1 = pop();
104     d3 = productoPunto(d1.val, d2.val);
105     push((Datum)d3);
106 }
107
108 void producto_cruz(){
109     Datum d1, d2;
110     d2 = pop();
111     d1 = pop();
112     d1.val = productoCruz(d1.val, d2.val);
113     push(d1);
114 }
115
116 void magnitud(){
117     Datum d1;
118     d1 = pop();
119     d1.num = vectorMagnitud(d1.val);
120     push(d1);
121 }
122
123 void assign( ){      /* Asigna el valor superior al siguiente valor */
124     Datum d1, d2;
125     d1 = pop();
126     d2 = pop();
127     if(d1.sym->type != VAR && d1.sym->type != INDEF)
128         execerror("assignment to non-variable", d1.sym->name);
129     d1.sym->u.vec = d2.val;
130     d1.sym->type = VAR;
131     push(d2);
132 }
```

```
134 void print(){       /* Se saca el valor del tope de la pila y se imprime */
135     Datum d;
136     d = pop();
137
138     imprimeVector(d.val);
139 }
140
141 void printd(){       /* Se saca el valor del tope de la pila y se imprime */
142     Datum d;
143     d = pop();
144     printf("%lf\n",d.num);
145 }
146
147 Inst *code(Inst f){
148     Inst *oprogp = progp;
149     if (progp >= &prog [ NPROG ])
150         execerror("program too big", (char *) 0);
151     *progp++ = f;
152
153     return oprogp;
154 }
155
156 void execute( Inst* p){
157     for( pc = p; *pc != STOP; )
158
159         (*(*pc++))();
160
161 }
162
```

Tambien se modifico hoc.h y se agrega una estructura llamada Datum la cual contiene un apuntador a Vector, un valor double y un apuntador a Symbol.

```
25
26  /****************** PRÁCTICA CUATRO *********************/
27  typedef union Datum{
28    Vector* val;
29    double num;
30    Symbol* sym;   /*Apunta a una entrada en la tabla de simbolos */
31  }Datum;
32
33  extern Datum pop();
34  typedef int (*Inst)(); /* Instruccion de maquina:
35                            Es un apuntador a funcion*/
36
37  #define STOP (Inst) 0
38  extern Inst prog[];
39
40  extern void assign();
41  extern void varpush();
42  extern void constpush();
43  extern void print();
44  extern void printd();
45  extern void constpushd();
46
47  extern void eval();
48  extern void add();
49  extern void sub();
50  extern void producto_cruz();
51  extern void producto_punto();
52  extern void magnitud();
53  extern void escalar();
54
```

Pruebas del programa

```
tony@tony-Aspire-E5-523:~/Escritorio/compiladores/Practica 4$ ./a.out
a=[0 1 0]
b=[1 1 1]
c=[ 5 5 5]
a
[ 0.000000 1.000000 0.000000 ]
b
[ 1.000000 1.000000 1.000000 ]
c
[ 5.000000 5.000000 5.000000 ]
a+b+c
[ 6.000000 7.000000 6.000000 ]
a#b
[ 1.000000 0.000000 -1.000000 ]
a.c
5.000000
|c|
8.660254
c#a
[ -5.000000 0.000000 5.000000 ]
b.a
1.000000
a-b-c
[ -6.000000 -5.000000 -6.000000 ]
c-a-b
[ 4.000000 3.000000 4.000000 ]
```

conclusiones

Esta practica igual que la anterior fue sencilla puesto que solo se deben agregar y modificar algunas cosas para que funcione y pueda ser continuacion de las practicas anteriores y en este caso fue el momento de agregar la maquina virtual de pila

Link del video

https://youtu.be/TI5q7srFisI