INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Nombre: Cruz López Adrián

Grupo: 3CM15

Asignatura: Compiladores

Profesor: Roberto Tecla Parra

Actividad: Práctica 4 Máquina Virtual de Pila "*Calculadora para Vectores*"

Fecha: 05/11/2021

INTRODUCCIÓN

Máquina virtual de pila

Una máquina de pila es un modelo computacional en el cual la memoria de la computadora toma la forma de una o más pilas. El término también se refiere a un computador real implementando o simulando una máquina de pila idealizada.

Adicionalmente, una máquina de pila también puede referirse a una máquina verdadera o simulada con un conjunto de instrucciones de "0 operandos". En tal máquina, la mayoría de las instrucciones implícitamente operan en valores en el tope de la pila y reemplazan esos valores por el resultado. Típicamente tales máquinas también tienen una instrucción "load" y una instrucción "store" que leen y escriben a posiciones arbitrarias de la RAM. (Como el resto de las instrucciones, las instrucciones "load" y "store" no necesitan ningún operando en una máquina de pila típica - ellas siempre toman la dirección de la RAM que se quiere leer o escribir desde el tope de la pila).

La ventaja de las máquinas de pila ("conjunto de instrucciones de 0 operandos") sobre las máquinas de acumulador ("conjunto de instrucciones de 1 operando") y las máquinas de registro ("conjunto de instrucciones de 2 operandos" o un "conjunto de instrucciones de 3 operandos") es que los programas escritos para un conjunto de instrucciones de "0 operandos" generalmente tienen una densidad de código más alta que los programas equivalentes escritos para otros conjuntos de instrucciones.

YACC

Es un programa para generar analizadores sintácticos. Las siglas del nombre significan Yet Another Compiler-Compiler, es decir, "Otro generador de compiladores más". Genera un analizador sintáctico (la parte de un compilador que comprueba que la estructura del código fuente se ajusta a la especificación sintáctica del lenguaje) basado en una gramática analítica escrita en una notación similar a la BNF. YACC genera el código para el analizador sintáctico en el Lenguaje de programación C.

Puesto que el analizador sintáctico generado por YACC requiere un analizador léxico, se utiliza a menudo juntamente con un generador de analizador léxico, en la mayoría de los casos lex o Flex, alternativa del software libre. El estándar de IEEE POSIX P1003.2 define la funcionalidad y los requisitos a Lex y YACC.

DESARROLLO

Se emplearán los mismos archivos que se utilizaron en la practica 3, por lo tanto en la gramática solo se realizaron algunos cambios menores, lo que cambió más fueron las acciones gramaticales, algunas de ellas se muestran a continuación:

```
57.     exp: vect             {code2(constpush, (Inst)$1);}
58.         | VAR              {code3(varpush, (Inst)$1, eval);}
59.         | asgn
60.         | exp '+' exp      {code(add);} //Caso SUMA
61.         | exp '-' exp      {code(sub);} //Caso RESTA
62.         | NUMBER '*' exp   {code(escalar);} //Caso MULT por ESCALAR 1
63.         | exp '*' NUMBER   {code(escalar);} //Caso MULT por ESCALAR 2
64.         | exp 'x' exp      {code(producto_cruz);} //Caso PROD CRUZ
65.         ;
66.     escalar: number       {code2(constpushd, (Inst)$1);}
67.         | exp '.' exp {code(producto_punto);} //Caso PROD PUNTO
68.         | '|' exp '|' {code(magnitud);} //Caso MAGNITUD
69.         ;
70.     vect: '[' NUMBER NUMBER NUMBER ']'  {Vector* v = creaVector(3);
71.                                                 v -> vec[0] = $2;
72.                                                 v -> vec[1] = $3;
73.                                                 v -> vec[2] = $4;
74.                                                 $$ = install("", VECT, v);}
75.         ;
```

Se puede observar que las acciones en los casos donde se debe realizar una operación se ligan a la función correspondiente según su caso, las cuales se encuentran en el archivo de *"code.c"* y que hacen uso de la pila para sacar dichos valores y realizar la operación.

Con los pequeños cambios realizados a la gramática, las producciones de la gramática de una forma más entendible serían:

list → ε

list → list '\n'

list → list asgn '\n'

list → list exp '\n'

list → list escalar '\n'

list → list error '\n'

exp → vect

exp → VAR

exp → asgn

exp → exp + exp

exp → exp - exp

exp → NUMBER * exp

escalar → number

escalar → exp . exp

escalar → | exp |

Vector → [NUMBER NUMBER NUMBER]

asgn → VAR = exp

number → NUMBER

exp → exp * NUMBER

exp → exp x exp

Una sintaxis más familiar que, aunque tiene recursividad por la izquierda, se resuelve con la jerarquía planteada en el mismo archivo. Los símbolos de +, -, *, etc., solamente se colocan entre comillas simples y no se declaran como token, debido a que tienen longitud de 1.

La jerarquía manejada se mantiene como en las prácticas anteriores:

%right '='
%left '+' '-'
%left '*'
%left UNARYMINUS
%left 'x' '.'
Tanto el producto cruz como el producto punto tienen mayor precedencia y todas asocian por la izquierda a excepción del operador de asignación "=".

Los archivos de *"vector_cal.c"*, tienen exactamente el mismo contenido que en los archivos de la práctica 3, debido a que se basó en ella para esta práctica. A continuación, se muestra una de sus funciones:

```
118.double vectorMagnitud(Vector* a){
119.    double resultado = 0.0f;
120.    int i;
121.    for(i = 0; i < a->n; i++){
122.        resultado += ( a -> vec[i] * a -> vec[i] );
123.    }
124.    return sqrt(resultado);
125.}
```

Es una función que retorna un dato tipo double, la cual recibe un apuntador a vector. Dentro del ciclo for se va multiplicando cada elemento del vector por sí mismo para obtener el cuadrado, y cada resultado se va guardando en la variable de tipo double "resultado", para que una vez terminado el ciclo se obtenga su raíz cuadrada y retorne dicho valor.

Algunas de las funciones que hacen uso de la pila son:

```
74. void add(){
75.     Datum d1, d2;
76.     d2 = pop();
77.     d1 = pop();
78.     d1.val = sumaVector(d1.val, d2.val);
79.     push(d1);
80. }
81.
82. void sub(){
83.     Datum d1, d2;
84.     d2 = pop();
85.     d1 = pop();
86.     d1.val = restaVector(d1.val, d2.val);
87.     push(d1);
88. }
```

Ambas funciones fueron modificadas para que se adaptaran al programa de la calculadora de vectores, una vez que las acciones gramaticales hagan uso de ellas, lo que hacen es sacar ambos valores recibidos de la pila, y en la parte de val de d1, guardar el resultado de la suma, usando la función de vectorSuma del archivo vector_cal.c.

Las funciones como el producto punto, producto cruz o magnitud, funcionan de la misma manera:

```
26. typedef union Datum{
27.   Vector* val;
28.   double num;
29.   Symbol* sym;
30. }Datum;
```

La unión da el tipo a los elementos de la pila, es decir, los elementos en la pila van a ser de tipo datum, de ahí que en el código anterior se declaren d1 y d2 como de tipo datum.

PRUEBAS DE FUNCIONAMIENTO

Suma y resta de vectores

La suma y resta de vectores, pueden realizarse tanto con números enteros como con números flotantes. El vector resultante se obtiene sumando o restando, (*dependiendo el caso*) los elementos en las mismas posiciones de cada vector; como se muestra en la siguiente imagen:

```
adrian@adrian-VirtualBox:~/Documentos/Práctica4$ ./pr4
[7 12 4] + [8 5 7]
[ 15.00 17.00 11.00 ]
[4.7 5.4 7.7] - [8.7 7.5 12.8]
[ -4.00 -2.10 -5.10 ]
```

Multiplicación por escalar

Es posible realizar la multiplicación cuando se encuentra primero el escalar o cuando esta primero el vector a multiplicar por el escalar, de igual manera el vector resultante se obtiene de multiplicar el escalar por cada uno de los elementos del vector:

```
adrian@adrian-VirtualBox:~/Documentos/Practica4$ ./pr4
17*[7 4 11]
[ 119.00 68.00 187.00 ]
[4 12 17]*15
[ 60.00 180.00 255.00 ]
```

Producto punto

El producto punto se obtiene multiplicando los elementos en las mismas posiciones y sumando los resultados de cada multiplicación.

```
adrian@adrian-VirtualBox:~/Documentos/Práctica4$ ./pr4
[3 7 9].[5 3 6]
90.000000
[4 7 2].[4 1 3]
29.000000
```

En el primer caso el producto punto es 3*5 + 7*3 + 9*6 = 15 + 21 + 54 = 90 y en el segundo es 4*4 + 7*1 + 2*3 = 16 + 7 + 6 = 29

Producto Cruz

El producto cruz de obtiene de la siguiente manera:

$$(a_1, a_2, a_3) \times (b_1, b_2, b_3) = [(a_2b_3 - a_3b_2), (a_3b_1 - a_1b_3), (a_1b_2 - a_2b_1)]$$

Entonces para el primer caso,

$$\begin{aligned}(5 \quad 1 \quad 7) \times (8 \quad 1 \quad 3) &= [(1*3 - 7*1) \quad (7*8 - 5*3) \quad (5*1 - 1*8)] \\ &= [(3 - 7) \quad (56 - 15) \quad (5 - 8)] \\ &= [-4 \quad 41 \quad -3]\end{aligned}$$

Para el segundo caso,

$$\begin{aligned}(1 \quad 7 \quad 5) \times (1 \quad 3 \quad 8) &= [(7*8 - 5*3) \quad (5*1 - 1*8) \quad (1*3 - 7*1)] \\ &= [(56 - 15) \quad (5 - 8) \quad (3 - 7)] \\ &= [41 \quad -3 \quad -4]\end{aligned}$$

Su funcionamiento se puede observar en la siguiente figura:

```
adrian@adrian-VirtualBox:~/Documentos/Practica4$ ./pr4
[5 1 7]x[8 1 3]
[ -4.00 41.00 -3.00 ]
[1 7 5]x[1 3 8]
[ 41.00 -3.00 -4.00 ]
```

Magnitud

La magnitud de un vector se obtiene sacando la raíz cuadrada de la suma de los cuadrados de los elementos, es decir, $\sqrt{a^2 + b^2 + c^2}$

```
adrian@adrian-VirtualBox:~/Documentos/Práctica4$ ./pr4
|[8 4 3]|
9.433981
```

En este caso es $\sqrt{8^2 + 4^2 + 3^2} = \sqrt{64 + 16 + 9} = \sqrt{89} \approx 9.43398$

Realiza exactamente las mismas operaciones que en la práctica anterior, también con la posibilidad de poder añadir y utilizar variables:

```
adrian@adrian-VirtualBox:~/Documentos/Práctica4$ ./pr4
vector1=[15 11 17]
vector2=[51 71 11]
vector1+vector2
[ 66.00 82.00 28.00 ]
vector1-vector2
[ -36.00 -60.00 6.00 ]
|vector1|
25.199206
|vector2|
88.107888
vector3=vector1+vector2
vector3
[ 66.00 82.00 28.00 ]
```

Como se puede observar, es posible asignarles un nombre a los vectores, en este caso, $vector1 = [15\ \ 11\ \ 17]$ y $vector2 = [51\ \ 71\ \ 11]$. Después se muestra que es posible realizar operaciones usando el nombre de la variable.

Al sumarlos ambos vectores, da como resultado $[66\ \ 82\ \ 28]$

La magnitud del primer vector es $\sqrt{15^2 + 11^2 + 17^2} = \sqrt{255 + 121 + 289} = \sqrt{665} \approx 25.20$

La magnitud del primer vector es $\sqrt{51^2 + 71^2 + 11^2} = \sqrt{7763} \approx 88.10$

Así como declarar un vector3, cuyo valor es la suma del vector1 y vector2, esto es: $vector3 = [15\ \ 11\ \ 17] + [51\ \ 11\ \ 17] \rightarrow vector3 = [66\ \ 82\ \ 28]$.

De esta manera se puede comprobar el correcto funcionamiento de cada una de las operaciones propuestas.

CONCLUSIÓN

Una vez realizada esta práctica, pude emplear los conocimientos que hemos visto hasta ahora durante este curso, conocimientos sobre YACC y sobre el HOC4, hice uso de la pila para almacenar cada una de las variables y sus resultados, a diferencia de las anteriores prácticas, en donde se hacía uso de arreglos y una lista ligada, el funcionamiento es el mismo pero la manera de guardar las variables y sus resultados es lo que es diferente. Utilice como base los códigos de la práctica 3, además de otros a manera de cambiar lo que sea necesario y agregar los códigos faltantes para el funcionamiento correcto de la práctica.