



INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

Práctica número 3:
Calculadora para vectores
(Con tabla de símbolos)

4 de enero de 2021

Grupo: 3CM7

Nombre del alumno:
Ramos Mesas Edgar Alain

Número de boleta:
2013090243

MATERIA: COMPILADORES

1. Introducción

La tabla de símbolos, también llamada “tabla de nombres” o “tabla de identificadores” tiene dos funciones principales:

- Efectuar chequeos semánticos
- Generación de código

La tabla almacena la información que en cada momento se necesita sobre las variables del programa, información tal como: nombre, tipo, dirección de localización, tamaño, etc. La gestión de la tabla de símbolos es muy importante, ya que consume gran parte del tiempo de compilación. De ahí que su eficiencia sea crítica. Aunque también sirve para guardar información referente a los tipos creados por el usuario, tipos enumerados y, en general, a cualquier identificador creado por el usuario. Respecto a cada una de ellas podemos guardar:

- Almacenamiento del nombre. Se puede hacer con o sin límite. Si lo hacemos con límite, emplearemos una longitud fija para cada variable, lo cual aumenta la velocidad de creación, pero limita la longitud en unos casos, y desperdicia espacio en la mayoría. Otro método es habilitar la memoria que necesitemos en cada caso para guardar el nombre.
- Dirección de memoria en que se guardará. Esta dirección es necesaria, porque las instrucciones que referencian a una variable deben saber dónde encontrar el valor de esa variable en tiempo de ejecución, también cuando se trata de variables globales. En lenguajes que no permiten recursividad, las direcciones se van asignando secuencialmente a medida que se hacen las declaraciones. En lenguajes con estructuras de bloques, la dirección se da con respecto al comienzo del bloque de datos de ese bloque, (función o procedimiento) en concreto.
- El número de dimensiones de una variable array, o el de parámetros de una función o procedimiento. Junto con el tipo de cada uno de ellos es útil para el chequeo semántico. Aunque esta información puede extraerse de la estructura de tipos, para un control más eficiente, se puede indicar explícitamente.

También podemos guardar información de los números de línea en los que se ha usado un identificador, y de la línea en que se declaró. La tabla de símbolos consta de una estructura llamada símbolo. Las operaciones que puede realizar son:

- **Crear** Crea una tabla vacía.
- **Insertar** Parte de una tabla de símbolo y de un nodo, lo que hace es añadir ese nodo a la cabeza de la tabla.
- **Buscar** Busca el nodo que contiene el nombre que le paso por parámetro.
- **Imprimir** Devuelve una lista con los valores que tiene los identificadores de usuario, es decir recorre la tabla de símbolos.

2. Desarrollo

Esta práctica consiste en agregar a nuestro programa de la práctica 1, la calculadora para vectores, lo necesario para guardar vectores en variables (agregar una tabla de símbolos), para esto utilizaremos el ejemplo de hoc3 que el profesor nos brindó. Primeramente, en la sección de declaraciones de YACC se especificó la unión que se utilizara, la cual contiene un apuntador a vector, un apuntador a la tabla de símbolos y un double. Además se especificaron los nuevos tokens VAR e INDEF que usaran la parte sym de la unión.

```

22 %union{
23     Vector *vector;
24     double numero;
25     Symbol *sym;
26 }
27 %token <numero> NUMBER
28 %token <sym> VAR INDEF

```

Posteriormente, cambiaremos la gramática debido a que tiene que ser modificada para cumplir con los nuevos requerimientos pedidos. Dicho lo anterior, a la gramática le declaramos la regla para las variables y la acción para que estas se instalen en la tabla de símbolos (asignación) o bien, para que se verifique si la variable está definida.

```

35 %%
36 list:
37     | list '\n'
38     | list exp '\n' { imprimeVector($2); }
39     | list asgn '\n' { imprimeVector($2); }
40     | list escalar '\n' { printf("%.2f\n", $2); }
41 ;
42 asgn: VAR '=' exp { $$ = $1->u.val = $3; $1->type=VAR; }
43 escalar: exp '*' exp { $$ = productoPunto($1, $3); }
44         | '|' exp '|' { $$ = magnitud($2); }
45         | NUMBER { $$ = $1; }
46         | '-' NUMBER %prec UNARYMINUS { $$ = -$2; }
47 ;
48 exp: vector { $$ = $1; }
49     | VAR { if($1->type == INDEF)execerror("Variable no
    ↪ definida, ", $1->name);
50             $$=$1->u.val;
51             }
52     | asgn { $$ = $1; }
53     | exp '+' exp { $$ = sumaVector ($1, $3); }
54     | exp '-' exp { $$ = restaVector ($1, $3); }
55     | exp '#' exp { $$ = productoCruz($1, $3); }
56     | '(' exp ')' { $$ = $2; }
57     | escalar '*' exp { $$ = productoEscalar($1, $3); }

```

```

58         | exp '*' escalar { $$ = productoEscalar($3, $1); }
59         ;
60
61 vector:      '[' component ']' { $$ = $2; }
62         ;
63 component:   component component { $$ = unirVectores($1, $2); }
64         | NUMBER                { $$ = creaVector(1, $1); }
65         ;
66
67 %%

```

Otro segmento de código que requirió de modificaciones fue el escrito en YYLEX para que se pudiera instalar una variable en la tabla de simbolos si es que esta aún no existe.

```

96 int yylex (){
97     int c;
98
99     while ((c = getchar ()) == ' ' || c == '\t')
100         ;
101     if (c == EOF)
102         return 0;
103     if (c == '.' || isdigit (c)) {
104         ungetc (c, stdin);
105         scanf ("%lf", &(yylval.numero));
106         return NUMBER;
107     }
108     if(isalpha(c)){
109         Symbol *s;
110         char sbuf[200], *p=sbuf;
111         do {
112             *p++=c;
113         } while ((c=getchar())!=EOF && isalnum(c) && c!='X');
114         ungetc(c, stdin);
115         *p='\0';
116         if((s=lookup(sbuf))==(Symbol *)NULL)
117             s=install(sbuf, INDEF, (Vector *)NULL);
118         yylval.sym=s;
119         if(s->type == INDEF){
120             return VAR;
121         } else {
122             //printf("func=(%s) tipo=(%d) \n", s->name, s->type);
123             return s->type;
124         }
125     }
126     if(c == '\n')
127         lineno++;
128     return c;
129 }

```

También es importante mencionar que se requirió modificar el archivo vectores.h para la implementación de la tabla de simbolos, cuya estructura fue especificada en dicho archivo.

```

16 typedef struct Symbol { /* entrada de la tabla de símbolos */
17     char    *name;
18     short   type;   /* VAR, BLTIN, UNDEF */
19     union {
20         Vector *val;      /* si es VAR */
21     } u;
22     struct Symbol *next; /* para ligarse a otro */
23 } Symbol;
24
25 Symbol *install(char *,int, Vector *);
26 Symbol *lookup(char *s);

```

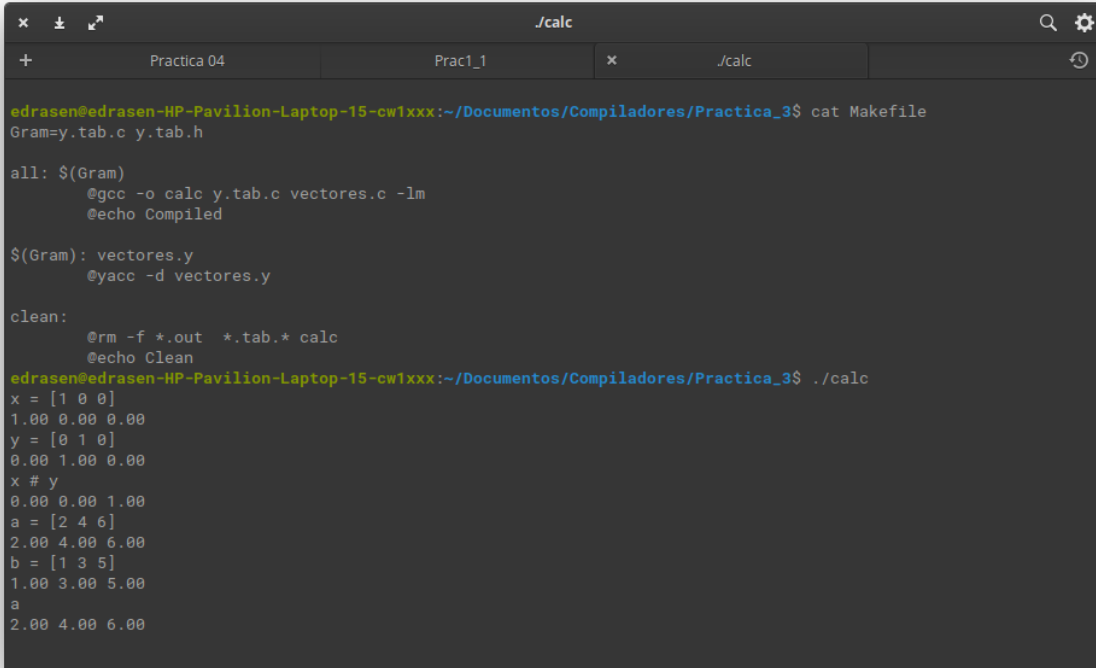
Igualmente se modificó el archivo vectores.c para poder realizar operaciones especificadas en la tabla de simbolos (install y lookup).

```

103 Symbol *lookup(char *s)    /* encontrar s en la tabla de símbolos */
104 {
105     Symbol *sp;
106     for (sp = symlist; sp != (Symbol *)0; sp = sp->next)
107         if (strcmp(sp->name, s) == 0)
108             return sp;
109     return 0;      /* 0 ==> no se encontró */
110 }
111
112 Symbol *install(char *s,int t, Vector *d) /* instalar s en la tabla de símbolos */
113 {
114     Symbol *sp;
115     sp = (Symbol *)malloc(sizeof(Symbol));
116     sp->name = (char *)malloc(strlen(s)+ 1) ; /* +1 para '\0' */
117     strcpy(sp->name, s);
118     sp->type = t;
119     sp->u.val = d;
120     sp->next = symlist; /* poner al frente de la lista */
121     symlist = sp;
122     return sp;
123 }

```

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_3$ cat Makefile
Gram=y.tab.c y.tab.h

all: $(Gram)
    @gcc -o calc y.tab.c vectores.c -lm
    @echo Compiled

$(Gram): vectores.y
    @yacc -d vectores.y

clean:
    @rm -f *.out *.tab.* calc
    @echo Clean

edrasen@edrasen-HP-Pavilion-Laptop-15-cw1xxx:~/Documentos/Compiladores/Practica_3$ ./calc
x = [1 0 0]
1.00 0.00 0.00
y = [0 1 0]
0.00 1.00 0.00
x # y
0.00 0.00 1.00
a = [2 4 6]
2.00 4.00 6.00
b = [1 3 5]
1.00 3.00 5.00
a
2.00 4.00 6.00
```

3. Conclusiones

El agregar una tabla de símbolos nos facilita el poder guardar las variables que el usuario requiera y operar con ellas sin importar el nombre que quiera asignarle a estas, además proporciona funciones que simplifican los cálculos pues ya no es necesario escribir los mismos números una y otra vez. Además se ha podido aprender que para la tabla de símbolos se necesita una unión que contenga un apuntador a una estructura símbolo, un apuntador al tipo de los datos en este caso tipo vector pointer.