Instituto Politécnico Nacional

Escuela Superior de Cómputo

Departamento de Ciencias e Ingeniería

de la Computación

Compiladores

Práctica 4: Máquina virtual de pila HOC4

Opción: Calculadora de vectores

Grupo: 5CM2

Alumno:

Sandoval Hernández Eduardo

Profesor:

Tecla Parra Roberto

Fecha de entrega:

10 de junio de 2022

Introducción

Yet Another Compiler Compiler (YACC) es un generador de analizadores. A partir de un archivo fuente en yacc se puede generar un archivo fuente en C el cual contendrá el analizador sintáctico. Este analizador requerirá de un analizador léxico externo para funcionar, esto debido a que el archivo fuente en C que genera yacc contiene llamadas a la función yylex() la cual debe estar definida y debe ser capaz de devolver el tipo de lexema encontrado. Adicionalmente es necesario incluir una función yyerror() la cual será invocada cuando el analizador sintáctico encuentre un símbolo que no encaje con la gramática [1].

El presente reporte consiste en los resultados obtenidos de la práctica 4 con el tema relacionado a HOC4, para esta práctica se reutilizó el código de la práctica 3 manteniendo la opción de la calculadora de vectores, realizando cambios para que en esta ocasión la calculadora pueda operar con una máquina virtual de pila. Para ello se realizaron cambios en los diferentes archivos de código que componen la práctica, logrando realizar las mismas acciones que en la práctica 3 con la diferencia de que para ellas utiliza la máquina virtual de pila.

Desarrollo

Para implementar la máquina virtual se agregó la estructura Datum para servir de apoyo para la máquina virtual de pila, junto con un apuntador a función Inst para la realización de las funciones, dos arreglos: un arreglo de Datum y otro de Inst, que servirán como las pilas necesarias para la ejecución de las operaciones y las siguientes funciones: initcode, la cual inicializa los apuntadores a los topes de la pila; push, la cual ingresa un Datum a la pila; constpushvector, la cual ingresa un Datum de tipo vector a la pila; constpushdouble, la cual ingresa un Datum de tipo double a la pila; varpush, la cual ingresa un Datum de tipo Symbol a la pila; eval, la cual ingresa a la pila el resultado de la evaluación de una variable; add, la cual realiza la suma de los dos últimos vectores ingresados a la pila; sub, la cual realiza la resta de los dos últimos vectores ingresados a la pila; escalarXvector, la cual realiza la multiplicación de un escalar por un vector; vectorXescalar, la cual realiza la multiplicación de un vector por un escalar; mulPunto, la cual realiza el producto punto de los dos últimos vectores ingresados a la pila; mulCruz, la cual realiza el producto cruz de los dos últimos vectores ingresados a la pila; get_magnitude, la cual obtiene la magnitud del último vector ingresado a la pila; assign, la cual realiza la asignación del valor a una variable e ingresa esta misma a la pila; printVector, la cual imprime el último vector ingresado a la pila; printDouble, la cual imprime el último double ingresado a la pila; code, la cual se encarga de ingresar los apuntadores a Inst a la pila de instrucciones; execute, la cual ejecuta la función que se le envía como parámetro.

En cuanto a la especificación de Yacc, se mantuvo en su mayoría con algunos cambios comenzando por el tipo de dato de la pila de Yacc donde se agregó un apuntador a Inst:

```
%union{
    double doublenumber;
    Vector *vector;
    Symbol *sym;
    Inst *inst;
}
```

Esto debido a que en la pila también se almacenarán instrucciones.

En cuanto a los símbolos terminales y no terminales, estos se mantuvieron casi iguales con la excepción de que se agregaron dos tokens para vectores y números con la finalidad de que se puedan agregar a la pila y un no terminal más para servir como auxiliar para números negativos:

```
%type<doublenumber>      auxNumberNoAction
//  Segundo No Terminal auxiliar para números
%token<sym>     VT
%token<sym>     NO
```

La precedencia de los operadores se mantuvo igual a la práctica anterior.

El cambio más notorio se encuentra en la gramática donde a pesar de que no se cambiaron las producciones y solo se agregó una sencilla, sí se cambiaron todas las acciones gramaticales, quedando de la siguiente manera:

```
list: '\n'
    | exp '\n'          {code2(printVector, STOP); return 1;}
    | asigna '\n'       {code2(pop, STOP); return 1;}
    | NTnumber '\n'     {code2(printDouble, STOP); return 1;}
    ;
```

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

```
exp : vector                {code2(constpushvector, (Inst)$1);}
    | VAR                   {code3(varpush, (Inst)$1, eval);}
    | asigna
    | exp '+' exp           {code(add);}      //  Suma de vectores
    | exp '-' exp           {code(sub);}     //  Resta de vectores
    | exp '*' auxNumber     {code(vectorXescalar);}
//  Producto escalar por la derecha
    | auxNumber '*' exp     {code(escalarXvector);}
//  Producto escalar por la izquierda
    | exp 'x' exp           {code(mulCruz);}
//  Producto cruz de vectores
    ;
```

```
NTnumber: auxNumber         {code2(constpushdouble, (Inst)$1);}
//  Numero negativo o positivo
        | exp '.' exp       {code(mulPunto);}
//  Resultado del producto punto
        | '|' exp '|'       {code(get_magnitude);}
//  Resultado de la magnitud
        ;
```

```
auxNumber   : NUMBER        {$$ = installDouble("", NO, $1);}
//  Número positivo
            | nNUMBER       {$$ = installDouble("", NO, $1);}
//  Número negativo
            ;
```

```
auxNumberNoAction   : NUMBER        //  Número positivo
                    | nNUMBER       //  Número negativo
                    ;
```

```
vector  : '[' auxNumberNoAction auxNumberNoAction ']'   {   //  Crea vector de 2 dimensiones
                                        Vector *v = creaVector(2);
                                        v -> vec[0] = $2;
                                        v -> vec[1] = $3;
                                        $$ = installVector("", VT, v);
                                    }
        | '[' auxNumberNoAction auxNumberNoAction auxNumberNoAction ']' {   //  Crea vector de 3 dimensiones
                                            Vector *v = creaVector(3);
                                            v -> vec[0] = $2;
                                            v -> vec[1] = $3;
                                            v -> vec[2] = $4;
                                            $$ = installVector("", VT, v);
                                        }
        |  '[' auxNumberNoAction auxNumberNoAction auxNumberNoAction auxNumberNoAction ']'  {
    //  Crea vector de 4 dimensiones
                                                    Vector *v = creaVector(4);
                                                    v -> vec[0] = $2;
                                                    v -> vec[1] = $3;
                                                    v -> vec[2] = $4;
                                                    v -> vec[3] = $5;
                                                    $$ = installVector("", VT, v);
                                                }
        |  '[' auxNumberNoAction auxNumberNoAction auxNumberNoAction auxNumberNoAction auxNumberNoAction ']'    {
    //  Crea vector de 5 dimensiones
                                                                Vector *v = creaVector(5);
                                                                v -> vec[0] = $2;
                                                                v -> vec[1] = $3;
                                                                v -> vec[2] = $4;
                                                                v -> vec[3] = $5;
                                                                v -> vec[4] = $6;
                                                                $$ = installVector("", VT, v);
                                                            }
        ;
```

En cuanto al código de soporte, este no tuvo cambios muy grandes con excepción del main donde se modificó la forma en que se llama a yyparce y se utilizan las funciones initcode y execute para esta práctica:

```
void main(int argc, char *argv[]){
    progname=argv[0];
    setjmp(begin);
    signal(SIGFPE,fpecatch);
    for(initcode(); yyparse(); initcode())
        execute(prog);
}
```

En cuanto al resto del código de soporte solo se modificó una línea en el código de yylex para en lugar de llamar a la función install sea a la función installVector:

```c
if(isalpha(c) && c != 'x'){
         // Comprueba que las variables sean letras excepto la 'x'
    Symbol *s;
    char sbuf[200], *p=sbuf;
    do{
        *p++ = c;
    }while((c=getchar()) != EOF && isalnum(c) && c != 'x');
    ungetc(c, stdin);
    *p = '\0';
    if((s=lookup(sbuf)) == (Symbol*)NULL)
   // Comprueba si se encuentra la variable en la tabla de simbolos
        s = installVector(sbuf, INDEF, (Vector*)NULL);
    // Si no se encuentra, la agrega
    yylval.sym=s;
    if(s->type == INDEF)
        return VAR;
    else
        return s->type;
}
```

Para la ejecución y compilación de la práctica se ejecutan los mismos comandos que en la práctica anterior y para esta ocasión se realizarán las mismas pruebas:

```
shosoylalo@LAPTOP-K69NVNG2:/mnt/c/Users/Eduardo_SH/Documents/IPN/ESCOM/Semestre_5/Compiladores$ clear
shosoylalo@LAPTOP-K69NVNG2:/mnt/c/Users/Eduardo_SH/Documents/IPN/ESCOM/Semestre_5/Compiladores$ cd Practica4
shosoylalo@LAPTOP-K69NVNG2:/mnt/c/Users/Eduardo_SH/Documents/IPN/ESCOM/Semestre_5/Compiladores/Practica4$ ls
Practica4_Especificacion.pdf  a.out  vcalc.c  vcalc.h  vcalc.y  y.tab.c  y.tab.h
shosoylalo@LAPTOP-K69NVNG2:/mnt/c/Users/Eduardo_SH/Documents/IPN/ESCOM/Semestre_5/Compiladores/Practica4$ ./a.o
ut
mundo = [1 2 3]
mundo
        Resultado: [ 1.000000 2.000000 3.000000 ]
hola = mundo * 20
hola
        Resultado: [ 20.000000 40.000000 60.000000 ]
escom = hola + mundo
escom
        Resultado: [ 21.000000 42.000000 63.000000 ]
hola . mundo
        Resultado: 280.000000
hola x mundo . escom
        Resultado: 0.000000
|escom|
        Resultado: 78.574805
escom * n8
        Resultado: [ -168.000000 -336.000000 -504.000000 ]
mundo x escom
        Resultado: [ 0.000000 0.000000 0.000000 ]
|hola x mundo|
        Resultado: 0.000000
```

Como se muestra en la evidencia anterior, el programa es capaz de almacenar vectores en variables, y usar dichas variables para realizar operaciones cuyo resultado dependiendo el caso puede ser asignado a una variable, justo como en la práctica anterior solo que el funcionamiento ahora se apoya en una máquina virtual de pila.

Conclusión

Esta práctica fue relativamente sencilla debido a que solo fue necesario adaptar el código para que trabajara con la máquina virtual de pila, como resultado se mantuvo igual en funcionamiento en cuanto al usuario le concierne siendo que en realidad cambió bastante la forma en que se realizan cada una de las operaciones y como muestra de un resultado exitoso no se percibe el cambio a simple vista.

Referencias

[1] F. Simmross Wattenberg (s.f.). “El generador de analizadores sintácticos yacc”. [Internet]. Disponible en https://www.infor.uva.es/~mluisa/talf/docs/labo/L8.pdf