INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Nombre: Cruz López Adrián

Grupo: 3CM15

Asignatura: Compiladores

Profesor: Roberto Tecla Parra

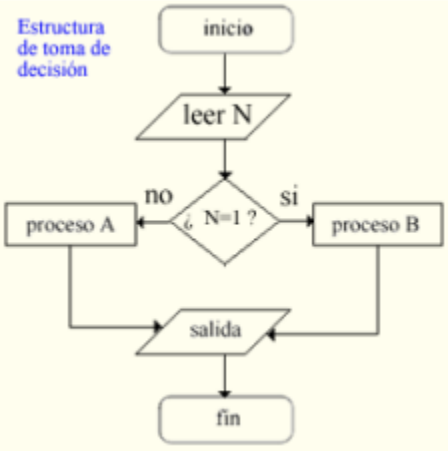
Actividad: Práctica 5 Decisiones y ciclos "*Calculadora para Vectores*"

Fecha: 01/12/2021

INTRODUCCIÓN

Decisiones

Las sentencias de decisión, también llamadas sentencias de control de flujo son estructuras de control la cual evalúa una condición retornando verdadero o falso según sea el caso y selecciona la siguiente instrucción a ejecutar dependiendo la respuesta o resultado.



En algún momento dentro de los algoritmos, es preciso cambiar el flujo de ejecución de las instrucciones, es decir, el orden en que las instrucciones son ejecutadas. Muchas veces se deben tomar decisiones en cuanto a que se debe ejecutar basándonos en una respuesta de la condición (verdadero o falso) .

La ejecución de las instrucciones incluyendo una estructura de control como el condicional funciona de la siguiente manera:

- Las instrucciones comienzan a ejecutarse de forma secuencial y cuando se llega a una estructura condicional, la cual está asociada a una condición, se decide qué camino tomar dependiendo siempre del resultado de la condición siendo esta falsa o verdadera.

- Cuando se termina de ejecutar este bloque de instrucciones se reanuda la ejecución en la instrucción siguiente a la de la condicional.

Sentencia If

La instrucción if es la más utilizada para construir estructuras de control de flujo.

Sintaxis

Primera Forma

```
if (condición){
        Set de instrucciones
}
```

Segunda forma

```
if (condicion){
        Set de instrucciones
} else{
        Set de instrucciones 2
}
```

Ciclos

Los ciclos o bucles permiten ejecutar repetidas veces una instrucción o un bloque de ellas. Deben estar construidos de manera tal que se pueda tener control de la cantidad de repeticiones a realizar, de lo contrario se generaría un ciclo de ejecución infinita que podría desencadenar un desborde de memoria y en consecuencia un fallo de la aplicación, o un bloqueo de la misma porque el flujo de ejecución quedaría estancado en el ciclo, sobrecargando de tareas al procesador de la máquina que ejecuta el programa.

Ciclo While

Permite ejecutar repetidas veces un bloque de instrucciones, su construcción consta únicamente de una condición, pero no hay que olvidar que se debe tener un mecanismo para que el ciclo termine su ejecución o este quedará ejecutándose infinitamente.

Sintaxis

```
while (condición) {

        // Bloque de instrucciones

}
```

DESARROLLO

Se utilizarán los mismos archivos que se utilizaron en la práctica 4, simplemente se modificó un poco la gramática y se agregaron algunas producciones adicionales.

Las producciones agregadas fueron las siguientes:

```
100.    stmt: exp
101.        | PRINT exp
102.        | while cond stmt end
103.
104.        | if cond stmt end
105.
106.        | if cond stmt end ELSE stmt end
107.
108.
109.        | '{' stmtlst '}'
110.        ;
111.    cond: '(' exp ')'
112.        ;
113.    while: WHILE
114.        ;
115.    if: IF
116.
117.        ;
118.    end: /*NADA
119.        ;
120.    stmtlst: /*NADA
121.        | stmtlst '\n'
122.        | stmtlst stmt
123.        ;
```

Se agregaron dichas producciones para abarcar los casos en los cuales se hacen uso de las decisiones y ciclos, que en la práctica únicamente se utilizaran el IF y el ciclo while. La parte de stmt, marca la estructura que deberá llevar cada condición, todo en una misma línea debido a que se correrá en terminal.

Con los pequeños cambios realizados a la gramática:

stmt → exp

stmt → PRINT exp

stmt → while cond stmt end

stmt → if cond stmt end

stmt → if cond stmt end ELSE stmt

stmt → { stmtlst }

cond → (exp)

while → WHILE

if → IF

end → epsilon

stmtlst → épsilon

stmtlst → stmtlst \n

stmtlst → stmtlst stmt

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 modificó quedando de la siguiente manera:

%right '='
%left OR AND
%left GT LT LE EQ NE
%left '+' '-'
%left '*'
%left 'x' '.' '|'
%left UNARYMINUS NOT

La siguiente línea "%left GT GE LT LE EQ NE" es de apoyo para las condiciones de igual, mayor, mayor igual, menor que, menor igual, etc.

En la parte del código de soporte se agrega un switch, el cual sirve para identificar cuando se tiene una condición de dos caracteres, como es el caso del mayor igual, menor igual, diferente, retornando el tipo, según sea el caso:

```
173. switch(c){ //Añadido para la practica 5
174.         case '>': return follow('=', GE, GT);
175.         case '<': return follow('=', LE, LT);
176.         case '=': return follow('=', EQ, '=');
177.         case '!': return follow('=', NE, NOT);
178.         case '|': return follow('|', OR, '|');
179.         case '&': return follow('&', AND, '&');
180.         case '\n': lineno++; return '\n';
181.         default: return c;
182.  }
```

Los archivos de "*vector_cal.c*", tienen exactamente el mismo contenido que en los archivos de la práctica 4, debido a que está basada en ella para realizar esta práctica. A continuación, se muestra la función de magnitud:

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

La magnitud de un vector, por ejemplo [a b c] es la raíz cuadrada de $(a^2 + b^2 + c^2)$

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

Los archivos "*hoc.h*" e "*init.c*" permanecieron casi igual, a excepción de algún cambio menor en las uniones, con el fin de adaptarlo al caso de los vectores. En el archivo "*code.c*" es donde se realizaron los mayores cambios, y donde se encuentran las funciones que manejan el caso de la decisión If y el ciclo while.

```
72. void whilecode() {
73.     Datum d;
74.     Inst  *savepc  = pc;    /*  cuerpo de la iteración  */
75.     execute(savepc+2);     /*   condición   */
76.     d  =  pop();
77.     while   (d.val)   {
78.         execute(*((Inst  **)(savepc)));     /*  cuerpo  */
79.         execute(savepc+2);
80.         d  = pop();
81.     }
82.     pc  =  *((Inst  **)(savepc+1));     /*   siguiente proposición   */
83. }
84.
85. void ifcode(){
86.     Datum d;
87.     Inst  *savepc  = pc;     /* parte then */
88.     execute(savepc+3);       /*  condición   */
89.     d  =  pop();
90.     if (d.val)
91.         execute(*((Inst   **)(savepc)));
92.     else  if   (*((Inst  **)(savepc+1)))   /*  ¿parte else?   */
93.         execute(*(( Inst  **) (savepc+1)));
94.     pc  =  *((Inst  **)(savepc+2)); /*  siguiente proposición   */
95. }
```

De igual manera se hicieron las funciones para los casos de las comparaciones:

```
202.void mayor(){
203.    Datum d1, d2;
204.    d2 = pop();
205.    d1 = pop();
206.    d1.num = (int)( vectorMagnitud(d1.val) > vectorMagnitud(d2.val) );
207.    push(d1);
208.}
209.
210.void menor(){
211.    Datum d1, d2;
212.    d2 = pop();
213.    d1 = pop();
214.    d1.num = (int)( vectorMagnitud(d1.val) < vectorMagnitud(d2.val) );
215.    push(d1);
216.}
```

Para estos casos en específico, el factor para determinar si un vector es mayor que otro, es la magnitud, por lo que se obtiene la parte val de los vectores y se saca su magnitud, y se obtiene un verdadero o falso, dependiendo de los valores, esto sirve para ejecutar o no las partes de código seguidas de la condición, o en su caso, la parte que se encuentre en el else.

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 se declare como tipo datum a d1 y d2.

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/Prácticas/Práctica5$ ./pr5
[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 ]
```

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/Prácticas/Práctica5$ ./pr5
[3 7 9].[5 3 6]
90.00
[4 7 2].[4 1 3]
29.00
```

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

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/Prácticas/Práctica5$ ./pr5
|[8 4 3]|
9.43
```

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 aceptar el uso de la decisión if y el ciclo while.

```
adrian@adrian-VirtualBox:~/Documentos/Prácticas/Práctica5$ ./pr5
|[8 4 3]|
9.43
A=[1 2 3]
B=[4 5 6]
C=[10 11 12]
D=[13 14 15]
E=[6 0 0]
F=[8 0 0]
G=A
A+B
[ 5.00 7.00 9.00 ]
|B|
8.77
if(A==G){print B}
[ 4.00 5.00 6.00 ]
if(A==B){print B}else{print C}
[ 10.00 11.00 12.00 ]
while(A<B){print A A=A+[1 1 1]}
[ 1.00 2.00 3.00 ]
[ 2.00 3.00 4.00 ]
[ 3.00 4.00 5.00 ]
while(B<D){print B B=B+[1 1 1]}
[ 4.00 5.00 6.00 ]
[ 5.00 6.00 7.00 ]
[ 6.00 7.00 8.00 ]
[ 7.00 8.00 9.00 ]
[ 8.00 9.00 10.00 ]
[ 9.00 10.00 11.00 ]
[ 10.00 11.00 12.00 ]
[ 11.00 12.00 13.00 ]
[ 12.00 13.00 14.00 ]
```

CONCLUSIÓN

Una vez realizada esta práctica, pude emplear los conocimientos que hemos visto hasta ahora durante este curso, conocimientos sobre YACC y sobre el HOC5. Se utilizaron los mismos archivos de la práctica anterior, además unos proporcionados por el profesor, el funcionamiento se mantiene igual, simplemente los archivos proporcionados, se tuvieron que modificar un poco para que se adaptaran a la opción elegida.