INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Nombre: Cruz López Adrián

Grupo: 3CM15

Asignatura: Compiladores

Profesor: Roberto Tecla Parra

Actividad: Práctica 7 Funciones y procedimientos "*Calculadora para Vectores*"

Fecha: 14/12/2021

INTRODUCCIÓN

Funciones

Son un elemento muy utilizado en la programación. Empaquetan y "aíslan" del resto del programa una parte de código que realiza alguna tarea específica.

Son por tanto un conjunto de instrucciones que ejecutan una tarea determinada y que se encapsulan en un formato estándar para que sea muy sencillo el manipular y reutilizar.

Consideraciones acerca de las funciones:

- Las funciones NO se pueden anidar, esto significa que una función no se puede declarar dentro de otra función.
- En C++ todas las funciones son externas o globales, es decir pueden ser llamadas desde cualquier punto del programa.

Sintaxis

tipo_de_retorno nombreFunción(listaDePárametros){

    cuerpo de la función

    return expresión

}

Donde:

tipo_de_retorno	Tipo de valor devuelto por la función ola palabra reservada void si la función no devuelve ningún valor
nombreFunción	Indentificador o nombre de la función
listaDePárametros	Lista de declaraciones de los prarametros de la función separaados por comas
Expresión	Valor que devuelve a función

Procedimientos

Son básicamente un conjunto de instrucciones que se ejecutan sin retornar ningún valor, hay quienes dicen que un procedimiento no recibe valores o argumentos, sin embargo, en la definición no hay nada que se lo impida. En el contexto de C++ un procedimiento es básicamente una función void que no nos obliga a utilizar una sentencia return.

Los procedimientos son similares a las funciones, aunque más resumidos. Debido a que los procedimientos no retornan valores, no hacen uso de la sentencia return para devolver valores y no tienen tipo específico, solo void.

Ejemplos:

```
void procedimiento(int n, string nombre){
        if(n == 0){
                cout << "hola" << nombre;
                return;
        }
        cout << "adios" << nombre;
}
```

De este ejemplo se observa ver que ya no se utiliza un tipo, sino que se pone void, indicando que no retorna valores, también se puede observar que un procedimiento también puede recibir parámetros o argumentos.

Importante: Los procedimientos también pueden hacer uso de la sentencia return, pero no con un valor. En los procedimientos el return sólo se utiliza para finalizar allí la ejecución de la función.

DESARROLLO

Se hará uso de los mismos archivos que fueron empleados en la práctica 6, solamente para esta práctica se realizaron algunas modificaciones a los archivos del código, para aceptar el uso de funciones y procedimientos. Se agregaron producciones a la gramática y se modificaron algunas producciones existentes, algunos de los cambios en la gramática son:

```
stmt: exp
        | RETURN
        | RETURN exp
        | PRINT prlist
        | while cond stmt end

        | if cond stmt end


        | if cond stmt end ELSE stmt end



        | '{' stmtlist '}'
        ;
        .
        .
        .
       ...
    prlist: exp

           | STRING
        | prlist ',' exp
        | prlist ',' STRING
        ;
    defn: FUNC procname

        |PROC procname

        ;
```

La parte de **stmt**, marca la estructura que deberá llevar cada condición, todo en una misma línea debido a que se correrá en terminal, un ejemplo de cómo deben introducirse las instrucciones es:

If(condición){ print VAR } else{ print VAR2 }

while(condición) { print VAR }

for(inicialización; condición; incremento) { print VAR }

Además, se le agregaron las producciones de return, para los casos en que la función declarada deba retornar algo, y la producción de print, para los casos donde las funciones o procedimientos sean empleadas para imprimir en pantalla alguna expresión con los vectores.

La producción **prlist** sirve de apoyo a stmt, de modo que muestra la forma de imprimir o retornar un valor según sea el caso.

Por último, se tiene la producción **defn** que muestra cómo se declara la función o procedimiento, en este caso puede ser con "*func*" o con "*proc*", seguido del nombre. Ejemplo de la sintaxis:

proc nombreProc () { print vect1+vect2 }

func nombreFunc () { return vect1+vect2 }

De igual forma se puede hacer uso de los ciclos dentro de dichas funciones o procedimientos.

Dichos cambios en las producciones hacen que queden de la siguiente manera:

stmt → exp
stmt → RETURN
stmt → RETURN exp
stmt → PRINT exp
stmt → while cond stmt end
stmt → if cond stmt end
stmt → if cond stmt end ELSE stmt
stmt → { stmtlst }
prlist → exp
prlist → STRING
prlist → prlist , exp
prlist → prlist , STRING
defn → FUNC procname
defn → PROC procname

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 token, debido a que tienen longitud de 1.

La jerarquía manejada se mantuvo igual que en prácticas anteriores quedando de la siguiente manera:

```
%left '+' '-'
%left '*'
%left '#' '.'
%left  OR AND
%left GT LT LE EQ NE
%left 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. y se cambió el símbolo del producto cruz por “#”, para evitar problemas con la “x” en el nombre de las variables.

La parte de código de soporte sigue manteniendo el switch, que ayuda a identificar cuando se tiene una condición de dos caracteres, como es el caso del mayor igual, menor igual, diferente, retornándonos el tipo, esto sigue siendo de mucha importancia y será de las partes que posiblemente ya no cambie, debido a que, con la agregación de los ciclos, se necesita identificar las condiciones de dos caracteres.

```
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 6, debido a que se basó en ella para realizar esta práctica. Abajo se muestra una de las funciones, en este caso, la multiplicación de vectores o producto cruz:

```
84. Vector *multiplicaVector(Vector *a, Vector *b){
85.     Vector *c;
86.     int i;
87.     c = creaVector(1);
88.     c->vec[0] = a->vec[0] * b->vec[0] + a->vec[1] * b->vec[1] + a-
    >vec[2] * b->vec[2];
89.     return c;
90. }
```

El producto punto es un número y se obtiene mediante la suma de sus componentes en la misma posición, por ejemplo, si tenemos un vector (a b c) y otro (d e f), su producto punto o multiplicación sería: a*d + b*e + c*f

Es una función que retorna un vector, recibe dos apuntadores a vector. Dentro de la función se crea el vector auxiliar de dimensión 1 y en el que se guarda el resultado de la multiplicación de sus componentes.

Dentro de “*init.c*“ *se* tiene una lista de palabras clave, las cuales ayudan en los ciclos o en este caso, que se agregaron las palabras de func, proc y return, las que permiten abarcar los casos de declaraciones de funciones y procesos, además del retorno de valores dentro de estos:

```
18. static struct {
19.     char   *name;   /* Palabras clave */
20.     int    kval;
21. } keywords[] = {
22.     "proc" ,    PROC,
23.     "func" ,    FUNC,
24.     "return",   RETURN,
25.     "if",       IF,
26.     "else" ,    ELSE,
27.     "while",    WHILE,
28.     "print",    PRINT,
29.     "read",     READ,
30.     0,      0,
31. };
```

Dentro de "*code.c*" se mantienen las funciones que sirven para determinar las condiciones de mayor, menor, igual, diferente, etc. Además de agregar las de producto cruz y producto punto, como se muestra a continuación:

```
116.void cruz()
117.{
118.    Datum d1, d2;
119.    d2 = pop();
120.    d1 = pop();
121.    d1.val = productoCruz(d1.val, d2.val);
122.    push(d1);
123.}
```

...

```
218.void eq( ) {
219.    Datum d1,  d2;
220.    d2  = pop();
221.    d1  = pop();
222.    d1.numero = (double)(magnitud(d1.val) == magnitud(d2.val));
223.    push(d1);
224.}
```

Para el caso de la función de **eq**, que sirve para determinar si la condición que se propone es igual. Se observa que 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 determina 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.

En el caso de la función **cruz**, lo único que se hace es sacar de la pila a d1 y d2 y posteriormente obtener su parte val y realizarle el producto cruz que está especificado en el archivo "*vector_cal.c*".

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 declare tipo datum a d1 y d2.

PRUEBAS DE FUNCIONAMIENTO

Las pruebas siguen siendo las mismas, que en prácticas anteriores.

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

Como se mostró anteriormente, las pruebas siguen siendo las mismas, ya que únicamente se cambió la gramática y el código para aceptar el uso de funciones y procedimientos.

```
adrian@adrian-VirtualBox:~/Documentos/Prácticas/Práctica7$ ./pr7
A=[1 1 1]
[ 1.00 1.00 1.00 ]
B=[6 6 6]
[ 6.00 6.00 6.00 ]
C=[12 12 12]
[ 12.00 12.00 12.00 ]
proc sumyres(){print $1+$2 print $1-$2}
sumyres(A,B)
[ 7.00 7.00 7.00 ]
[ -5.00 -5.00 -5.00 ]
proc resta(){print $1-$2}
resta(B,C)
[ -6.00 -6.00 -6.00 ]
proc oper(){print $1+$2 print $1-$2 print mag($1)}
oper(B,A)
[ 7.00 7.00 7.00 ]
[ 5.00 5.00 5.00 ]
[ 10.39 ]
```

```
func ciclos(){while($1<$2){$1=$1+[1 1 1] print $1}}
ciclos(A,C)
[ 1.00 1.00 1.00 ]
[ 2.00 2.00 2.00 ]
[ 1.00 1.00 1.00 ]
[ 3.00 3.00 3.00 ]
[ 1.00 1.00 1.00 ]
[ 4.00 4.00 4.00 ]
[ 1.00 1.00 1.00 ]
[ 5.00 5.00 5.00 ]
[ 1.00 1.00 1.00 ]
[ 6.00 6.00 6.00 ]
[ 1.00 1.00 1.00 ]
[ 7.00 7.00 7.00 ]
[ 1.00 1.00 1.00 ]
[ 8.00 8.00 8.00 ]
[ 1.00 1.00 1.00 ]
[ 9.00 9.00 9.00 ]
[ 1.00 1.00 1.00 ]
[ 10.00 10.00 10.00 ]
[ 1.00 1.00 1.00 ]
[ 11.00 11.00 11.00 ]
[ 1.00 1.00 1.00 ]
[ 12.00 12.00 12.00 ]
func suma(){return $1+$2}
suma(C,B)
[ 18.00 18.00 18.00 ]
```

CONCLUSIÓN

Una vez realizada esta práctica, pude emplear los conocimientos adquiridos durante este curso, conocimientos sobre YACC y sobre el HOC6. Al igual que en las practicas anteriores, se realizaron modificaciones en la gramática, esto para abarcar los casos en los que se usaran funciones, procedimientos y la combinación de estos con lo que ya se tenía en prácticas pasadas, al igual que los archivos que nos fueron proporcionados, se tuvo que modificar en cierta parte para adaptarlo a la opción elegida y agregar unas cuantas funciones para el correcto funcionamiento.