UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXÁCTAS E INGENIERÍAS

DIVISIÓN DE ELECTRÓNICA Y COMPUTACIÓN

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

INGENIERÍA EN COMPUTACIÓN

TRADUCTORES DE LENGUAJE II

REPORTE ANALIZADOR SEMÁNTICO

NOVOA ORTEGA DIEGO

PONCE RAMOS DALIA

ZÁRATE MACÍAS ALEJANDRO

SECCIÓN D06 – CALENDARIO 2021A

GUADALAJARA JALISCO, 28-MAYO-2021

CONTENIDO

INTRODUCCIÓN

- **ANALIZADOR SEMÁNTICO**

El Analizador semántico ejecuta sus funciones en conjunto al analizador sintáctico, ambos justo antes de la generación de código intermedio, es decir, es una etapa clave dentro del proceso de la compilación de código. Para esta etapa del proyecto, tenemos que entender el significado de “semántica” y a lo que esta se refiere.

La semántica es un conjunto de reglas que especifican el significado de cualquier sentencia sintácticamente correcta y escrita en un determinado lenguaje. Corresponde al significado asociado a las estructuras formales (sintaxis) del lenguaje, no se pueden describir todos los elementos sintácticos del lenguaje por lo que se hace presente algún análisis adicional. Así, se denomina tradicionalmente “análisis semántico” a todo aquello que forma parte del front-end más allá de lo que la gramática utilizada nos permite.

La fase de análisis semántico revisa el programa fuente para tratar de encontrar errores semánticos y reúne la información sobre los tipos para la fase posterior de generación de código. En ella se utiliza la estructura jerárquica determinada por la fase de análisis sintáctico para identificar los operadores y operandos de expresiones y proposiciones.

Un componente importante del análisis semántico es la verificación de tipos. Aquí, el compilador verifica si cada operador tiene operandos permitidos por la especificación del lenguaje fuente.

OBJETIVO GENERAL

El Analizador Semántico es la etapa III, la cual da continuidad al Analizador Léxico que se generó en la etapa I, junto al Analizador Sintáctico de la etapa II.

El objetivo general consta de un Analizador Semántico, por lo que en esta etapa se deben de reutilizar los árboles que se generaron en el Analizador Sintáctico, ya que en esta fase del proyecto es necesario comprobar la consistencia semántica del programa fuente con la definición del lenguaje.

Como se mencionó en la introducción, esta parte del programa consta de la semántica y su conjunto de reglas al momento de especificar una sentencia, por lo tanto, el Analizador Semántico debe de ser capaz de recopilar la información sobre los tipos de datos y almacenarla para posteriormente poder usarla durante la generación de código, ya que una parte importante del Analizador Semántico es la verificación de las partes que conforman el código.

OBJETIVO PARTICULAR

Para esta actividad se pidió hacer la elaboración de la tercera fase, el Analizador Semántico, esto con el fin de reconocer, identificar y verificar las instrucciones con las que trabajará nuestro lenguaje en base a los tokens y herramientas que decidimos utilizar en la primer y segunda fase del Compilador. Para su elaboración se utilizará el lenguaje de programación C++ con el "framework" de Qt.

El programa seguirá funcionando mediante archivos de texto ya que al ejecutar el programa se solicitará el nombre del archivo que contenga el código a analizar. Se tendrán las siguientes instrucciones a reconocer:

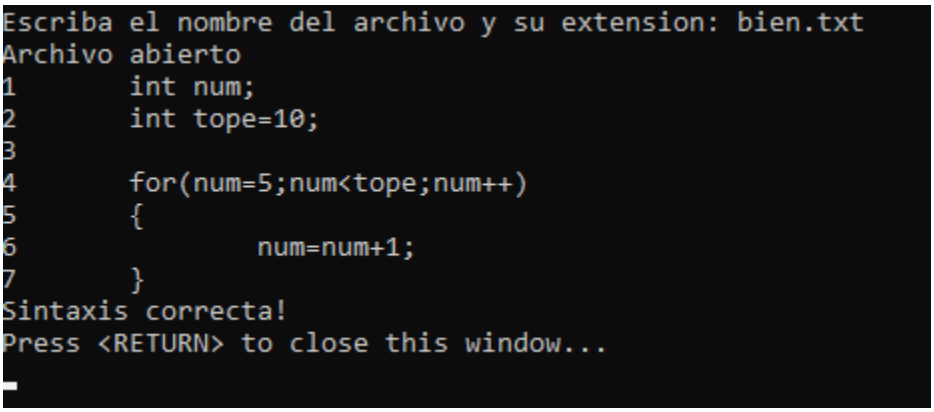
- Cout
- Cin
- Asignaciones
- Declaraciones
- If
- Else
- For
- While
- Do-While

El Analizador Semántico reconocerá las siguientes cuestiones y situaciones si lo requiere:

- Identificar cada tipo de instrucción y sus componentes.
- Si una variable está queriendo tomar un valor que no corresponde a su tipo de dato, este mostrará un error.
- No se pueden asignar variables de diferentes tipos de dato.
- No se pueden usar variables que no hayan sido declaradas previamente.

DESARROLLO

- **EJEMPLO DE UN CÓDIGO CORRECTO**



- **EJEMPLO DE SINTAXIS CORRECTA CON UN WARNING**

```
Escriba el nombre del archivo y su extension: bienwar.txt
Archivo abierto
1       int numero;
2       string cadena="Me imprimo";
3       float cantidad;
4       string hola="Hola Mundo!";
5
6       cout<<"ingrese una cantidad: ";
7       cin>>cantidad;
8
9       do{
10              cout<<cadena;
11              numero=numero+1;
12      }while(numero!=cantidad);
Sintaxis correcta!
WARNING: La variable "hola" no ha sido utilizada.
Press <RETURN> to close this window...
```

- **EJEMPLO DE SINTAXIS INCORRECTA SINTACTICAMENTE**

```
Escriba el nombre del archivo y su extension: mal.txt
Archivo abierto
1       int numero=1;
2       string cadena="hola";
3       float flotante=2.5;
4       int iguala=2;
5
6       iguala=cadena;
7
8       cout<<"imprimiendo su cadena...";
9       cout<<holamundo;
10
11      if(numero<cadena)
12      {
13              numero=numero+cadena;
14              cout<<numero;
15      }
16      else
17      {
18              cout<<"numero es mayor o diferente a cadena";
19      }
20      if(flotante>cadena)
21      {
22              cout<<"float es mayor a cadena";
23      }
24      else
25      {
26              cin>>holamundo;
27      }
"iguala" no es compatible con "cadena". Error en la linea 6
La variable "holamundo" no existe. Error en la linea 9
"numero" no es compatible con "cadena". Error en la linea 11
Los datos son incompatiblesError en la linea 13
"flotante" no es compatible con "cadena". Error en la linea 20
La variable "holamundo" no existe. Error en la linea 26
Press <RETURN> to close this window...
```

- **EJEMPLO DE ERROR CON VARIABLE INEXISTENTE**

```
Escriba el nombre del archivo y su extension: malcomp.txt
Archivo abierto
1       string hola="Hola mundo";
2       int num=2;
3
4       if(hola==num)
5       {
6               cout<<"El numero y cadena son iguales";
7       }
"hola" no es compatible con "num". Error en la linea 4
Press <RETURN> to close this window...
```

CONCLUSIÓN

El análisis semántico, a diferencia de otras fases, no se realiza claramente diferenciado del resto de las tareas que lleva a cabo el compilador, más bien podría decirse que el análisis semántico completa las dos fases anteriores de análisis léxico y sintáctico incorporando ciertas comprobaciones que no pueden asimilarse al reconocimiento de una cadena dentro de un lenguaje.

Una de las principales funciones del análisis semántico es analizar el código y detectar errores de tipo semánticos que existan. Esta parte del proyecto resulta bastante útil ya que en la gran mayoría de los códigos que se generan suelen presentar errores de semántica, los cuales pueden ser:

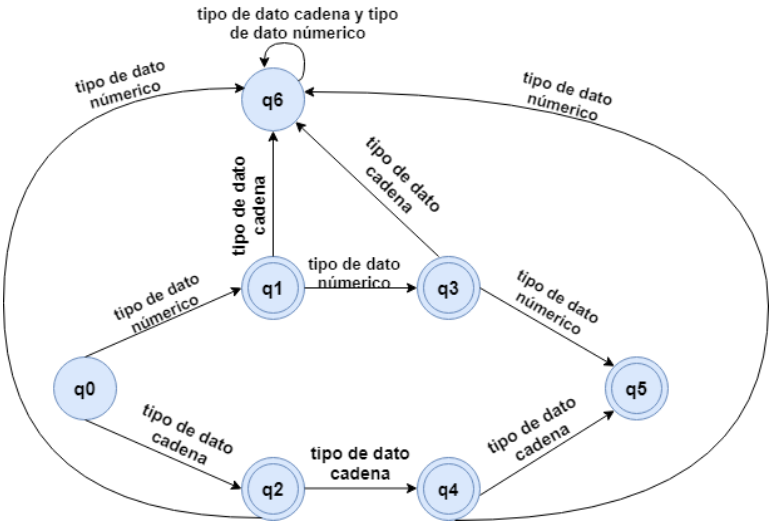
- Que no coincidan los tipos de datos.
- Variables no declaradas.
- Identificador usado de manera incorrecta.
- Declaración de variables múltiples en un ámbito.
- Acceder a una variable fuera de alcance.
- Parámetro formal y real no coincide.

Por lo tanto, el analizador semántico ayuda a prevenir el uso de estos errores.

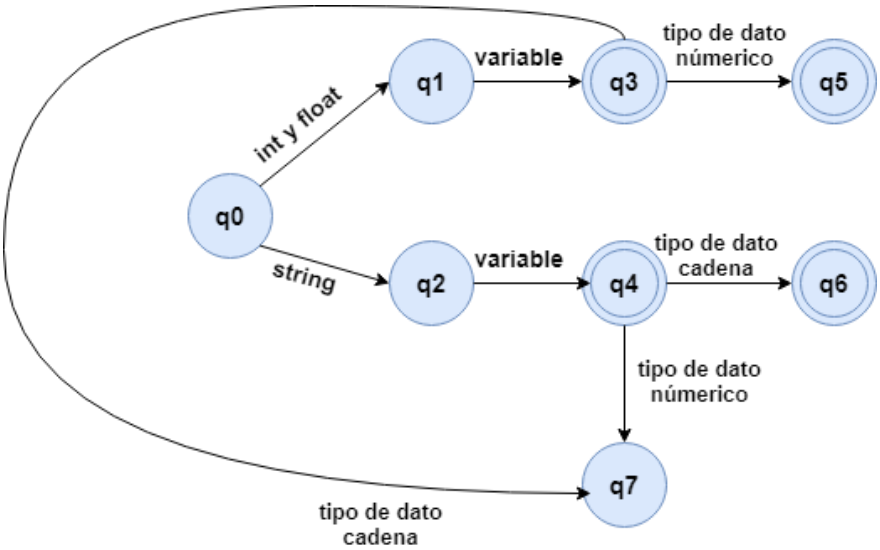
BIBLIOGRAFÍAS

- ***2.4. Análisis semántico - Teoría de Lenguajes y Compiladores. (s. f.). TEORÍA DE LENGUAJES Y COMPILADORES. Recuperado 24 de mayo de 2021, de https://sites.google.com/site/teoriadelenguajesycompiladores/procesadores-de-lenguaje/analisis-semantico#:%7E:text=2.4.-,An%C3%A1lisis%20sem%C3%A1ntico,con%20la%20definici%C3%B3n%20del%20lenguaje.***
- ***Análisis Semántico. (2014). ANÁLISIS SEMÁNTICO. http://zonainformaticaavanzada.blogspot.com/2016/11/analisis-semantico.html***

APÉNDICES



Entradas	Tipo de dato numérico	Tipo de dato cadena
Q0	Q1	Q2
Q1	Q3	Q6
Q2	Q6	Q4
Q3	Q5	Q6
Q4	Q6	Q5
Q5	-	-
Q6	Q6	Q6



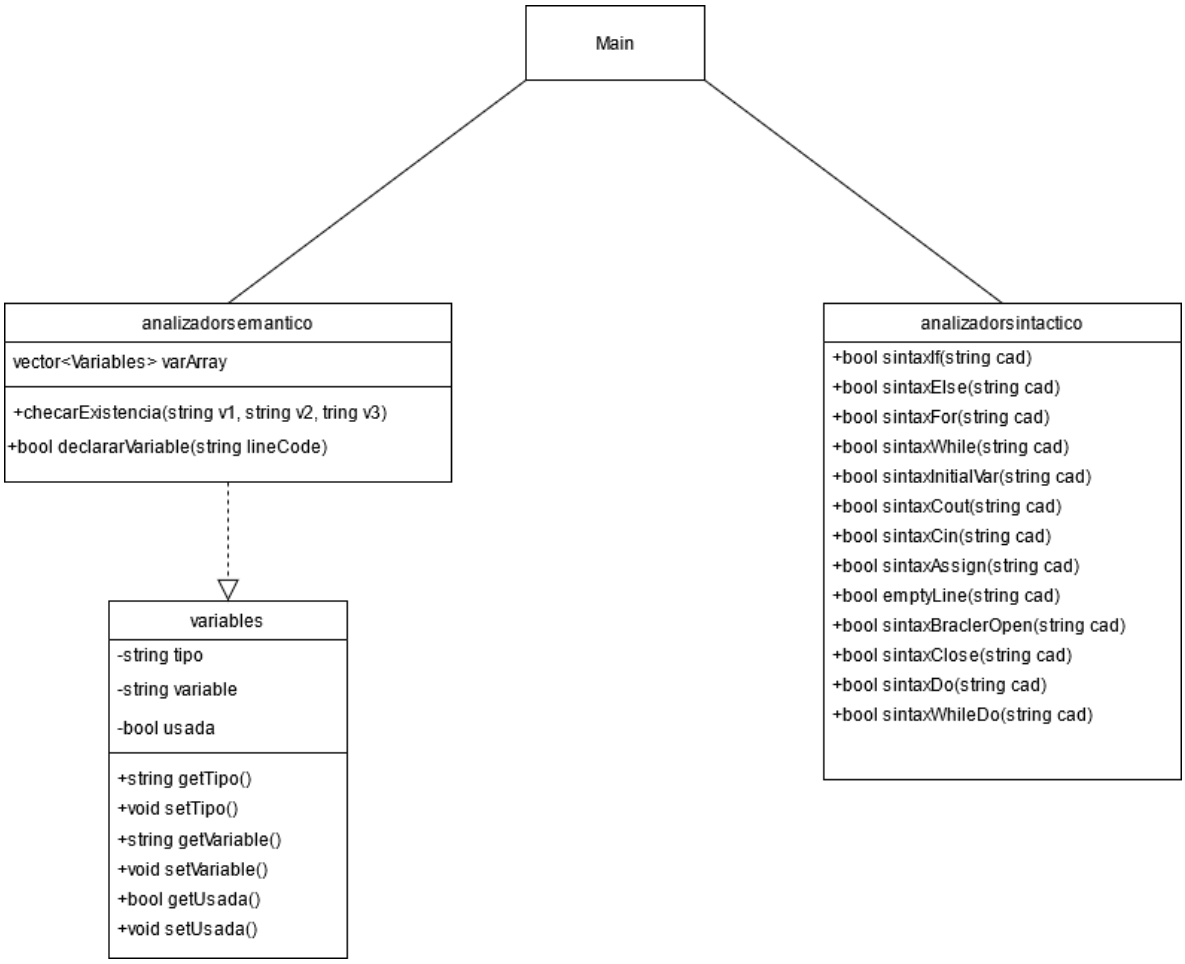
Entradas	Tipo de dato numérico	Tipo de dato cadena	Int	Float	String	Variable
Q0	-	-	Q1	Q1	Q2	-
Q1	-	-	-	-	-	Q3
Q2	-	-	-	-	-	Q4
Q3	Q5	Q7	-	-	-	-
Q4	Q7	Q6	-	-	-	-
Q5	-	-	-	-	-	-
Q6	-	-	-	-	-	-
Q7	-	-	-	-	-	-

ACRÓNIMOS

- **QT:** Qt es un “framework” multiplataforma orientado a objetos ampliamente usado para desarrollar programas que utilicen interfaz gráfica de usuario, así como también diferentes tipos de herramientas para la línea de comandos y consolas para servidores que no necesitan una interfaz gráfica de usuario.
- **C++:** Lenguaje de Programación
- **COUT:** Es el flujo de salida estándar que por lo general es la pantalla.
- **CIN:** Es el flujo de entrada estándar que normalmente es el teclado.
- **IDENTIFICADORES:** Un identificador es un conjunto de caracteres alfanuméricos de cualquier longitud que sirve para identificar las entidades del programa (clases, funciones, variables, tipos compuestos) Los identificadores pueden ser combinaciones de letras y números.
- **ASIGNACIONES:** Esta operación es una manera diferente que se utiliza para que una variable reciba un valor de forma directa sin intervención del usuario, o como resultado de la evaluación de una expresión.
- **DECLARACIONES:** Una declaración es una construcción de lenguaje que especifica las propiedades de un identificador: declara lo que una palabra (identificador) "significa".
- **IF:** Se utiliza para evaluar una expresión condicional: si se cumple la condición (es verdadera), ejecutará un bloque de código. Si es falsa, es posible ejecutar otras sentencias.
- **ELSE:** Orden que se ejecuta si la condición es falsa o no se cumple.
- **FOR:** Se usa cuando queremos repetir un conjunto de instrucciones un número finito de veces. las instrucciones se repiten el número de veces que le decimos, normalmente le ponemos un número (o el valor de una variable o una constante).
- **WHILE:** Se va repitiendo el código en base a una condición, es decir, mientras esa condición sea verdadera.
- **DO-WHILE:** El bucle do, bucle hacer, hacer-mientras o también llamado ciclo do-while, es una estructura de control de la mayoría de los lenguajes de programación estructurados cuyo propósito es ejecutar un bloque de código y repetir la ejecución mientras se cumpla cierta condición expresada en la cláusula while.

DIAGRAMA

- **DIAGRAMA DE CLASES**



REQUISITOS FUNCIONALES

- Pedir el nombre del archivo donde se encuentra el código.
- Imprimir el código.
- Almacenar las nuevas variables en un arreglo.
- Revise una existencia al declarar variables.
- Marcar correctamente los errores.
- Revisar operaciones entre tipos de datos compatibles.
- Mandar alertas si se declaran variables y no se usan en el código.

REQUISITOS NO FUNCIONALES

- El programa deberá estar hecho en consola.
- Deberá se rápido y eficiente.
- Los mensajes de error deberán ser claros.

COMPLEJIDAD CICLOMÁTICA

Para el cálculo de la complejidad ciclomática se requieren los siguientes datos:

- **M** = Complejidad ciclomática.
- **E** = Número de aristas del grafo. Una arista conecta dos vértices si una
- sentencia puede ser ejecutada inmediatamente después de la primera.
- **N** = Número de nodos del grafo correspondientes a sentencias del programa.

Fórmula:

M = E − N + 2

Sustituimos nuestros valores en la fórmula:

M = 48-33+2

M = 17

La complejidad ciclomática es igual a 17

COCOMO

Para realizar un cálculo básico del Cocomo, se requieren los siguientes datos:

- **Variable "A" = 2.4**
- **Variable "B" = 1.05**
- **Variable "C" = 2.5**
- **Variable "D" = 0.38**
- **KLOC = 1.045**

Los datos de las variables son constantes ya definidas mientras que KLOC es el número de líneas en código en miles.

Podemos obtener el resultado del Cocomo con las siguientes formulas: Esfuerzo persona/mes = a * KLOC ^{b}.

- **Esfuerzo = (2.4) (1.045) $^{(1.05)}$**
- ***Esfuerzo = 2.51352***

- **Duración en meses = c * Esfuerzo ^{d}**
- **Duración = (2.5) (2.51352) $^{(0.38)}$**
- ***Duración = 3.54852***

- **Personal = Esfuerzo / Duración**
- **Personal = 2.51352/3.54852**
- ***Personal = 07.832***

CAJA NEGRA

Entrada	Salida

int n;
int m=5;
n=m+4;
	Código Correcto

string cad; cad=”hola”;	Código Correcto
float b=5.5; cout<<b;	float b=5.5; cout<<b;

float b=5.5;
string c;
if(c!=b)
	Error, c y b no son compatibles

int a; int b=6; string c=”5.5”; a=b+c	Error, los tipos de datos son incompatibles
int a; a=a+b;	Error, b no ha sido declarada previamente

CAJA BLANCA

Entrada	Proceso	Salida

int n;
int m=5;
n=m+4;
	1-3-5-6-8-9-11-13-14-16-17-18-24-26-28-31	Código Correcto

string cad; cad=”hola”;	1-3-5-6-8-9-11-12-16-24-26-27-29-33	Código Correcto
float b=5.5; cout<<b;	1-3-5-6-8-24-25-33	float b=5.5; cout<<b;

float b=5.5;
string c;
if(c!=b)
	1-3-5-6-8-9-11-13-14-16-24-26-27-30-33	Error, c y b no son compatibles

int a; int b=6; string c=”5.5”; a=b+c	1-3-5-6-8-9-11-13-14-16-17-19-21-22-24-23-28-32-33	Error, los tipos de datos son incompatibles
int a; a=a+b;	1-3-5-6-8-9-11-13-14-16-17-19-21-23-33	Error, b no ha sido declarada previamente

GRAFOS DEL ANALIZADOR LEXICO

- **FUNCIÓN DE CHEQUEO DE VARIABLES**

```
bool AnalizadorSemantico::declararVariable(const string lineCode)
{
    string t="", v="", a="";
    bool tF=true, vF=false, aF=false;
    for(size_t i(0);i<lineCode.size();++i)
    {
        if(tF)
        {
            if(lineCode[i]=='\t'){}
            else
                if(lineCode[i]!=' ')
                {
                    t+=lineCode[i];
                }
                else
                {
                    tF=false;
                    vF=true;
                }
        }
        else
            if(vF)
            {
                if(lineCode[i]!=';' && lineCode[i]!='=')
                {
                    v+=lineCode[i];
                }
                else
                {
                    if(lineCode[i]=='=')
                        aF=true;
                    vF=false;
                }
            }
            else
                if(aF)
                {
                    if(lineCode[i]!=';')
                    {
                        a+=lineCode[i];
                    }
                    else
                    {
                        aF=false;
                    }
                }
                else
                    break;
    }
```

```
    if(a.size()>0)
    {
        regex varReg(VarSintax);
        regex numReg(NumSintax);
        if(a[0]=='"')
        {
            if(t!="string")
            {
                cout<<"La variable \""<<v<<"\" no puede recibir ese valor. ";
                return false;
            }
        }
        else
            if(regex_match(a,varReg))
            {
                size_t j;
                for(j=0;j<varArray.size();++j)
                {
                    if(a==varArray.at(j).getVariable())
                    {
                        varArray[j].setUsada(true);
                        if( (varArray.at(j).getTipo()=="string" && (t=="int" || t=="float")) || (t=="string" && (varArray.at(j).getTipo()=="int" || varArray.at(j).getTipo()=="float")) )
                        {
                            cout<<"La variable \""<<v<<"\" no es compatible con \""<<a<<"\". ";
                            return false;
                        }
                        break;
                    }
                }
                if(j==varArray.size())
                {
                    cout<<"La variable \""<<a<<"\" no existe. ";
                    return false;
                }
            }
            else
                if(regex_match(a,numReg) && t=="string")
                {
                    cout<<"La variable \""<<v<<"\" no puede recibir ese valor. ";
                    return false;
                }
    }
    for(size_t i(0);i<varArray.size();++i)
    {
        if(v==varArray.at(i).getVariable())
        {
            cout<<"La variable \""<<v<<"\" ya ha sido declarada previamente. ";
            return false;
        }
    }
```

- **FUNCIÓN PARA EL USO CORRECTO DE VARIABLES**

```
bool AnalizadorSemantico::checarExistencia(const string v1, const string v2, const string v3)
{
    string wi1="", wi2="", wi3="";
    bool b2=false, b3=false;

    //Para una variable
    regex numReg(NumSintax);
    if(v1[0]=='"')
    {
        wi1="s";
    }
    else
        if(regex_match(v1,numReg))
        {
            wi1="n";
        }
        else
        {
            size_t j;
            for(j=0;j<varArray.size();++j)
            {
                if(v1==varArray.at(j).getVariable())
                {
                    varArray[j].setUsada(true);
                    if(varArray.at(j).getTipo()=="string")
                        wi1="s";
                    else
                        wi1="n";
                    break;
                }
            }
            if(j==varArray.size())
            {
                cout<<"La variable \""<<v1<<"\" no existe. ";
                return false;
            }
        }
```

```
    //Para dos variables
    if(v2.size()>0)
    {
        if(v2[0]=='"')
        {
            wi2="s";
        }
        else
            if(regex_match(v2,numReg))
            {
                wi2="n";
            }
            else
            {
                size_t j;
                for(j=0;j<varArray.size();++j)
                {
                    if(v2==varArray.at(j).getVariable())
                    {
                        varArray[j].setUsada(true);
                        if(varArray.at(j).getTipo()=="string")
                            wi2="s";
                        else
                            wi2="n";
                        break;
                    }
                }
                if(j==varArray.size())
                {
                    cout<<"La variable \""<<v2<<"\" no existe. ";
                    return false;
                }
            }

        b2=true;
```

```
    //Para dos variables
    if(v2.size()>0)
    {
        if(v2[0]=='"')
        {
            wi2="s";
        }
        else
            if(regex_match(v2,numReg))
            {
                wi2="n";
            }
            else
            {
                size_t j;
                for(j=0;j<varArray.size();++j)
                {
                    if(v2==varArray.at(j).getVariable())
                    {
                        varArray[j].setUsada(true);
                        if(varArray.at(j).getTipo()=="string")
                            wi2="s";
                        else
                            wi2="n";
                        break;
                    }
                }
                if(j==varArray.size())
                {
                    cout<<"La variable \""<<v2<<"\" no existe. ";
                    return false;
                }
            }

        b2=true;
```

```
    if(!b2 && !b3)
    {
        return true;
    }
    else if(b2 && !b3)
    {
        if(wi1==wi2)
            return true;
        else
        {
            cout<<"\""<<v1<<"\" no es compatible con \""<<v2<<"\". ";
            return false;
        }
    }
    else
    {
        if(wi1==wi2&&wi1==wi3)
            return true;
        else
        {
            cout<<"Los datos son incompatibles";
            return false;
        }
    }

}
```

- **FUNCIÓN PARA CHECAR LAS VARIABLES NO USADAS**

```
void AnalizadorSemantico::buscarNoUsadas()
{
    for(size_t i(0); i<varArray.size();++i)
    {
        if(!varArray.at(i).getUsada())
        {
            cout<<"WARNING: La variable \""<<varArray.at(i).getVariable()<<"\" no ha sido utilizada."<<endl;
        }
    }
}
```