Mi Primer Proyecto Utilizando GOLD Parser (Windows)

Se desarrollará un intérprete que recibe como entrada varias expresiones aritméticas y presenta como salida el resultado de dichas expresiones.

Tecnologías

- **GOLD Parser Builder:** Generador de analizadores léxicos y sintácticos diseñado para funcionar en múltiples lenguajes de programación.
- **Visual Studio 2017:** Entorno de desarrollo integrado para programar Visual Basic y C#
- **Windows 8.1:** Sistema operativo (Compatible con Windows 10)
- **.NET DLL:** DLL que permite la interpretación de las tablas de análisis.

El proyecto completo puede descargarse del siguiente enlace:

Mi Primer Proyecto Utilizando GOLD Parser (Windows).

GOLD Parser

Es un generador de analizadores léxicos y sintácticos multiplataforma que soporta lenguajes tales como C#, COBOL, DELPHI, Visual Basic, Java entre otros. Este programa realiza de manera conjunta el análisis léxico y sintáctico, por lo que no tenemos la necesidad de recurrir a ningún programa externo.

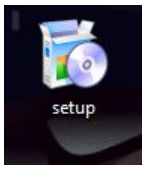
El análisis léxico es el proceso a través del cual se leen los caracteres de un programa fuente y se agrupan en secuencias representativas denominados tokens, la salida del análisis léxico es una lista de tokens, la cual se ingresa a la siguiente fase del compilador, el análisis sintáctico.

El análisis sintáctico consiste en verificar la consistencia gramatical de un programa fuente, generalmente la salida de este análisis es una representación intermedia denominada árbol de análisis sintáctico.

Pre-Requisitos

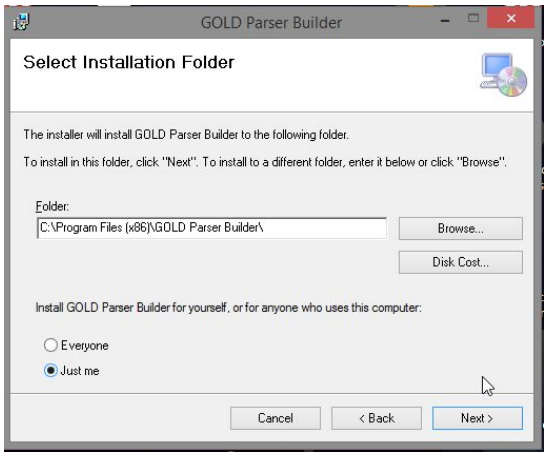
GOLD Parser Builder

Este programa puede descargarse del siguiente enlace, descomprimimos los archivos y para la versión de Windows el instalador es un .exe por lo tanto deberemos de hacer click sobre el archivo que dice setup.exe

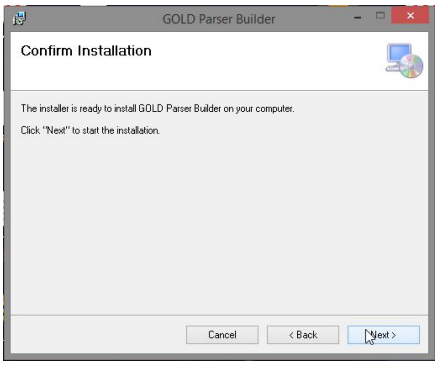


Esto nos desplegará el asistente de instalación, es bastante intuitivo y en la mayoría de las opciones solo hay que seleccionar “siguiente”, pero las opciones se detallan a continuación:

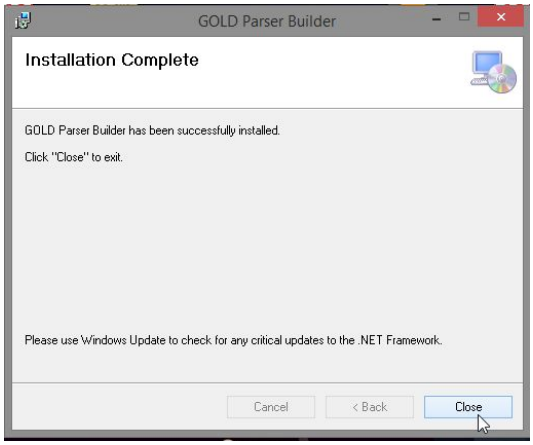
En la primera ventana no debemos de seleccionar nada, por lo que únicamente presionaremos el botón de siguiente.



Se nos preguntara en que ruta deseamos instalar GOLD Parser, debemos seleccionar la ruta (por defecto es “C:\Program Files (x86)\GOLD Parser Builder”) y seleccionar si queremos instalarlo para todos los usuarios o solo para nosotros, en este caso seleccione “solo para mí”.



Daremos click en siguiente para confirmar la instalación.



Se nos mostrará la ventana de confirmación que indica que GOLD Parser fue instalado correctamente.

Gramática Utilizada

En el ejemplo original realizado en flex y cup, del que se inspiró este proyecto, la gramática planteada fue la siguiente:

```
ini::=instrucciones;

instrucciones ::=
   instruccion instrucciones
 | instruccion
 | error instrucciones
;

instruccion ::=
   REVALUAR CORIZQ expresion:a CORDER PTCOMA{:System.out.println("El valor de la expresión es: "+a);:}
;

expresion  ::=
    MENOS expresion:a                   {:RESULT=a*-1;:}%prec UMENOS
 |  expresion:a MAS       expresion:b   {:RESULT=a+b;:}
 |  expresion:a MENOS     expresion:b   {:RESULT=a-b;:}
 |  expresion:a POR       expresion:b   {:RESULT=a*b;:}
 |  expresion:a DIVIDIDO  expresion:b   {:RESULT=a/b;:}
 |  ENTERO:a                           {:RESULT=new Double(a);:}
 |  DECIMAL:a                          {:RESULT=new Double(a);:}
 |  PARIZQ expresion:a PARDER           {:RESULT=a;:}
;
```

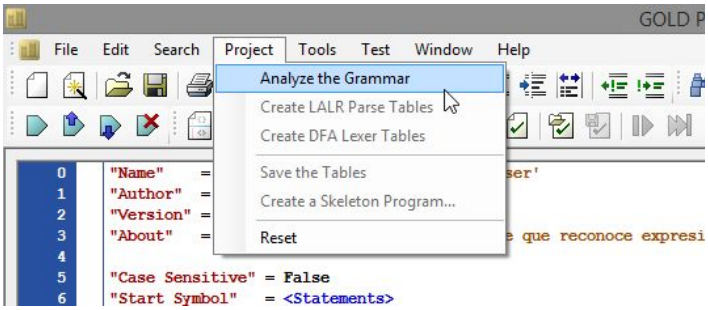
Estas herramientas permiten definir la precedencia y asociatividad de los operadores sin necesidad de modificar la gramática, en el caso de GOLD Parser, esta opción no está disponible, por lo que necesitamos utilizar una gramática no ambigua que respete la precedencia y asociatividad de estos. Se propuso la gramática a continuación:

```
ENTERO = {Digit}+
DECIMAL = ENTERO('.'ENTERO)?

<Statements> ::= <Statement> <Statements>
               | <Statement>

<Statement>  ::= Evaluar '[' <Expression> ']'';'

<Expression>   ::= <Expression> '+' <Mult Exp>
                 | <Expression> '-' <Mult Exp>
                 | <Mult Exp>

<Mult Exp>     ::= <Mult Exp> '*' <Negate Exp>
                 | <Mult Exp> '/' <Negate Exp>
                 | <Negate Exp>

<Negate Exp>  ::= '-' <Value>
                | <Value>

<Value>       ::= ENTERO
                | DECIMAL
                | '(' <Expression> ')'
```

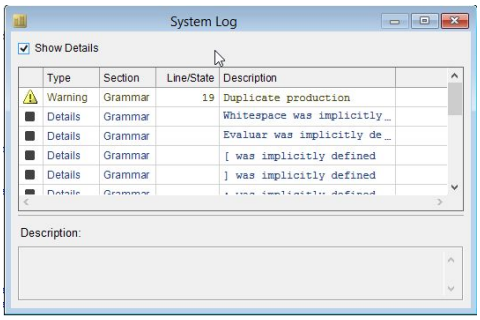
Creando la Gramática

Para Comenzar deberemos definir la gramática, para este caso utilizaremos la gramática de una calculadora básica, que permite evaluar operaciones aritméticas. Toda la documentación relacionada sobre la sintaxis del GOLD Parser y ejemplos de gramáticas puede encontrarse en [este enlace.]

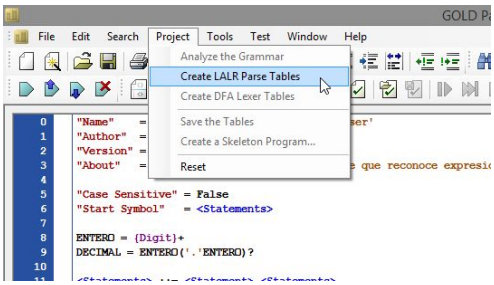
Una vez tengamos lista nuestra gramática, seleccionaremos en la pestaña de Project y seleccionaremos la primera opción “Analyze the Grammar” esto analizara la gramática y nos mostrará los conflictos si es que existiesen. Debemos corregir todos los errores antes de continuar, para este ejemplo no había ninguno.



Si existiesen errores o notificaciones se nos mostrarán en una ventana emergente de la siguiente manera:

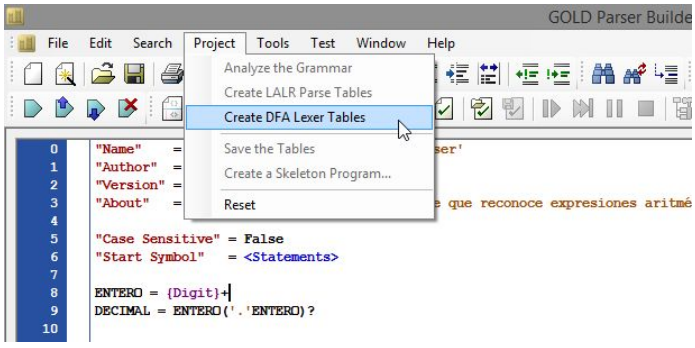


Cuando verifiquemos que nuestra gramática no contiene ningún error, podremos proceder a crear las tablas para el análisis LALR, estas son las encargadas de realizar el análisis sintáctico de la entrada que proporcionamos.

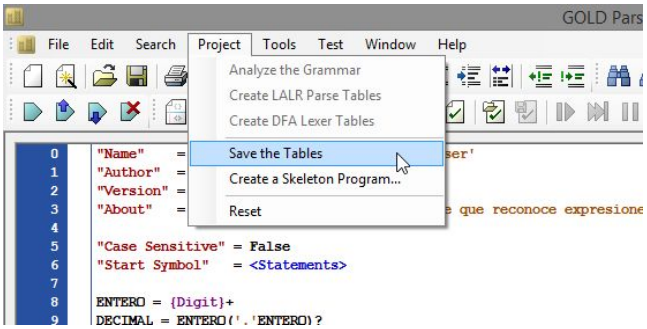


C

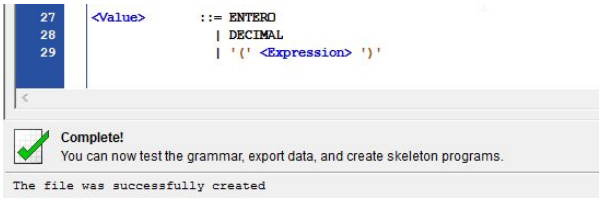
Durante la creación de las tablas existe la posibilidad de que tengamos un conflicto de desplazamiento-reducción o reducción-reducción y el asistente no nos permitirá continuar, primero deberemos de resolver estos conflictos. Si, por el contrario, no tenemos ningún error, podremos continuar al último paso, crear al autómata finito determinista que será el encargado de realizar el análisis léxico.



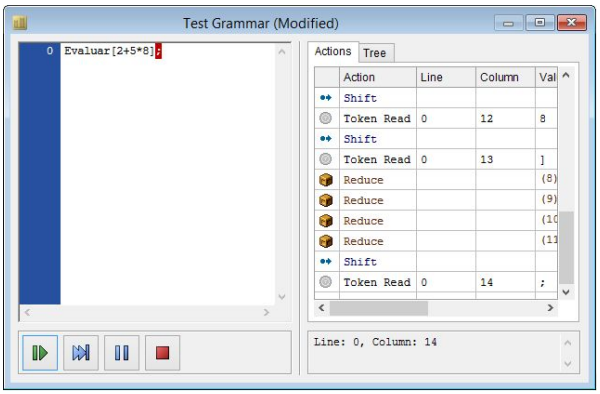
Generalmente si no hemos tenido ningún error en los procesos anteriores, este proceso no nos mostrará ningún error. Una vez concluidos todos los procesos, procederemos a guardar todas las tablas, estas son las que tenemos que importar en nuestro programa para que se pueda realizar el análisis.



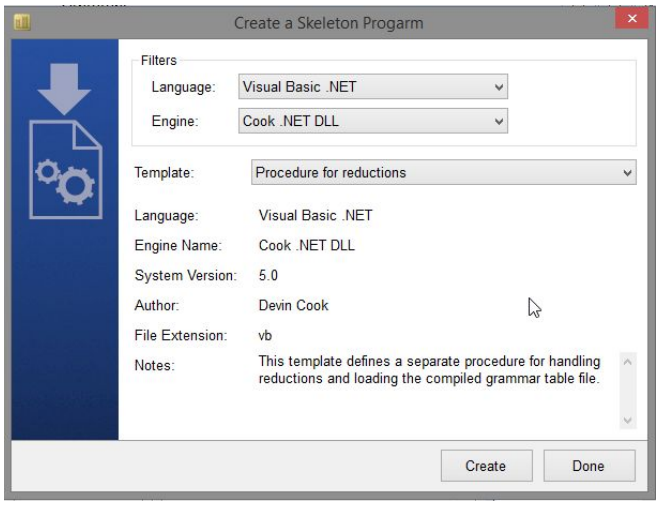
Se nos mostrará una ventana para que indiquemos la ruta en la cual deseamos almacenar las tablas, la seleccionamos y damos click en aceptar, esta ventana se cerrará y GOLD Parser nos mostrará un mensaje diciendo que se guardaron las tablas correctamente.



Una de las principales ventajas de GOLD Parser es que tiene un “debugger” que permite visualizar el proceso de análisis de una forma detallada, se proporciona una entrada y por medio de una interfaz gráfica se muestran los estados en los que se encuentra el análisis, los desplazamientos y reducciones realizadas, etc.

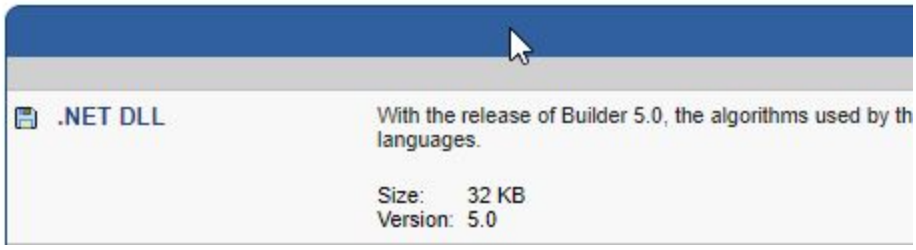


El siguiente paso es crear el esqueleto de un programa, seleccionaremos la pestaña de proyecto y la opción de “create a Skeleton Program”. Al ser GOLD Parser un analizador multiplataforma, tendremos una vasta cantidad de opciones al momento de seleccionar el lenguaje, para este ejemplo en específico utilizaremos Visual Basic .net y como motor Cock.net DLL

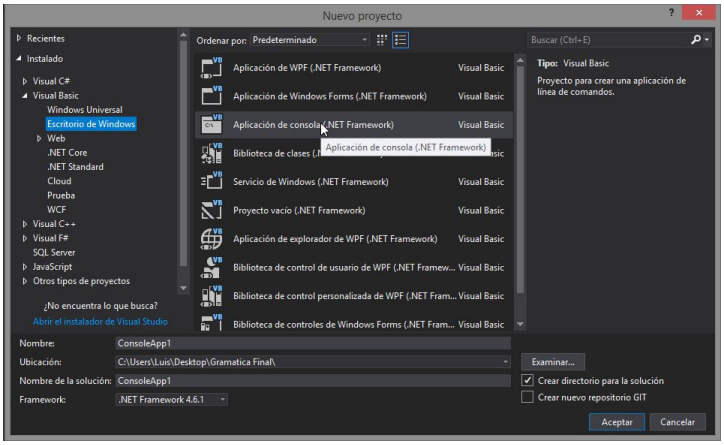


Seleccionaremos la opción de crear y nos mostrará una ventana desde la cual podremos seleccionar la ruta en la cual queremos guardar el "esqueleto" de nuestro programa. Esto es todo el procedimiento que debemos de realizar en GOLD Parser, de acá en adelante continuaremos utilizando Visual Studio.

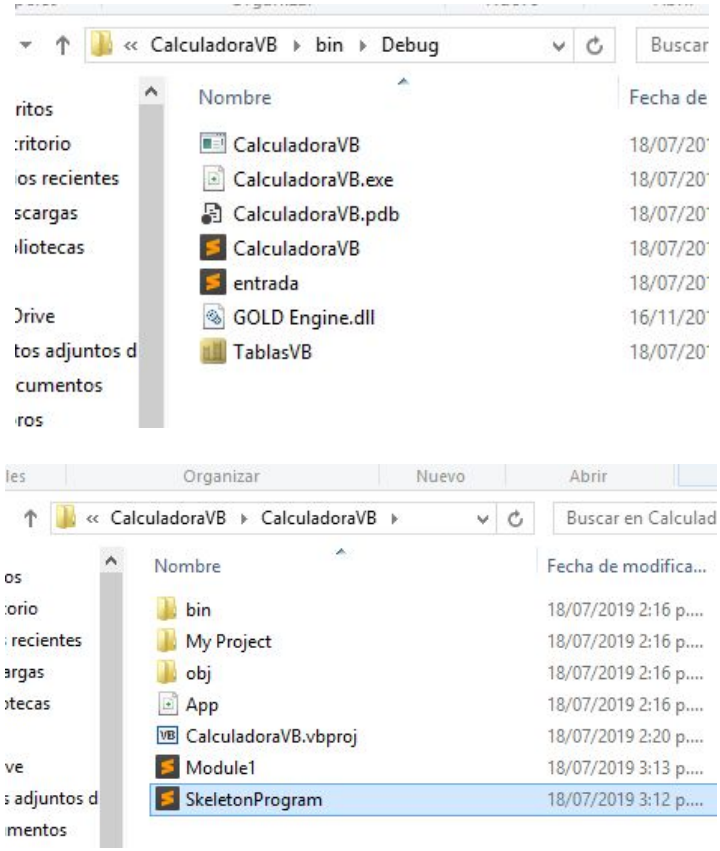
Para poder utilizar los archivos que generamos desde Visual Studio necesitaremos importar la librería que realiza el proceso de análisis, dicha librería la podremos descargar desde el siguiente enlace.



Abriremos Visual Studio y seleccionamos Nuevo Proyecto y dentro de este apartado seleccionamos Visual Basic y específicamente la opción de "Aplicación de consola". Esto es para este ejemplo en específico, pero podremos crear cualquier tipo de proyecto, desde una aplicación de escritorio hasta una aplicación WEB y nuestro analizador funcionaria de la misma manera.

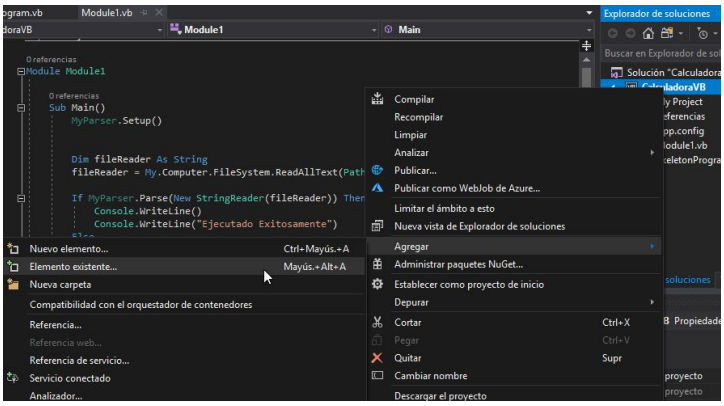


Con el proyecto creado debemos de pegar el archivo de las tablas de análisis que generamos desde GOLD Parser (el archivo con extensión .egt) y la librería que acabamos de descargar (el archivo con extensión .dll) en la carpeta /bin/debug de nuestro proyecto. También debemos de pegar el “esqueleto” que generamos, pero este lo pegaremos en la carpeta principal de nuestro proyecto.

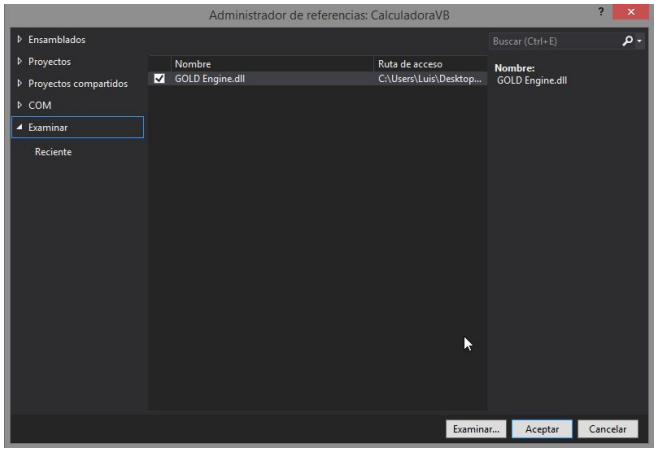


Nuevamente en Visual Studio, desde el explorador de soluciones debemos de realizar dos procedimientos, el primero es importar el archivo que contiene el “esqueleto” de nuestro

programa, en mi caso se llama SkeletonProgram.vb, para ello daremos click derecho sobre el nombre del proyecto -> agregar -> elemento existente y seleccionamos nuestro archivo.



El segundo es importar la librería que acabamos de pegar en nuestra carpeta debug, para hacerlo daremos click derecho en referencia -> agregar referencia, esto nos desplegara una nueva ventana, seleccionamos la opción de examinar y seleccionamos nuestro archivo .dll y daremos click en aceptar.



Por último, debemos de abrir el archivo que contiene el “esqueleto” del programa y dentro de las líneas de código buscar el método con el nombre Setup. En la única instrucción que posee este método, debemos de cambiar el nombre de gramar.egt al nombre con el que guardamos nuestras tablas, en mi caso es TablasVB.egt, es de suma importancia que hayamos pegado nuestras tablas en la carpeta debug, de otra manera nuestro programa no las podrá encontrar y nos arrojará un error en tiempo de ejecución.

```
referencia
ublic Sub Setup()
    'This procedure can be called to load the parse tables. The class can
    'read tables using a BinaryReader.

    Parser.LoadTables(Path.Combine(System.AppDomain.CurrentDomain.BaseDirectory, "TablasVB.egt"))
nd Sub
```

Estas son todas las configuraciones que debemos de realizar para poder utilizar GOLD Parser, de acá en adelante, el tutorial se enfoca en explicar el funcionamiento de los archivos que se generaron anteriormente.

Utilizando el “esqueleto” del analizador

En los pasos anteriores nos enfocamos en la definición de la gramática y las configuraciones que tenemos que realizar para que podamos utilizar los archivos que generamos von GOLD Parser, pero no hemos programado ningún tipo de instrucción dentro de nuestro programa. Al ser un analizador multiplataforma, este se centra en generar un árbol de análisis sintáctico, de esta manera nosotros podremos recorrer este árbol a nuestro gusto y antojo y no tendremos ninguna limitación, es por ellos que no nos permite incrustar acciones semánticas al momento de definir la gramática, debemos de realizarlo directamente en el archivo “esqueleto”

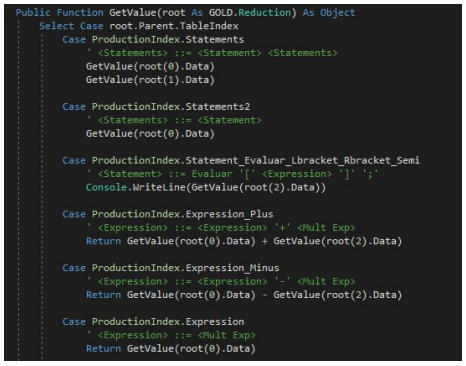
Podremos notar que este archivo, en el método parse, posee una serie de estados los cuales por defecto tienen comentado su funcionamiento, los únicos que utilizaremos son los de LexicalError y SyntaxError para poder realizar los reportes de errores y el de Accept para poder crear la raíz de nuestro árbol de análisis sintáctico.

```
Select Case Response
    Case GOLD.ParseMessage.LexicalError
        'Cannot recognize token
        Done = True

    Case GOLD.ParseMessage.SyntaxError
        'Expecting a different token
        Done = True

    Case GOLD.ParseMessage.Reduction
        'Create a customized object to store the reduction
        '.CurrentReduction = CreateNewObject(Parser.CurrentReduction)

    Case GOLD.ParseMessage.Accept
        'Accepted!
        'Program = Parser.CurrentReduction  'The root node!
        Root = Parser.CurrentReduction
        GetValue(Root)
        Done = True
        Accepted = True

    Case GOLD.ParseMessage.TokenRead
        'You don't have to do anything here.

    Case GOLD.ParseMessage.InternalError
        'INTERNAL ERROR! Something is horribly wrong.
        Done = True

    Case GOLD.ParseMessage.NotLoadedError
        'This error occurs if the CGT was not loaded.
        Done = True

    Case GOLD.ParseMessage.GroupError
        'COMMENT ERROR! Unexpected end of file
```

Debemos de definir una variable llamada Root (puede ser cualquier nombre, Root es para este ejemplo en específico) la cual será la raíz de nuestro árbol, esta variable es de tipo GOLD.Reduction. El resultado del análisis es un árbol n-ario y podremos asignarlo a la variable Root colocando la siguiente instrucción en el estado de aceptación de nuestro analizador

```
Root = Parser.CurrentReduction
```

Adicionalmente podremos notar que el archivo también posee una función denominada CreateNewObject que posee una serie de casos los cuales tiene comentado a que producción de la gramática pertenecen, es aquí donde debemos de introducir las acciones semánticas que deseamos que se ejecuten al momento de reducir por esa producción.



Esta función viene definida únicamente como una plantilla, pero podremos darle la funcionalidad que nosotros deseemos, cambiar el tipo de retorno, agregar un parámetro para usarlo como atributo heredado, el único límite es nuestra imaginación. Para este ejemplo en específico se le cambió el nombre por “GetValue” y el tipo de retorno es Object.

La manera más sencilla de recorrer este árbol es mediante llamadas recursivas a la función GetValue, esto se puede visualizar de mejor manera con la siguiente imagen:

```
Case ProductionIndex.Statements
    ' <Statements> ::= <Statement> <Statements>
    GetValue(root(0).Data)                          ''Recorre la produccion Statement
    GetValue(root(1).Data)                          ''Recorre la produccion Statements
```

Aquí se recorren los dos hijos del nodo Statements, no era necesario retornar ningún valor.

```
Case ProductionIndex.Value_Decimal
    ' <Value> ::= DECIMAL
    Return Double.Parse(root(0).Data.ToString())        ''Retorna la representacion decimal del numero
```

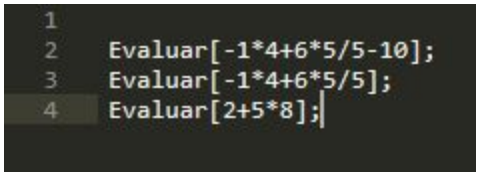
En el caso de los nodos hoja, se creaba la representación numérica del valor ingresado y se retornaba para que su nodo padre pudiera realizar operaciones con él. El árbol que nos genera GOLD Parser es completamente compatible con cualquier patrón de diseño, en lugar de retornar un número podríamos retornar un objeto, o una lista de objetos, prácticamente cualquier cosa, dejando la puerta abierta a un sinfín de posibilidades haciendo de GOLD Parser una herramienta altamente útil al momento de realizar la fase de análisis del proceso de compilación.

Por último, los métodos de nuestro parser son estáticos, por lo tanto, únicamente necesitamos enviarle la entrada sin la necesidad de crear una instancia de este, pero primeros debemos de ejecutar el método Setup para que cargue las tablas de análisis.

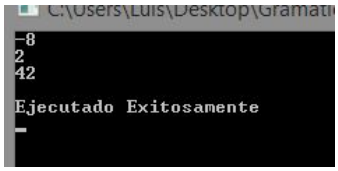
```
Sub Main()
    MyParser.Setup()

    Dim fileReader As String
    fileReader = My.Computer.FileSystem.ReadAllText(Path.Combine(System.AppDomain.CurrentDomain.BaseDirectory, "entrada.tx

    If MyParser.Parse(New StringReader(fileReader)) Then
        Console.WriteLine()
        Console.WriteLine("Ejecutado Exitosamente")
    Else
        Console.WriteLine("No se pudo ejecutar")
    End If

    Console.ReadLine()

End Sub
```

El programa por defecto está configurado para leer el archivo entrada.txt en la carpeta bin/debug del proyecto, pero también es escalable para poder utilizarlo con una interfaz gráfica, únicamente es necesario cambiar la cadena que se le envía al método Parse

A continuación, se muestra un ejemplo de cómo funciona nuestro programa al proporcionarle la siguiente entrada:



Salida:



Conclusiones

En conclusión, a pesar que GOLD Parser no es un generador de analizadores léxicos y sintácticos popularmente utilizado, resulta muy útil para personas que se deseen iniciar en el mundo de los compiladores ya que nos permite visualizar el proceso de compilación de una manera que no había visto con ningún otro generador de analizadores, y el hecho que nos genere un árbol que podamos manipular a nuestro gusto y antojo lo vuelve una herramienta altamente competitiva que cualquier estudiante de compiladores debería conocer.

Referencias

- Compiladores: Principios, técnicas y herramientas, 2da Edición, Alfred V. Aho, Monica S. Lam, Ravi Sethi y Jeffrey D. Ullman
- http://goldparser.org/about/how-it-works.htm
- Calculadora con Gold Parser (https://www.youtube.com/watch?v=JcWOEts-Cu4&t=1871s)