Definición general

Esta segunda etapa del proyecto conocida formalmente como parser o Análisis Sintáctico es, si se quiere la más importante del proyecto, pues de la gramática depende que el compilador se desempeñe de la mejor forma posible. De ahí que este proyecto se desarrolle con cuidado y lo mejor posible.
Para esta etapa del proyecto se debe entregar un programa que reciba un código fuente escrito en Solidity y realice el análisis léxico y sintáctico correspondiente. Para esto se debe utilizar la definición del lenguaje aceptado por el Scanner junto con la gramática. Por lo tanto el programa debe hacerse en Java utilizando Jflex y Cup
Al finalizar el parseo el programa deberá desplegarle al usuario el resultado del Análisis léxico y sintáctico que se efectúo. Se espera que despliegue

1. Listado de errores léxicos encontrados: El programa debe desplegar una lista de todos los errores léxicos que se encontraron en el código fuente. Debe desplegar la línea en la que se encontró el error. Es importante que el programa deba poder recuperarse del error y no desplegar los errores en cascada ni terminar de hacer el scaneo al encontrar el primer error.

2. Listado de errores sintácticos encontrados: El programa debe desplegar una lista de todos los errores sintácticos que se encontraron en el código fuente. Debe desplegar la línea en la que se encontró el error. Además, el mensaje de error debe ser lo más específico posible, con el fin de que el programador pueda llegar al error y corregirlo de forma eficiente. Es importante que el programa deba recuperarse del error y evitar no desplegar errores en cascada ni terminar de hacer el parseo al encontrar el primer error.

Descripción Detallada

Se les sugiere tomar en cuenta los siguientes aspectos para asegurar la completítud del programa.

- La estructura del programa es la siguiente. Un conjunto de contratos (debe venir al menos uno), cada uno con la siguiente lista de componentes. Deben venir en ese orden y puede no venir alguno de ellos.

```
pragma solidity ^0.4.0;
      contract NombreContrato {
       Enums
       Variables
       Estructuras
       Funciones
}
```

- Los **Enums** tienen la siguiente forma:

```
enum NombreEnum { id1, id2, id2…..}
```

la palabra reservada enum seguido por el id que identificará el enum y entre {} la lista de los identificadores que darán nombre a los valores del enum, divididos por ,

- Las variables pueden ser de tipo: bool, byte, bytes, address, int, string, ufixed, uint y cualquiera de los tipos de tipos definidos con su tamaño, ej: uint8, bytes4 .

  La declaración de **Variables** tiene la siguiente forma:

```
tipo [public/private] IdVariable;
tipo[] IdVariable;    // para los arrays
```

La declaración puede ir seguida de una asignación:

```
int public x = 2;
```

- Las **Estructuras** tienen la siguiente forma:

```
struct IdStruct {
        //conjunto de declaraciones de Variables
    }
```

- La estructura de las **funciones** es la siguiente

```
function  idFuncion (tipo x, tipo y, ….  ) [modificadores] [returns(tipo id)]
{
    [declaraciones variables] (Pueden no venir y la estructura es igual que las del
    contrato)

    cuerpo

    return valor;
}
```

Los modificadores pueden ser: payable, private, public, internal.
En el cuerpo pueden aparecer:

- Las **asignaciones** se hacen así: `Variable =  Expresión;`
- **Llamadas a funciones**: `función(parámetros);`
- Las **expresiones** pueden ser valores literales, identiticador, operaciones aritméticas, operaciones booleanas, llamadas a funciones, true o false. En las operaciones se pueden mezclar.
- Recuerde que los identificadores dentro del cuerpo pueden ser de la forma `id.id, id.id(), this.id o this.id()` haciendo referencia a miembros y funciones de contratos.
- Las estructuras de control tienen la siguiente estructura:

```
while (expresionBooleana) {
  sentenecia 1
  sentencia 2
  ......
 }
```

```
do{
  sentencia 1
  sentencia 2
  ......
}
while (expresionBooleana);
```

```
for (asignación; expresion1; expresion2){
  sentenecia 1
  sentencia 2
  ......
}
```

```
if (expresionBooleana){
   sentencia1
   sentencia2
else {
    sentencia1
     sentencia2
…}  //el else podría o no venir
```

```
variablename =  (expBooleana)? value1:value2;
```

Las sentencias pueden ser asignaciones, llamadas a función, otras estructuras de control, sentencia de return, sentencia de break o sentencia de continue;

- Los operadores que se deben tomar en cuenta son los siguientes:

Aritméticos: "+" "-" "*" "/" "%" "(" ")"
"+=" "-=" “*=” "/="

Booleanos: "==" ">=" ">" <=" "<" "!=" "||" "&&" "!"

En este aspecto es importante recordar que las condiciones de las estructuras de control utilizan solamente expresiones booleanas. Para las asignaciones si pueden tener ambas operaciones.

- Se deben implementar buenos mensajes de error

Documentación

Se espera que sea un documento donde especifique lo siguiente:

a. Portada, índice, introducción
b. Estrategia de Solución: Una explicación de cómo lograron solucionar el proyecto. Puede incluir consideraciones de diseño, lógica o cualquier aspecto que consideren importante para llegar a la solución.
c. Análisis de Resultados: Deberá elaborar un listado de todas y cada una de las actividades y tareas que deben cubrirse a nivel funcional, para cada una de ellas debe aportar el porcentaje de realización y en caso de no ser el 100% debe justificarse.
d. Lecciones aprendidas: Debe prepararse un listado de las lecciones aprendidas producto del desarrollo de la tarea programada. Las lecciones aprendidas pueden ser de carácter personal y/o técnico que involucre aspectos que han logrado un aprendizaje en temas de investigación, desarrollo de habilidades técnicas y habilidades blandas como trabajo en equipo, comunicación, forma de expresar ideas, entre otros.
e. Casos de pruebas: se espera que definan claramente cada prueba, cuáles son los resultados esperados y cuáles fueron los resultados obtenidos. No es necesario que sean grandes pero deben evaluar la funcionalidad completa del programa.
f. Manual de usuario: especificar como compilar y correr su tarea.
g. Bitácora de trabajo durante las semanas de trabajo, incluyendo verificaciones realizadas (si existieran) de consultas realizadas con el profesor o asistente.
h. Bibliografía y fuentes digitales utilizadas

Aspectos Administrativos

- Los grupos deben permanecer iguales a la primera etapa del proyecto.
- El trabajo se debe de entregar el día 21 de Julio de 2020 antes de medianoche, enviarlo por correo o subirlo al tec digital.
- Recuerde que oficialmente no se recibirán trabajos con entrega tardía.
- https://solidity-es.readthedocs.io/es/latest/solidity-in-depth.html
- https://solidity-es.readthedocs.io/es/latest/structure-of-a-contract.html#funciones
- https://solidity-es.readthedocs.io/es/latest/control-structures.html