Aspectos léxicos, sintácticos y semánticos para DECAF

El proyecto para el curso de Construcción de Compiladores, consiste en la implementación de las fases de compilación para el lenguaje DECAF.

Consideraciones Léxicas

- Todas las palabras claves (if, while, etc.) son en minúsculas.
- El lenguaje es case-sensitive (ej: foo y Foo, son dos variables distintas)
- Los comentarios comienzan por // y terminan en el fin de línea
- El espacio en blanco se define como uno o más espacios, tabuladores, saltos de línea y comentarios.
- Las palabras claves e identificadores se encuentran separados por espacios en blanco.
- Los números, son valores decimales que van desde -2147483648 y 2147483647.
- Un char es cualquier caracter ASCII imprimible (32 al 126), además de contener las siguientes secuencias de escape: \\, \', \", \t, \n.
- Las palabras reservadas son: **class**, **struct**, **true**, **false**, **void**, **if**, **else**, **while**, **return**, **int**, **char**, **boolean.**
- **Program**, no es una palabra reservada, simplemente es el nombre de la clase o programa que se está analizando.

Consideraciones Sintácticas

- Ver la definición de la gramática DECAF.

Consideraciones Semánticas.

- Un programa en DECAF, consiste en una declaración de una clase llamada **Program.** La clase consiste en declaraciones de métodos (procedimientos) y atributos (variables, estructuras). La declaración de atributos dentro de la clase, los marca como atributos globales, es decir que pueden ser utilizados en todos los métodos de la clase.
- El programa debe de tener una declaración de un método llamado **main**, el cual será el punto de partida en la ejecución del programa.
- Tipos
  - Los tipos básicos del lenguaje son **int**, **char y boolean.** Estos tipos básicos (y otros tipos derivados) se pueden utilizar como base para derivar nuevos tipos con la palabra reservada **struct.**
  - De todos los tipos del lenguaje (básicos y derivados) se pueden declarar arreglos de una dimensión (int[N], char[N], boolean[N]), cuyo tamaño es definido en tiempo de compilación. Debido a esto, no se puede declarar arreglos como parámetros.
- Ámbito
  - Una variable debe de ser declarada antes de su uso.
  - Un método puede ser llamado solamente luego de ser declarado (métodos recursivos son permitidos)
  - Existen dos ámbitos principales en el programa, el ámbito Global y el ámbito Local. El ámbito global es el que se introduce en la definición de la clase **Program.** El ámbito local es el que se introduce dentro de la definición de un método (variables locales y parámetros formales). Además dentro de las estructuras de control, pueden definirse nuevos ámbitos (es decir, cada <block> es un nuevo ámbito dentro de la ejecución del programa).
  - Los identificadores en un ámbito local, pueden ocultar identificadores y métodos del ámbito global.

  - Identificadores en un ámbito local, pueden ocultar identificadores en ámbitos anteriores e identificadores y métodos en el ámbito global.
  - Ningún identificador (y método en el caso de ámbito global) puede ser definido más de una vez en un mismo ámbito.
- Valores iniciales
  - Una variable tipo **int** tendrá un valor inicial de 0.
  - Una variable tipo **boolean** tendrá un valor inicial de **false**.
  - Las variables son inicializadas al momento de entrar en su ámbito.
- Asignación
  - <location> = <expr> copia el valor resultante de evaluar <expr> en <location>
  - En <location> = <expr>, <location> y <expr> deben de tener el mismo tipo. Si <location> se refiere a un arreglo, la posición debe de estar definida. Si <location> se refiere a una estrucutra, el miembro debe de estar definido.
- Invocación de métodos y valor devuelto
  - Los argumentos en el momento de invocar a un método se pasan por valor.
  - Los argumentos formales de un método son considerados variables locales del método y son inicializados por asignándole el valor que resulta de las expresiones de los argumentos. Los argumentos son evaluados de izquierda a derecha.
  - Un método que devuelva **void** solamente puede ser utilizado como <statement>, es decir que no puede ser utilizado en ninguna expresión. El método devuelve el control al método que lo llamo por medio de la palabra reservada **return** ó cuando se termina el texto del método.
  - Un método que devuelve un valor puede ser utilizado dentro de una <expression> y como <statement>. El resultado de la llamada es el resultado que se obtiene al evaluar <expression> en la instrucción **return <expression>**. El tipo de este resultado debe de ser el mismo que el tipo que devuelve el método.
- Sentencias de control
  - **if**
    - Primero se evalua <expr>, si <expr> es del tipo **boolean** y tiene valor **true**, entonces se ejecuta el bloque correspondiente a **true**, de lo contrario si existe un **else**, se ejecuta su bloque correspondiente.
  - **while**
    - El bloque correspondiente, se ejecuta mientras que el tipo de <expr> sea **boolean** y tenga valor **true**.
- Expresiones
  - Las expresiones siguen las reglas normales de evaluación. Los paréntesis sirven para agrupar y modificar la precedencia de operadores.
  - La expresión <location> se evalua al valor que contiene dicho <location>
  - Los operadores aritméticos <arith_op> y el – unario, poseen la precedencia y significado normalmente aceptada. Estos operadores son utilizados en variables del tipo **int**.
  - Los operadores relacionales <rel_op> se utilizan para comparar tipos expresiones del tipo **int**. La comparación devuelve un tipo **boolean**.
  - Los operadores de comparación <eq_op> se utilizan para comparar expresiones del tipo **int, char y boolean**. Las expresiones a comparar deben de tener el mismo tipo. Esta comparación devuelve un tipo **boolean**.
  - Los operadores booleanos <cond_op> se utilizan en expresiones con operandos tipo **boolean**. Dicha expresión se evaluará utilizando una evaluación en corto circuito, es decir la expresión del segundo operando no es evaluada o ejecutada si el resultado del primer operando determina el valor de toda la expresión.
- Funciones de Entrada/Salida
  - Se definirán firmas para métodos para entrada y salida de datos. Estas firmas generarán código ensamblador que implementen la lectura del teclado y

la impresión en pantalla específicamente. Los tipos de dato que se leerán del teclado e imprimirán en pantalla son **int** y **char**.

- Resumen de reglas semánticas.
  - Ningún identificador es declarado dos veces en el mismo ámbito
  - Ningún identificador es utilizado antes de ser declarado.
  - El programa contiene una definición de un método **main** sin parámetros, en donde se empezará la ejecución del programa.
  - **num** en la declaración de un arreglo debe de ser mayor a 0.
  - El número y tipos de argumentos en la llamada a un método deben de ser los mismos que los argumentos formales, es decir las firmas deben de ser idénticas.
  - Si un método es utilizado en una expresión, este debe de devolver un resultado.
  - La instrucción **return** no debe de tener ningún valor de retorno, si el método se declara del tipo **void**.
  - El valor de retorno de un método debe de ser del mismo tipo con que fue declarado el método.
  - Si se tiene la expresión id[<expr>], id debe de ser un arreglo y el tipo de <expr> debe de ser **int**.
  - El tipo de <expr> en la estructura **if y while**, debe de ser del tipo **boolean**.
  - Los tipos de operandos para los operadores <arith_op> y <rel_op> deben de ser **int**.
  - Los tipos de operandos para los operadores <eq_ops> deben de ser **int**, **char o boolean**, y además ambos operandos deben de ser del mismo tipo.
  - Los tipos de operandos para los operadores <cond_ops> y el operador **!** deben de ser del tipo **boolean**
  - El valor del lado derecho de una asignación, debe de ser del mismo tipo que la locación del lado izquierdo.