CONSIDERACIONES GENERALES:

El trabajo deberá realizarse en grupos de dos o tres integrantes.

El compilador se implementará en tres etapas:

- Análisis léxico y sintáctico.
- Análisis léxico, sintáctico y semántico con generación del AST.
- Análisis léxico, sintáctico y semántico con generación de código ejecutable a partir del AST.

Para la aprobación de la cursada no se exigirá la implementación total del compilador pero si se pedirá la entrega de una parte en cada una de las etapas, dejando el resto de la implementación para ser presentada al momento de rendir el examen final.

La aprobación de cada etapa será requerida para acceder a la siguiente etapa.

ETAPAS:

1ra Etapa: Realizar un analizador sintáctico utilizando las herramientas vistas para tal fin.

El programa debe mostrar por pantalla las reglas sintácticas que va realizando el analizador, teniendo como entrada un archivo con código fuente dado para la validación del compilador.

Será requerido: Implementar el análisis léxico y sintáctico para el lenguaje descripto en este documento, pudiendo hacer una entrega en la que sólo se incluya una de las estructuras de selección (**`if`** o **`when`**), una de las estructuras de iteración (**`while`** o **`for`**) y uno de los casos especiales a considerar en el análisis léxico (ignorar comentarios correctamente balanceados o realizar el procesamiento de los códigos de escape dentro de las cadenas de caracteres).
No está de más aclarar que siempre será preferible una entrega soportando todo lo mencionado.
En esta etapa no se pedirá realizar una validación semántica del código analizado.

Fecha de entrega: **5 de mayo de 2021**.

2da Etapa: Aumentar el analizador sintáctico desarrollado en la etapa anterior para que pueda construir un AST semánticamente correcto para el código de un programa dado como entrada.

El AST del programa compilado se deberá mostrar a través de un archivo escrito con sintaxis DOT para generar una visualización mediante graphviz[1]. En esta etapa, el compilador deberá verificar que los tipos de datos de las expresiones, variables, funciones y parámetros sean correctos, agregando conversiones implícitas donde se las necesite o reportando errores donde no se las admita (por ejemplo, un valor booleano en una expresión numérica o un valor numérico en una expresión booleana). Además, deberá asegurar que los identificadores (variables o funciones) existan en el ámbito donde se los utiliza y su uso sea correcto.

Será requerido: Implementar en el AST y visualizar las mismas sentencias implementadas en la etapa anterior. Soportar la validación de tipos y reportar los errores apropiados.
No se exigirá agregar conversiones implícitas.

Fecha de entrega: **31 de junio de 2021**.

3ra Etapa: Realizar un compilador utilizando el código desarrollado en la etapa anterior.

El compilador deberá generar un AST para el código del programa dado como entrada y, tras validarlo semánticamente y realizar las optimizaciones pertinentes, generar código intermedio apropiado (usando la IR de LLVM) para producir una versión ejecutable del programa dado.

Será requerido: A definirse.

Fecha de entrega: **A definirse**.

[1] https://www.graphviz.org

DEFINICIÓN SEMÁNTICA DEL LENGUAJE

Variables:

Toda variable definida sin una expresión asociada a la misma se asume inicializada con:

- `0` para el tipo **`integer`**.
- `0.0` para el tipo **`float`**.
- **`false`** para el tipo **`boolean`**.

Conversiones:

Se soportan conversiones implícitas entre los tipos de datos **`integer`** y **`float`**, promoviendo un entero a un número representado en punto flotante o truncando un número representado en punto flotante al número entero más próximo.

No hay sintaxis para conversiones explícitas ni para conversiones entre números y valores booleanos.

Expresiones:

Todas las variables que aparezcan en una expresión obligatoriamente deberán estar definidas antes de su uso. La única excepción se da al referenciar dentro de una función a variables globales definidas luego de dicha función.

Para operadores aritméticos o relacionales, se realizarán las conversiones implícitas necesarias para que todos los valores participantes en la operación sean del mismo tipo.

Lo mismo aplicará a los valores retornados de invocaciones a funciones, a los valores pasados como parámetros reales y a la expresión retornada por una función, donde se realizarán las conversiones necesarias para que los tipos de las expresiones coincidan con los tipos esperados.

Sentencias **`break`** y **`continue`**:

Dichas sentencias sólo podrán aparecer dentro de un bloque definido por un **`while`** o un **`for`** y, en el caso de sentencias de iteración anidadas, aplicarán al lazo más cercano a la aparición del **`break`** o **`continue`**.

Sentencia **`return`**:

La sentencia **`return`** sólo podrá aparecer dentro del cuerpo de una función, no admitiendose como una de las sentencias del programa principal.

Verificación de tipos:

Deberán reportarse todas las situaciones donde una conversión implícita no pueda convertir una expresión al tipo esperado.

Ámbitos y definición de variables:

Existen dos ámbitos claramente marcados: El ámbito global y el ámbito local a una función. Además, cada bloque define un nuevo ámbito implícito dentro del ámbito que lo contiene.

Todas las variables definidas dentro de un bloque o una función existirán únicamente dentro del ámbito asociado a dicho bloque o función.

Una variable no podrá ocultar la definición de otra variable definida previamente, sin importar si la definición se encuentra en el mismo ámbito o en alguno de los ámbitos que lo engloban.

Así, por ejemplo, una variable local a una función o local al programa principal no podrá tener el mismo nombre que una variable global.

De la misma forma, una variable definida en el cuerpo de un if dentro de una función, por ejemplo, no podrá llamarse igual que una variable local definida en el cuerpo de dicha función.

Las expresiones podrán hacer referencia a variables definidas anteriormente dentro del mismo ámbito o en ámbitos locales que lo engloban.

También podrán hacer referencia a variables definidas en el ámbito global sin la restricción de que deban estar definidas antes de su uso.

Los parámetros de una función son considerados como locales a dicha función y cumplen con las mismas restricciones que las variables locales.

Por ejemplo, en el siguiente fragmento de un programa, se pueden ver varios ámbitos:

```
function ejemplo() return boolean                          Ámbito global
begin
  variable local_a_ejemplo is boolean = true;              Ámbito local a ejemplo
  if global > 10 then
  begin
    variable local_al_then is float = global;              Ámbito local al then del if
    while local_al_then > 5 do
    begin
       local_al_then = local_al_then - 1;                  Ámbito local al while
       variable local_al_while is float = local_al_then * 1.5;
       writeln(local_al_while);
    end;                                                   Ámbito local al then del if
    writeln(local_al_then);
    local_a_main = false;
  end else
    global = global + 5;                                   Ámbito local al else del if
  writeln(local_a_main);
  return local_a_ejemplo;                                  Ámbito local a ejemplo
end;                                                       Ámbito global

variable global is int = 42;

main is
  variable local_al_main is boolean = true;                Ámbito local al main

  if not local_al_main then
      variable local_al_then is integer = 1                Ámbito local al then
  else
      ejemplo();                                           Ámbito local al else
end.
```