PROCESADORES DE

LENGUAJES

Práctica Obligatoria

Entrega final

GRADO EN INGENIERÍA INFORMÁTICA

ETSII

Texto

Descripción generada automáticamente

Grupo 30

Gonzalo de la Cruz Cepeda

Álvaro Brea Arjona

Contenido

[Introducción: 3](#_Toc75527515)

[Gestión de errores: 3](#_Toc75527516)

[Recuperación por escaneo de los siguientes tokens: 3](#_Toc75527517)

[Recuperación por tokens equivocados: 3](#_Toc75527518)

[Recuperación por errores en subreglas: 3](#_Toc75527519)

[Recuperación de error Fallo-Seguro: 4](#_Toc75527520)

[Trabajo realizado: 4](#_Toc75527521)

[Traducción dirigida por sintaxis: 5](#_Toc75527522)

[Nivel Aprobado: 5](#_Toc75527523)

[Nivel Notable: 5](#_Toc75527524)

[Casos de prueba: 6](#_Toc75527525)

[Caso1A: 7](#_Toc75527526)

[Caso1B: 8](#_Toc75527527)

[Caso2A: 9](#_Toc75527528)

[Caso2B: 10](#_Toc75527529)

[Caso3A: 11](#_Toc75527530)

[Caso3B: 12](#_Toc75527531)

[Caso4A: 13](#_Toc75527532)

[Caso4B: 14](#_Toc75527533)

[Anexo: 15](#_Toc75527534)

[Cabeceras: 15](#_Toc75527535)

[Siguientes: 15](#_Toc75527536)

[Directores: 16](#_Toc75527537)

Introducción:

La práctica consiste en implementación de un visualizador de código fuente para un lenguaje de programación.

Esto se llevará a cabo partiendo de un fichero de texto de entrada con el código, y terminará con un fichero HTML en el que aparecerá dicho código, mostrado de forma correcta, añadiendo tabulaciones y haciendo uso de las reglas de diseño en la programación.

Para llevar a cabo dicha tarea, haremos uso del lenguaje ANTLR v4, Java y HTML.

Gestión de errores:

Para este apartado, hemos añadido una nueva clase llamada errorListener.java para sustituir el errorListener() que actuaba por defecto. En él hemos seguidos los pasos explicados en el libro de ANTLR recomendado por la asignatura. Hemos añadido un par de métodos, uno que nos señalase la ubicación del error, otro que nos tradujese los cuatro tipos de errores mencionados en el libro.

En cuanto a la gestión de errores a la hora de generar el HTML, hemos hecho que cuando el código introducido, tenga algún error, no se generará ni código HTML, ni el propio fichero .html.

Recuperación por escaneo de los siguientes tokens:

Cuando el parser se encuentra con un error del que no hay recuperación posible de otra manera, empieza a ignorar tokens hasta que reconozca un estado conocido. Ese estado conocido es básicamente un token desde el que puede volver a continuar analizando.

A este método como el “modo pánico”, pero en este caso funciona realmente bien.

El parser, cuando no encuentra la entrada necesaria para el analizador sintáctico, empieza a ignorar tokens hasta llegar a un estado conocido desde el que se pueda recuperar.

Recuperación por tokens equivocados:

Una de las operaciones más utilizadas por el parser, es el de hacer un “match” o emparejar tokens con la gramática. Si el token que está siendo reconocido, se notifica a los ErrorListeners() y se intentará resincronizarlo.

Para resincronizarlo, hay tres formas posibles: Puede borrar el token, puede añadir uno o puede lanzar una excepción para activar el mecanismo de “sync-and-return”.

Recuperación por errores en subreglas:

Esta estrategia de recuperación se utiliza en situaciones en las que se da un error dentro de un bucle. ANTLR intenta realizar la eliminación de un token utilizada en la estrategia anterior, y si no funciona, y el error se produce después de haber reconocido una iteración correcta del bucle, ANTLR ignora tokens hasta que encuentra algo que encaje con el contenido del bucle, lo que va justo después, o el conjunto de resincronización.

Recuperación de error Fallo-Seguro:

Por último, los parsers generados por ANTLR tienen una función llamada fail-safe, que garantiza que la recuperación de error termine si entra en un bucle, forzando a consumir un token si llega a la misma posición del parser con la misma entrada.

Trabajo realizado:

Para facilitar el trabajo en grupo y más de forma remota, hemos hecho uso de Github, y a continuación se encuentra una imagen del control de versiones del proyecto, en el que se han ido añadiendo las distintas partes del proyecto. Dicha imagen es antigua ya que si pusiésemos una imagen de la actual, serían más de 60 commits. En primer lugar, desarrollamos el analizador léxico para que nuestra gramática sea capaz de reconocer los identificadores, las constantes etc.

A continuación, añadimos la parte del analizador sintáctico que nos fue facilitado en el enunciado de la práctica. Empezando por la parte obligatoria seguida de la parte opcional. Este fragmento de código no cumplía las propiedades necesarias como para ser una gramática LL(1) ya que tenía recursividad por la izquierda y también requería de factorización. Así que con ayuda de las cabeceras y los directores que calculamos y que se incluyen más adelante en el anexo, hicimos de esta gramática, una gramática LL(1).

Mas adelante corregimos ciertos elementos que eran innecesarios y de esta manera se simplificaba el código.

Por último, se creó una clase llamada Síntesis(), en la que se encontraban todos los métodos para generar el String con el código HTML y por medio de llamadas desde el .g4, se iba traduciendo el código por medio atributos sintetizados.

Captura de pantalla de computadora

Descripción generada automáticamente

Traducción dirigida por sintaxis:

En cuanto a la traducción dirigida por sintaxis, se ha hecho únicamente uso de atributos sintetizados, de tipo String, en los que se ha ido introduciendo el código HTML y se ha ido concatenando.

Nivel Aprobado:

Para empezar, tendremos los 2 métodos principales del programa ya que son la estructura sobre la que se añadirán todos los demás. El primero será el método inic(), que creará la cabecera del HTML y mostrará el título del caso de prueba con el que se haya ejecutado el programa y abrirá la etiqueta del body para poder llamar a todos los demás métodos, y el método end(), que cerrará el body y el propio HTML para indicar la finalización de la escritura en el documento.

A continuación, tenemos el método cabecera(), que se encargará de como su propio nombre indica, mostrar las cabeceras de las funciones y procedimientos que se encuentren en el documento, enlazándolas con la propia función. Y por supuesto también se mostrarán las funciones y los procedimientos de forma completa gracias a distintas funciones menores concatenadas por atributos sintetizados.

Nivel Notable:

En cuanto a la parte opcional, tendremos la función div(), que se encargará de añadir un nivel de identación al código que se encuentra entre un “inicio” y un “fin”.

Para la comprobación de si existe una función o un procedimiento llamado “Principal”, tendremos una función llamada esPrincipal(), que se encargará de eso y por último para enlazar las variables o llamadas a funciones o procedimientos dentro de los mismos, se hará uso de las funciones referencia() y referenciado().

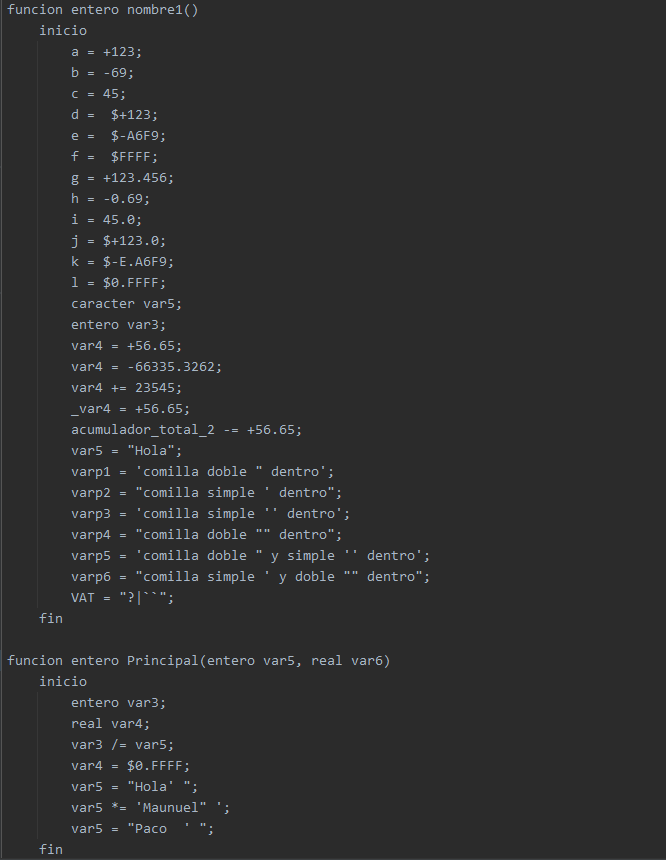
Casos de prueba:

Pasa estos 8 casos de prueba, estarán separados por la mitad en la que una mitad estará dedicada a los casos de correcta estructura y funcionamiento, y los otros cuatro, estarán dedicados a mostrar los errores.

Cada caso estará tematizado para mostrar las distintas funcionalidades de nuestro programa.

Caso1A:

Este primer caso de prueba estará dedicado a la declaración de variables.



Caso1B:

Este caso de prueba consistirá en los errores partiendo del caso 1A.

Texto

Descripción generada automáticamente

Caso2A:

En este segundo caso de prueba, se mostrarán ejemplos de operaciones y asignaciones.

Texto

Descripción generada automáticamente

Caso2B:

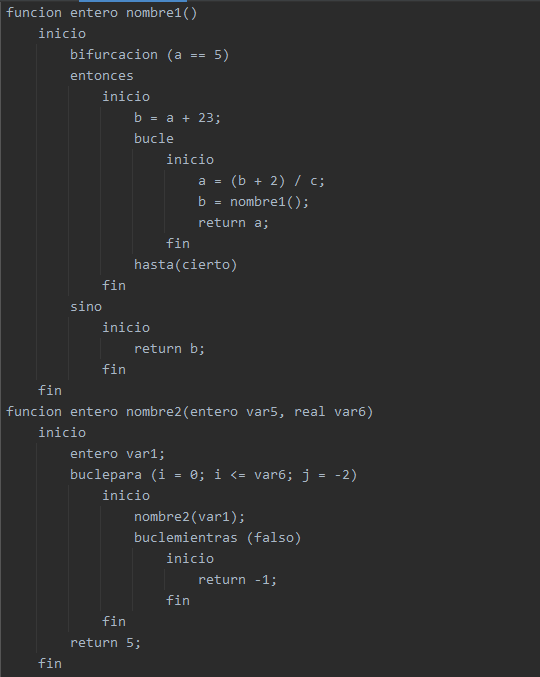
Este caso de prueba consistirá en los errores partiendo del caso 2A.

Texto

Descripción generada automáticamente

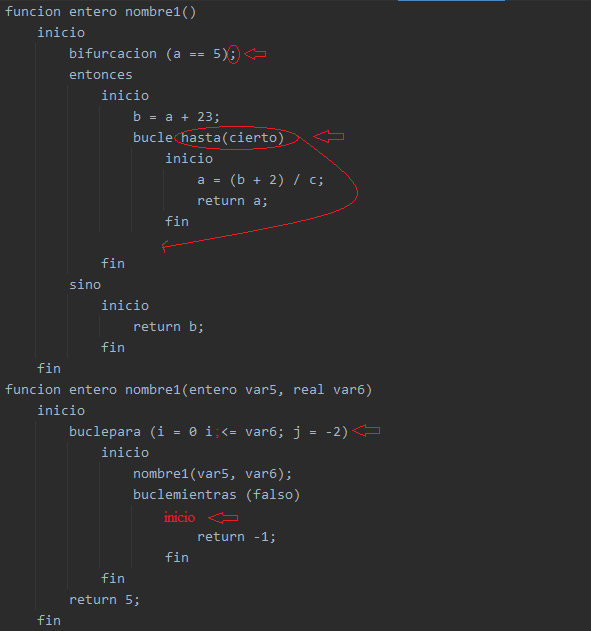
Caso3A:

Este caso de prueba consistirá en mostrar las distintas opciones del sent.



Caso3B:

Este caso de prueba consistirá en los errores partiendo del caso 3A.



Caso4A:

En este último caso de prueba, se dedicará a mostrar los comentarios.

Texto

Descripción generada automáticamente

Caso4B:

Este caso de prueba consistirá en los errores partiendo del caso 4A.

Texto

Descripción generada automáticamente

Anexo:

Cabeceras:

CAB(program) = CAB'(part) U CAB'(part) = {"function"}  
  
CAB(part) = {"function", "procedimiento"}  
  
CAB(restpart) = {IDENTIFICADOR}  
  
CAB(listparam) = ̶C̶A̶B̶'̶(̶l̶i̶s̶t̶p̶a̶r̶a̶m̶)̶ U CAB'(type) = {"entero", "real", "caracter"}  
  
CAB(type) = {"entero", "real", "caracter"}  
  
CAB(blq) = {"inicio"}  
  
CAB(sentlist) = ̶C̶A̶B̶'̶(̶s̶e̶n̶t̶l̶i̶s̶t̶)̶ U CAB'(sent) =  
 {"entero", "real", "caracter", "inicio, "return", "bifurcacion", "buclepara", "buclemientras", "bucle"}  
  
CAB(sent) = CAB'(type) U CAB'(blq) U {IDENTIFICADOR} U {"return"} U {"bifurcacion"} U {"buclepara"} U {"buclemientras"} U {"bucle"} =  
 = {"entero", "real", "caracter", "inicio, "return", "bifurcacion", "buclepara", "buclemientras", "bucle"}  
  
CAB(lid) = {IDENTIFICADOR}  
  
CAB(asig) = {"="} U {"+="} U {"-="} U {"\*="} U {"/="} =  
 = {"=", "+=", "-=", "\*=", "/="}  
  
CAB(exp) = C̶A̶B̶'̶(̶e̶x̶p̶)̶ U {IDENTIFICADOR} U {"("} U {CONSTENTERO} U {CONSTREAL} U {CONSTLIT} =  
 {IDENTIFICADOR, "(", CONSTENTERO, CONSTREAL, CONSTLIT}  
  
CAB(op) = {"+"} U {"-"} U {"\*"} U {"/"} =  
 = {"+", "-", "\*", "/"}  
  
CAB(lcond) = ̶C̶A̶B̶'̶(̶l̶c̶o̶n̶d̶)̶ U CAB'(cond) U {"no"} =  
 = {IDENTIFICADOR, "(", CONSTENTERO, CONSTREAL, CONSTLIT} U {"no"} =  
 = {IDENTIFICADOR, "(", CONSTENTERO, CONSTREAL, CONSTLIT, "no"}  
  
CAB(cond) = CAB'(exp) U {"cierto"} U {"falso"} = {IDENTIFICADOR} U {"("} U {CONSTENTERO} U {CONSTREAL} U {CONSTLIT} =  
 = {IDENTIFICADOR, "(", CONSTENTERO, CONSTREAL, CONSTLIT}  
  
CAB(opl) = {"y"} U {"o"} = {"y", "o"}  
  
CAB(opr) = {"=="} U {"<>"} U {"<"} U {">"} U {">="} U {"<="} =  
 = {"==", "<>", "<", ">", ">=", "<="}

Siguientes:

En cuanto a los siguientes, en este caso, no han hecho falta calcularlos, ya que, para la creación de los directores, al no haber λ, no serán necesarios.

Directores:

Estos directores están hechos antes de corregir la recursividad por la izquierda, para resolver la factorización cuando había más de un símbolo de anticipación. También deberíamos haber hecho los directores de las partes en las que se ha arreglado la recursividad, pero no hizo falta ya que eran visibles de forma clara.

DIR(program ::= part program) = CAB'(part) = {"function", "procedimiento"}  
DIR(program ::= part) = CAB'(part) = {"function", "procedimiento"}  
  
DIR(part ::= "funcion" type restpart) = {function}  
DIR(part ::= "procedimiento" restpart) = {procedimiento}  
  
DIR(restpart ::= IDENTIFICADOR "(" listparam ")" blq) = {IDENTIFICADOR}  
DIR(restpart ::= IDENTIFICADOR "(" ")" blq) = {IDENTIFICADOR}  
  
DIR(listparam ::= listparam "," type IDENTIFICADOR) = {"entero", "real", "caracter"}  
DIR(listparam ::= type IDENTIFICADOR ) = {"entero", "real", "caracter"}  
  
DIR(type ::= "entero") = {"entero"}  
DIR(type ::= "real") = {"real"}  
DIR(type ::= "caracter") = {"caracter"}  
  
DIR(blq ::= "inicio" sentlist "fin") = {"inicio"}  
  
DIR(sentlist ::= sentlist sent) = {"entero", "real", "caracter", "inicio, "return", "bifurcacion", "buclepara", "buclemientras", "bucle"}  
DIR(sentlist ::= sent) = {"entero", "real", "caracter", "inicio, "return", "bifurcacion", "buclepara", "buclemientras", "bucle"}  
  
DIR(sent ::= type lid ";") = {"entero", "real", "caracter"}  
DIR(sent ::= IDENTIFICADOR asig exp ";") = {IDENTIFICADOR}  
DIR(sent ::= "return" exp ";") = {"return"}  
DIR(sent ::= IDENTIFICADOR "(" lid ")" ";") = {IDENTIFICADOR}  
DIR(sent ::= IDENTIFICADOR "(" ")" ";") = {IDENTIFICADOR}  
DIR(sent ::= "bifurcacion" "(" lcond ")" "entonces" blq "sino" blq) = {"bifurcacion"}  
DIR(sent ::= "buclepara" "(" IDENTIFICADOR asig exp ";" lcond ";" IDENTIFICADOR asig exp ")" blq ) = {"buclepara"}  
DIR(sent ::= "buclemientras" "(" lcond ")" blq ) = {"buclemientras"}  
DIR(sent ::= "bucle" blq "hasta" "(" lcond ")") = {"bucle"}  
DIR(sent ::= blq) = {"inicio"}  
  
DIR(lid ::= IDENTIFICADOR) = {IDENTIFICADOR}  
DIR(lid ::= IDENTIFICADOR "," lid) = {IDENTIFICADOR}  
  
DIR(asig ::= "=") = {"="}  
DIR(asig ::= "+=") = {"+="}  
DIR(asig ::= "-=") = {"-="}  
DIR(asig ::= "\*=") = {"\*="}  
DIR(asig ::= "/=") = {"/="}  
  
DIR(exp ::= exp op exp) = {IDENTIFICADOR, "(", CONSTENTERO, CONSTREAL, CONSTLIT}  
DIR(exp ::= IDENTIFICADOR "(" lid ")") = {IDENTIFICADOR}  
DIR(exp ::= "(" exp ")") = {"("}  
DIR(exp ::= IDENTIFICADOR) = {IDENTIFICADOR}  
DIR(exp ::= CONSTENTERO) = {CONSTENTERO}  
DIR(exp ::= CONSTREAL) = {CONSTREAL}  
DIR(exp ::= CONSTLIT) = {CONSTLIT}  
  
DIR(op ::= "+") = {"+"}  
DIR(op ::= "-") = {"-"}  
DIR(op ::= "\*") = {"\*"}  
DIR(op ::= "/") = {"/"}  
  
DIR(lcond ::= lcond opl lcond) = {IDENTIFICADOR, "(", CONSTENTERO, CONSTREAL, CONSTLIT, "no"}  
DIR(lcond ::= lcond) = {IDENTIFICADOR, "(", CONSTENTERO, CONSTREAL, CONSTLIT, "no"}  
DIR(lcond ::= "no" cond) = {"no"}  
  
DIR(cond ::= exp opr exp) = {IDENTIFICADOR, "(", CONSTENTERO, CONSTREAL, CONSTLIT}  
DIR(cond ::= "cierto") = {"cierto"}  
DIR(cond ::= "falso") = {"falso"}  
  
DIR(opl ::= "y") = {"y"}  
DIR(opl ::= "o") = {"o"}  
  
DIR(opr::= "==") = {"=="}  
DIR(opr::= "<>") = {"<>""}  
DIR(opr::= "<") = {"<"}  
DIR(opr::= ">") = {">"}  
DIR(opr::= ">=") = {">="}  
DIR(opr::= "<=") = {"<="}