**ANOTACIONES SOBRE LA GRAMÁTICA**

La estructura general de los algoritmos con los que trabaja nuestro compilador es la siguiente:

**algoritmo** nombre\_algoritmo **;** cabecera\_algoritmo bloque\_algoritmo **falgoritmo**

Se destaca en la estructura del algoritmo la necesidad del punto y coma tras el nombre del algoritmo pero no al final del todo.

Para la declaración de tipos, constantes o variables, se requiere obligatoriamente del uso de punto y coma al finalizar la declaración, esto es:

**var**

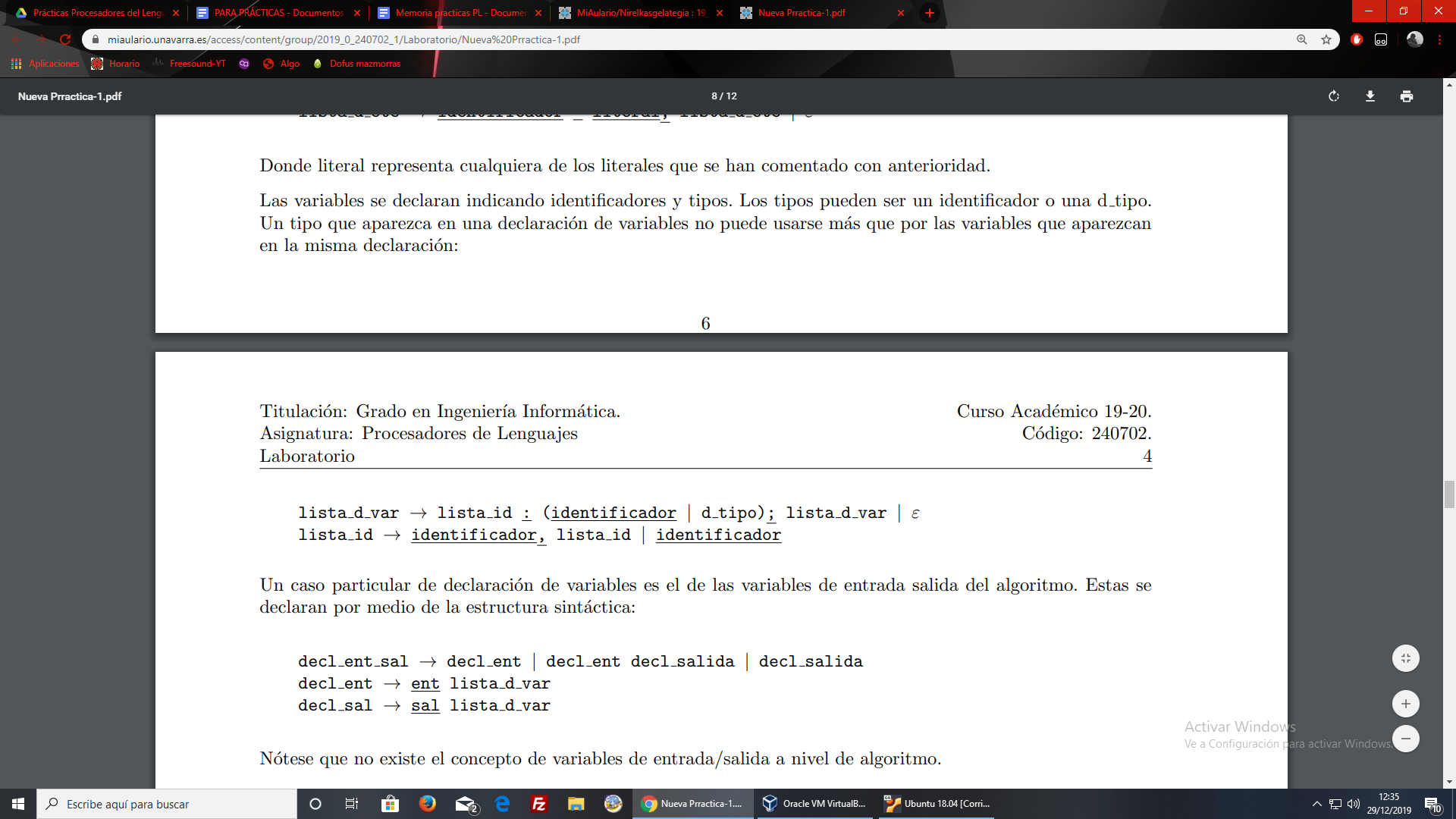
declaraciones

**fvar;**

Nuestra gramática produce

**DECLARACIÓN DE VARIABLES**

Hemos modificado ligeramente la gramática para nuestra declaración de variables. La propuesta inicial de la gramática era la siguiente



y nosotros la hemos tranformado a

lista\_d\_var → lista\_id **;** lista\_d\_var | lista\_id **;** d\_tipo **;** lista\_d\_var | Ɛ

lista\_id → **identificador ,** lista\_id | **identificador :** d\_tipo

De este modo, cuando tenemos varias variables del mismo tipo, cuando queramos indicar su tipo, esperaremos hasta ver la última de ellas y actualizaremos el tipo de todas las anteriores.

**TRATAMIENTO DE EXPRESIONES ARITMÉTICAS Y BOOLEANAS**

Debido a los problemas surgidos con los dos tipos de expresiones, hemos decidido unificar todas las expresiones, tanto aritméticas como booleanas en únicamente un tipo de expresión. El nombre que recibe es ExpArit y, para poder trabajar con las traducciones que ambas expresiones por separado requieren, utilizamos la siguiente estructura:

struct ExpArit{

int place;

int type;

struct lista\_bool \*l\_true;

struct lista\_bool \*l\_false;

}ExpArit

**CARACTERÍSTICAS RESEÑABLES**

Para facilitar todo el tratamiento de los elementos de la tabla de símbolos, ya que puede haber elementos temporales que no tienen nombre, les asignamos un nombre característico de la forma (**temp\_**numero) para facilitar mucho todo lo relacionado con la tabla de símbolos.

Si los operadores lógicos **Y** y **O** admiten operadores que no sean booleanos, la función de backpatch, que trabaja con una lista dinámica, dará fallo al intentar acceder a algo que no existe. Para evitar esto, si las expresiones que trabajan con estos operadores no son booleanas, mostraremos un error de tipos y seguiremos trabajando sin hacer el backpatch ni la asignación de $$.true y $$.false. En caso de que ambas sean booleanas, este proceso se hará sin ningún problema y continuará de forma correcta.

Debido a la alta modificación necesaria sobre la gramática y a la poca información disponible, nos hemos visto con la necesidad de finalizar el compilador con la generación de código intermedio para las expresiones booleanas.

Para las asignaciones de verdadero y falso, deberíamos pre-definirlas en la tabla de símbolos como variables temporales, pero hemos optado por definir dos operaciones de asignación propias para asignar verdadero y asignar falso, para no tener la necesidad de trabajar con estas variables temporales.

**EJEMPLOS PARA EJECUTAR Y CÓMO EJECUTARLO**

Para compilar todos nuestros ficheros y generar nuestro propio compilador, se proporciona un Makefile en la carpeta comprimida, por lo que basta con escribir en la línea de comandos *make* para generar nuestro compilador. Una vez realizado este paso, basta con ejecutar el comando *./compiler fichero\_ejemplo* donde, si el fichero\_ejemplo tiene un algoritmo escrito siguiendo la gramática de forma correcta, se mostrarán los Shift y Reduce del parser. Una vez ha finalizado la compilación del algoritmo, se muestran las tablas de símbolos y de cuádruplas, así como las constantes utilizadas en la tabla de cuádruplas para facilitar su visualización.

Se disponen de dos ejemplos disponibles con el resto de archivos: ejemplo.txt y ejemplo2.txt

**REPARTICIÓN DE LA NOTA**

Los dos miembros del grupo hemos realizado la mayor parte del trabajo durante las horas de clase, por lo que el trabajo ha ido parejo en este aspecto. Es difícil medir la cantidad de trabajo de lo que hemos hecho cada uno fuera del laboratorio. El miembro Alejandro Gabarre se encargó de la creación de la tabla de símbolos y sus funciones, mientras que Asier Ballesteros arregló los problemas surgidos con las expresiones booleanas y, durante las traducciones, facilitó información acerca de conceptos que no teníamos claros o considerábamos de forma incorrecta.

Debido a esto y de que la nota no se puede repartir de manera equitativa, hemos optado por otorgar un 51% de la nota del proyecto a Asier Ballesteros Domínguez mientras que Alejandro Gabarre González se quedaría con el 49% restante.