Práctica 2
Un analizador sintáctico LL(1) para miniAda

Objetivos

1) Desarrollar un analizador sintáctico LL(1) para miniAda
2) Integrar un analizador léxico con un analizador sintáctico
3) Usar la utilidad "make" de Unix

Contenidos

Como objetivo final, esta práctica propone el desarrollo de un analizador sintáctico LL(1) para miniAda.

Como primer paso, es necesario escribir la gramática del lenguaje. Para ello debeis tratar de "deducir" las características sintácticas del lenguaje a partir de la batería de fuentes en el directorio de salidas. En cualquier caso, las dudas sobre el lenguaje deben ser planteadas al profesor. Es aconsejable, primero, definir una gramática "intuitiva" para, en un paso posterior, tratar de transformarla en una que sea LL(1).

La gramática puede empezar como:

```
programa →
  tkPROGRAM
  tkID
  ';'
  parteDecs
  bloqueInst
  '.'

parteDecs →
| listaDecs

listaDecs →
  listaDecs dec
| dec

dec →
  decConsts
| decTipos
| decVars
| decFuncProc
```



```
decConsts →
  tkCONST listaDecCons ';'

...

decCons →
  tkID '=' valConstante
| tkID '=' tkID

...
```

Resultados

Como resultado de esta práctica hay que entregar el fichero denominado 'pract2.tar', de manera que la ejecución de la instrucción 'tar -xvf pract2.tar' genere un directorio denominado 'pract2' con los siguientes ficheros:

- 'AL_miniAda2.l' que contiene el fuente Flex del analizador léxico usado (notar que el nombre no puede coincidir con el que sometieriais para la primera práctica, por lo que ha sido cambiado).
- 'AS_miniAda.c' que contiene el fuente C con la implementación del analizador sintáctico. La implementación puede realizarse tanto por tabla como descendente recursivo.
- 'AS_miniAdaMake' que contiene el fichero Make para generar el ejecutable del analizador sintáctico, de manera que la invocación

```
make -f AS_miniAdaMake
```

genere el ejecutable 'AS_miniAda' que realiza el análisis sintáctico. La invocación al analizador debe ser como sigue (el significado del número 3 en la invocación se explicará más tarde):

```
AS_miniAda 3 nombreDelFicheroConElFuente
```

- Todos aquellos ficheros que sean necesarios (conteniendo, por ejemplo, implementaciones de tipos de datos, funciones para el tratamiento de errores, etc.).

El analizador léxico no debe dar ningún mensaje (ni realizar ninguna tarea adicional) cuando el fuente sea léxica y sintácticamente correcto.

En caso de error léxico, el comportamiento ha de ser el establecido en la primera práctica.

En caso de error sintáctico, se debe suministrar información al usuario como se muestra en el siguiente ejemplo. Considerar el programa

```
Procedure  is
  a: integer; c: integer;
Begin
   get(a);
   If a>4 Then
      c:=a;
   End If
   put("Se acabó");new_line;
End;
```

Deberá dar un mensaje de error del estilo del siguiente:

```
(1) Error:lin_1: Se esperaba un identificador
Procedure is ....;
----------^
```

La sintaxis que aquí se utiliza para la salida de errores sintácticos es:

- entre paréntesis el número identificativo que el analizador asigna al error; empieza por 1 y se incrementa correlativamente (de momento, pararemos en cuanto encontremos uno; cuando implementemos recuperación de errores, podrán detectarse varios en una misma pasada).
- a continuación "lin_#", donde "#" representa el número de la línea donde se ha detectado el error
- después información referente al error detectado. Esta información debe ser tan precisa y útil para el usuario como sea posible
- a continuación, en la línea siguiente, el texto que se lleva reconocido de la línea que se está procesando
- en la línea siguiente, un apuntador a la posición precisa donde se ha localizado el error.

Es preciso que tengais en cuenta que para que el último apartado sea viable, es necesario saber a cuántos espacios en blanco ha de equivaler un tabulador cuando se escribe el mensaje de error. Para ello, la invocación a la aplicación será como sigue:

```
AS_miniAda 3 nombreDelFicheroConElFuente
```

donde "3" indica el número de espacios a que equivale un tabulador (habitualmente, la salida estándard está configurada, por defecto, a 8 espacios en blanco por tabulador).

Notar que será necesario que hagais una nueva versión del analizador léxico para poder ir almacenando el texto de la línea que se está procesando de acuerdo con las nuevas especificaciones. Suponer para ello que ningún programador con sentido común utilizaría nunca una línea de más de 132 caracteres.

Observaciones

Para establecer la gramática de miniAda será de ayuda consultar las que aparecen en los libros de C, Ada, Pascal, etc. Suele ser especialmente crítica la parte de la gramática correspondiente a las expresiones, por lo que es importante que esta parte se trate con cuidado. La sintaxis de expresiones que se consideran válidas en miniAda es (casi) la misma que en Ada.

No construyais toda la gramática de una sola vez. Probar a hacerlo de una manera incremental. Construir primero la gramática que sólo reconozca la declararación del procedimiento y el cuerpo del mismo. Progresivamente, ir añadiendo nuevas reglas, realizando tests, y así sucesivamente. La gramática completa de miniAda, separando las reglas con líneas en blanco (por claridad), ocupa alrededor de tres páginas de fuente Yacc. Su transformación para el análisis LL(1) ocupará un poco más (pero no mucho más).

Fecha límite para entrega de resultados: