Procesadores de Lenguaje. Curso 2001-2002. Junio de 2002. Cuestiones.
No se permiten libros ni apuntes. Tiempo: 1 hora 15 min.
FINAL: Contestad a las preguntas 1,2,3,5,6.
SEGUNDO PARCIAL: Contestad a las preguntas 3,4,5,6.

1. Dada la siguiente gramática, construye la tabla de análisis sintáctico. ¿ Es una gramática LL(1)?. Justifica la respuesta. Analiza la entrada *aaaa* y comenta los problemas que encuentres.

$$A \rightarrow aAa|\epsilon$$

2. Describe brevemente cómo se implementa la recuperación en modo de pánico en los analizadores descendentes, dirigidos por tabla y predictivos recursivos.

3. ¿ Es una gramática de atributos-S siempre de atributos-L? Indica un ejemplo de una gramática que no sea de atributos-L.

4. Diseña un ETDS que traduzca consultas sencillas en SQL escritas en inglés a castellano. La gramática que genera este tipo de consultas en SQL es:

   `Consult` → `ClausulaSelect ClausulaFrom ClausulaWhere`
   `ClausulaSelect` → **select id (, id)***
   `ClausulaFrom` → **from id (, id)***
   `ClausulaWhere` → **where** `E`
   `E` → **E == E | E > E** `| id | num`
   De manera que la entrada **select** `Nombre, DNI` **from** `Estudiantes` **where** `Edad>20` se convierte en **seleccionar** `Nombre, DNI` **desde** `Estudiantes` **donde** `Edad>20`

   ¿ De qué tipo de atributo(s) se trata? Indica la estructura del árbol que representaría este tipo de programas. Implementa en pseudocódigo la función para su evaluación, indicando el código a ejecutar para los diferentes tipos de nodo.

5. Supongamos la siguiente gramática de atributos en la que tenemos un atributo heredado (h) y otro sintetizado (s).

S → E * E	$\{E_1.h = 2 * S.h;\ E_2.h = S.h;\ S.s = E_1.s * E_2.s;\ print(S.s)\ \}$
E → num	{print(E.h); E.s=lexema(num);}

   Coloca las acciones semánticas en el lugar adecuado para cada producción. ¿Qué se imprime para el caso de $S.h = 1$ y la entrada $3 * 4$? ¿ Se podría realizar la evaluación de ambos atributos en una única pasada? ¿ Cómo?.

6. Comenta qué se entiende por equivalencia estructural de tipos. Supongamos que se quiere establecer la equivalencia estructural entre expresiones de tipos `registro`. Indica como seria la función `typeEqual(t1,t2:TypeExp):Boolean;` para este tipo de expresiones. Indica un ejemplo de tipos registro estructuralmente equivalentes.

Procesadores de Lenguaje. Curso 2001-2002. Junio de 2002. Problemas.
Se permiten libros y apuntes. Tiempo: 1 hora 30 min.
FINAL: Contestad a las preguntas 1,3.
SEGUNDO PARCIAL: Contestad a las preguntas 2,3.

1. Supongamos una gramática para generar expresiones entre conjuntos:

   $$\text{Exp} \rightarrow \text{Exp} \cap \text{Exp} \mid \text{Exp} \cup \text{Exp} \mid \text{Exp} \setminus \text{Exp} \mid \text{Letra}$$
   $$\text{Letra} \rightarrow \text{A} \mid \text{B} \mid \ldots \mid \text{Z}$$

   (i) Transforma la gramática para que se verifiquen las siguientes relaciones de prioridad: las operaciones de unión e intersección entre conjuntos tienen la misma prioridad y son asociativas a la derecha; la operación de resta de conjuntos $\setminus$ tiene mayor prioridad que las anteriores y es asociativa a izquierdas. (ii) Implementa en pseudocódigo el analizador descendente recursivo. No olvides comprobar que la gramática resultante del aparatdo (i) es LL(1).

2. Supongamos que se nos pide diseñar el modulo de reconocimiento figuras en un programa de diseño gráfico por ordenador. El sistema posee un modulo de procesamiento de imágenes cuya entrada es una imagen de un plano y la salida es una serie de elementos rectilíneos y curvilíneos, que llamaremos primitivas, y que forman las figuras (en este sentido la función del modulo de análisis de imágenes puede compararse con la del analizador léxico). Las primitivas que proporciona el modulo de análisis de imágenes se muestran en la figura 1(a).

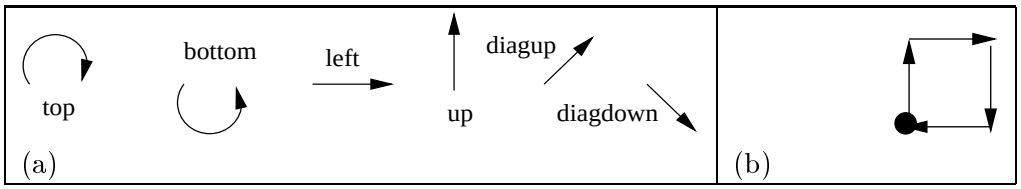

Figure 1: (a) conjunto de primitivas; (b) objeto `up+left+¬up+¬left`

Las operaciones que se pueden realizar con estas primitivas son: la suma, operador +, que permite la concatenación de dos primitivas, y la rotación, operador ¬, que permite un giro de un ángulo de 180 grados. Supongamos que estas primitivas se pueden combinar con la siguiente gramática: Se pide:

Figura →	Union
Union →	Union + Union | ¬ Union | Primitiva
Primitiva →	**up** | **left** | **top** | **bottom** | **diagup** | **diagdown**

- Construye el autómata de elementos LR(0). Justifica si se trata de una gramática SLR(1).
- Construye la tabla de análisis sintáctico. En caso de que existan conflictos, indica la acción a realizar teniendo en cuenta que el operador ¬ tiene mayor prioridad que el operador +. El operador + es asociativo por la izquierda y el ¬ por la derecha.
- Diseña un sencillo mecanismo de recuperación de errores a nivel de frase.
- Indica el contenido de la pila, la entrada y la acción a realizar para reconocer la cadena `up+diagup+diagdown+¬up+¬left`. Dibuja el árbol de análisis sintáctico. Indica de qué figura se trata. (*Pista: la has hecho decenas de veces cuando eras un niño/a :-)*).

3. Supongamos una nueva construcción de los lenguajes de programación que llamaremos **do-while-do** (hacer-mientras-hacer). Esta construcción nace de forma natural, como las construcciones **while-do, do-while** que ya conocéis. Esta construcción surge para implementar la idea en que hay casos en los que no se desea salir de la iteración al principio, ni al final de la iteración, sino a la mitad, después de que se ha hecho cierto procesamiento.

   Por ejemplo para implementar el código:

```
dowhiledo
        read(X);
        while (no_final_fichero)
              process(X);
end dowhiledo
```

   La sintaxis de esta construcción es:

$$S \rightarrow \quad \textbf{dowhiledo} \quad S* \quad \textbf{while} \quad E \quad S* \quad \textbf{end dowhiledo} \quad | \quad \epsilon$$

   donde se ha usado el operador * para indicar cero o más repeticiones. Se pide:

   - Dibujar el diagrama de flujo de esta construcción.
   - Dibujar la forma del árbol abstracto de análisis sintáctico que usarías para su traducción a código de 3-direcciones.
   - Escribir el pseudocódigo de la función `generar_código` para traducir este tipo de sentencias a una lista de cuádruplos.
   - Desafortunadamente ningún lenguaje común de programación implementa esta construcción. ¿A qué crees que se debe esto?. ¿ Qué construcción(es) usa el lenguaje C para implementar esta idea?