Universidad de Huelva

Departamento de Tecnologías de la Información

Área de Ciencias de la Computación e Inteligencia Artificial

Procesadores de lenguajes

Examen de septiembre

EJERCICIO 1 (2 puntos)

Considere la gestión de la memoria en tiempo de ejecución.

(a) ¿En que consiste la memoria de pila?

(b) Describa la estructura del registro de activación de una función.

(c) Describa el proceso de llamada a una función.

(d) Describa el proceso de retorno de una función.

EJERCICIO 2 (2 puntos)

La siguiente figura muestra una expresión regular formada por los símbolos **a**, **b** y **o**.

```
b  a  ( (b | o )*  a  a*  o)*  (b | o)*  a   a*  b
```

Obtenga el Autómata Finito Determinista asociado, indicando el conjunto de expresiones regulares punteadas que describen cada estado del autómata.

EJERCICIO 3 (2 puntos)

La siguiente gramática permite describir un circuito formado por resistencias unidas en serie o en paralelo (el token ***serie*** se representa mediante el símbolo **-**, el token ***paralelo*** mediante el símbolo | y el token ***resistencia*** mediante un identificador):

Circuito → *Circuito* **paralelo** *CircuitoSerie*

Circuito → *CircuitoSerie*

CircuitoSerie → *CircuitoSerie* **serie** *Base*

CircuitoSerie → *Base*

Base → **resistencia**

Base → **parab** *Circuito* **parce**

(a) Construya la tabla SLR de la gramática planteada. Utilice para ello la tabla incluida en la última página.

(b) Desarrolle la traza (contenido de la pila, estado del flujo de entrada y acción a realizar en cada paso) del analizador SLR para la siguiente cadena:

```
R1 - ( R2 | R3 )
```

EJERCICIO 4 (2 puntos)

La siguiente gramática describe expresiones formadas con las operaciones producto y potencia de números:

Expresión → *Factor* *Producto*

Producto → **prod** *Factor* *Producto*

Producto → λ

Factor → *Base* *Potencia*

Potencia → **elev** *Base* *Potencia*

Producto → λ

Base → **num**

Base → **lparen** *Expresión* **rparen**

Considere las siguientes clases que permiten definir expresiones con estos operadores:

```
// Clase abstracta que describe una expresión aritmética
public abstract class  Expression {
}

// Clase que describe un número constante
public class Number extends Expression {
  private double value;
  public Number(double val) {  this.value = val; }
}

// Clase que describe la potencia entre dos expresiones
public class Power extends Expression {
  public Expression base;
  public Expression pow;
  public Power(Expression a, Expression b) {  this.base = a;   this.pow = b;  }
}

// Clase que describe el producto entre dos expresiones
public class Product extends Expression {
  public Expression left;
  public Expression right;
  public Product(Expression a, Expression b) {  this.left = a; this.right = b;  }
}
```

(a) Calcule los conjuntos Primeros, Siguientes y Predicción de la gramática.

(b) Construya la tabla de análisis sintáctico LL(1) para la gramática.

(c) Desarrolle un ETDS que genere el árbol de sintaxis abstracta asociado a una expresión formada por los operadores producto y potencia. Es importante tener en cuenta que la potencia es un operador asociativo a la derecha, mientras que el producto es un operador asociativo a la izquierda.

EJERCICIO 5 (2 puntos)

Los lenguajes C, C++ y Java disponen de un operador ternario (conocido en inglés como *hook* por la interrogación) formado por una condición y dos expresiones. Cuando la condición es cierta, se evalúa la primera expresión y se devuelve su resultado. Cuando la condición resulta falsa se evalúa la segunda expresión y se devuleve su resultado. A continuación se describe la gramática del operador en formato JavaCC.

```
void Hook() :
{}
{
   <LPAREN>  Cond()  <HOOK>
             Expr()  <COLON>
             Expr()  <RPAREN>
}
```

Se pide:

(a) Describir la estructura del código intermedio asociado a este operador.

(b) Desarrollar el ETDS que genere el código intermedio asociado al operador ternario.

Para ello se deberán tener en cuenta las siguientes consideraciones:

- Se dispone del método *getNewLabel()* para generar nuevas etiquetas.
- Se dispone del método *getNewTemp()* para generar nuevas variables temporales.
- El símbolo *Cond()* devuelve un objeto de la clase *Condition* con los siguientes campos: *code*, que contine el código intermedio de la condición; *label_true*, que contiene el nombre de la etiqueta a la que se salta cuando la condición es cierta; *label_false*, que contiene el nombre de la etiqueta a la que se salta cuando la condición es falsa.
- El símbolo *Expr()* devuelve un objeto de la clase *Expression* con los siguientes campos: *code*, que contiene el código intermedio para evaluar la expresión; *temp*, que contiene el nombre de la variable en el que se almacena el resultado de la expresión.
- El símbolo *Hook()* deberá devolver un objeto de tipo *Expression*.

Tabla del ejercicio 3

Estado	resistencia	parab	parce	serie	paralelo	$	Circuito	CircuitoSerie	Base