Universidad de Huelva

Departamento de Tecnologías de la Información

Área de Ciencias de la Computación e Inteligencia Artificial

Procesadores de lenguajes

Examen de diciembre

EJERCICIO 1 (2 puntos)

Considere un esquema de gestión de memoria dinámica basado en la estructura de bloques marcados.

(a) Describa la estructura de cada bloque.

(b) Describa el proceso de alojamiento.

(c) Describa el proceso de desalojo.

EJERCICIO 2 (1.5 puntos)

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

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

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

EJERCICIO 3 (2.5 puntos)

Dada la siguiente gramática:

Stm → *Call*

Stm → *Assign*

Call → **id**

Assign → *Var* **eq** *Exp*

Var → *Var* **lparen** *Exp* **rparen**

Var → **id**

Exp → *Var*

Exp → **num**

(a) Construya la tabla de análisis SLR

(b) Construya la tabla de análisis LR(1)

EJERCICIO 4 (2 puntos)

La siguiente figura muestra una gramática LL(1) que describe el operador potencia.

Factor → **num** *Potencia*

Potencia → **elev** **num** *Potencia*

Potencia → λ

La potencia es un operador binario que tiene la particularidad de considerarse asociativo a la derecha. Esto quiere decir que una expresión del tipo

a ^ b ^ c

debe entenderse como

a ^ (b ^ c)

al contrario de lo que ocurre con la mayoría de operadores binarios como la suma, el producto, etcétera.

Considere las siguientes clases que permiten definir las potencias de números:

```
// 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 {
  private Expression base;
  private Expression pow;

  public Power(Expression a, Expression b) {
    this.base = a;
    this.pow = b;
  }
}
```

Desarrolle un ETDS que genere el árbol de sintaxis abstracta asociado a una expresión formada por el operador potencia.

EJERCICIO 5 (2 puntos)

La siguiente figura muestra la descripción sintáctica de la instrucción SWITCH (con el significado habitual en C, C++ y Java) en el formato de la herramienta JavaCC.

```
void InstSwitch() :
{}
{
  <SWITCH> <PARAB> Expresion() <PARCE> <LLAVEAB>
  ( Clausula() )*  <LLAVECE>
}

void Clausula() :
{}
{
   <CASE> Value() <DP> ( Instruccion() )*
 | <DEFAULT> <DP> ( Instruccion() )*
}
```

(a) Describa la estructura del código intermedio asociado a esta instrucción

(b) Modifique la gramática para que la función *InstSwitch()* devuelva un objeto *Inst* cuyo campo *code* contenga dicho código.

Tenga en cuenta las siguientes consideraciones:

(a) La función *Expresión()* devuelve un objeto de la clase *Expr* con los campos *code*, que contiene el código intermedio que describe la expresión, y *temp*, con la referencia a la variable en que se almacena el valor de la expresión.

(b) La función *Instruccion()* devuelve un objeto de la clase *Inst* que contiene el campo *code* con el código asociado a la instrucción.

(c) La función *Value()* devuelve un entero (int) con el valor de la constante que aparece en la cláusula *case*.

(d) Se dispone del método *getNewLabel()* que devuelve una nueva etiqueta (es decir, no utilizada en ningún punto del código)

(e) Se dispone del método *getNewTemp()* que devuelve una referencia a una nueva variable temporal.

(f) Se dispone del método *pushBreakLabel(String)* que apila la etiqueta a la que deben saltar las instrucciones break.

(g) Se dispone del método *popBreakLabel()* que desapila la etiqueta a la que deben saltar las instrucciones break.

(h) Se dispone de la clase *CaseDefaultClause* con el siguiente código:

```
public class CaseDefaultClause {

  public boolean isDefault;

  public int caseValue;

  public Inst[] instBlock;

  public CaseDefaultClause(int caseValue) { ... } // Cláusula case

  public CaseDefaultClause() { ... } // Cláusula default

  public void addInstruction(Inst inst) { ... } // Añade una instrución
}
```

(i) Las instrucciones de salto (condicionales e incondicionales) del código de 3 direcciones son las siguientes:

```
if var1 == var2 goto etiqueta
if var1 != var2 goto etiqueta
if var1 > var2 goto etiqueta
if var1 < var2 goto etiqueta
if var1 >= var2 goto etiqueta
if var1 <= var2 goto etiqueta
goto etiqueta
```