CC41A Lenguajes de Programación
Control 1 - Otoño 2005
Profesor: Luis Mateu

Pregunta 1

La sintaxis de una s-expresión en el lenguaje Scheme es muy simple. Una s-expresión puede ser (i) un átomo, o (ii) una lista de 0 o más s-expresiones encerradas entre paréntesis. Un átomo puede ser (i') un número, (i") un string encerrado entre doble comillas, o (i'") un símbolo. Un símbolo es cualquier otra cosa que no contenga espacios en blanco, paréntesis o doble comillas y que no corresponda a un número. Los siguientes son ejemplos de s-expresiones válidas:

- 3 números (c/u es una s-expresión, pero en conjunto no lo son): 0 123 -3
- 4 strings: "hola que tal" "123" "+" "("
- 5 símbolos: abc 42abc + / *valor*
- una lista con 3 s-expresiones: (-3 abc "hola")
- una lista con 0 s-expresiones: ( )
- una lista con 3 s-expresiones: ((0 1 2 ( )) abc ((1)))
- una lista con 3 s-expresiones: (ab(c de)c)
  que equivale a: (ab (c de) c)

En cambio, los siguientes ejemplos no son s-expresiones válidas:

- a b porque hay dos átomos, pero no es una lista.
- (a b porque falta un paréntesis derecho
- (a ("hola)) porque falta cerrar el string
- ((a))) (((c)) porque los paréntesis no están bien balanceados

(a) Escriba un analizador léxico que distinga como tokens: paréntesis izquierdo o derecho, números, strings o símbolos.

(b) Escriba un analizador sintáctico para s-expresiones utilizando CUP. No necesita colocar acciones en la gramática.

Indicaciones: Hay muchas soluciones para este problema. Lo importante es que su solución tiene que aceptar los ejemplos de s-expresiones válidas de este enunciado y rechazar los ejemplos no válidos. Olvídese de los *tabs*, *newlines* y caracteres que no aparecen en los ejemplos. Observe que en JLex una expresión regular como `[^abc]` acepta cualquier caracter que no sea a, b o c. Además \" reconoce el caracter ".

Pregunta 2

(a) Para el siguiente programa en C, haga un diagrama de variables y objetos después de ejecutar la última instrucción de `main`, indicando para cada variable u objeto su nombre (si lo hay), localidad (espacio en memoria), tipo y contenido (con sus campos).

```
struct L { double x; struct L *n; } *l;
int main() {
  l= (struct L*) malloc(3*sizeof(*l));
  /* equivalente a: malloc(3*sizeof(struct L)); */
  l->x= 1.2;  l->n= l+2;
  l= l+2;
  l->x= 2.4;  l->n= l-1;
}
```

(b) Dadas las siguientes declaraciones de variables en Java:

```
int i; byte b; double d; String s; boolean t;
```

Haga árboles de sintaxis abstracta para las siguientes expresiones, etiquetando todos los nodos con el tipo de la sub-expresión a la cual se encuentra asociado:

- b + i + d + s
- b+b == 2 ? i+s : (Object)new Double(d)
- (t= true) && s.equals(new Integer(i))

(c) El siguiente programa está escrito en un lenguaje hipotético en donde el paso de parámetros es por nombre:

```
double prod(int i, int ini, int fin, double exp) {
  double p= 1;
  for (i= ini; i<=fin; i++)
    p= p * exp;
}
void main() {
  int v;
  print prod(v, 1, 10, ((double)v)*v);
}
```

Aplique la regla de la copia para explicar el comportamiento de la invocación del procedimiento `prod`.

(d) Dada las siguientes declaraciones de clases y variables:

`class A {`
`  void m(A a, Object o) ... //(1)`
`  void m(B b, A a) ...  //(2)`
`}`
	`class B extends A {`
	
	`  void(B b, A a) ... //(3)`
	`  void(B b, B b) ... //(4)`
	`}`

`A aa= new A(), ab= new B();`	`B bb= new B()`

Para cada una de las siguientes invocaciones indique número de método que (i) el compilador selecciona estáticamente para ser invocado y (ii) el que se invoca en tiempo de ejecución:

- aa.m(bb, aa); ab.m(bb, aa);
- aa.m(ab, bb); ab.m(ab, bb);
- ab.m(bb, bb); bb.m(bb, bb);