

# Compiladores: Examen final de laboratorio.

8 de junio de 2021

**ATENCIÓN:** En el Racó encontraréis los juegos de pruebas y el código necesario para realizar el examen. El paquete contiene versiones modificadas del módulo *common/SemErrors.\** con los métodos requeridos para producir los nuevos errores que aparecen en los ejercicios del examen. **ANTES DE EMPEZAR, leed *atentamente* las instrucciones al final de este enunciado para descargar e instalar correctamente los ficheros del examen.**

**ATENCIÓN:** Hay que entregar el examen en un fichero *.tgz* subido al Racó. **Leed las instrucciones al final de este enunciado para ver como generarlo.**

**PUNTUACIÓN:** Los tres primeros puntos de la nota se obtienen con los juegos de pruebas de la práctica base. El resto se obtienen superando los juegos de pruebas específicos del examen. La corrección es **automática**, a través de los juegos de pruebas de este enunciado, más un conjunto adicional de juegos de pruebas privados.

**IMPORTANTE:** El examen consta de dos ejercicios independientes. Podéis hacerlos en cualquier orden. Se recomienda hacer cada ejercicio incrementalmente, resolviendo cada JP antes de pasar al siguiente.

## 1 Operación potencia (3 puntos)

Queremos ampliar ASL con la operación de elevar un número a una potencia entera, pudiendo hacer programas como el siguiente:

```
1 func main()
2   var a,b: int
3   var A: array[2] of float
4   var x,y: float
5
6   x = 3;
7   a = 5;
8   read b;
9   y = x**a;
10  if ( a**b - 2 > (y/2)**a ) then
11    A[0] = A[1]**(b+3);
12  endif
13 endfunc
```

Usaremos el operador **\*\*** para denotar la potencia. Tiene las siguientes propiedades:

- Es un operador infijo (**base\*\*exponente**).
- Tiene la máxima prioridad, superior a cualquier otro operador.
- Es asociativo por la izquierda (como el resto de operadores).

- La base puede ser entera o real.
- Solo se aceptan exponentes enteros. Adicionalmente, para este examen, no consideramos exponentes negativos.
- El resultado siempre es real (aunque la base fuera entera).

**Juego de pruebas 1 (0.5 puntos).** Empezaremos modificando solo la gramática para añadir el operador **\*\*** a la subgramática de las expresiones.

Una vez modificada la gramática, el primer juego de pruebas

```

1 func main()
2   var a,b: int
3   var A: array[2] of float
4   var x,y: float
5   var p : char
6
7   x = 3;
8   c = 5;
9   read b;
10  y = x**a;
11  if a**b-2 > (y/2)**a then
12    A[0] = A[1]**(b+3);
13    if x>p or not y then
14      p = a+1;
15    endif
16    a[1] = A+1;
17  endif
18 endfunc

```

debe producir la salida:

```

Line 8:2 error: Identifier 'c' is undeclared.
Line 13:9 error: Operator '>' with incompatible types.
Line 13:15 error: Operator 'not' with incompatible types.
Line 14:9 error: Assignment with incompatible types.
Line 16:5 error: Array access to a non array operand.
Line 16:13 error: Operator '+' with incompatible types.

```

**Juego de pruebas 2 (0.5 puntos).** A continuación comprobaremos que los operandos son del tipo correcto: la base debe ser numérica y el exponente entero. Se utilizará un mensaje de error similar al del resto de operadores.

Asi superaremos el segundo juego de pruebas:

```

1 func main()
2   var a,b: int
3   var A: array[2] of float
4   var x,y: float
5   var p : char
6   var n : bool
7
8   x = 3;
9   c = 5;
10  read b;
11  y = x**a;
12  y = a**x;
13  while a**b == 2 do
14    A[0] = A[1]**(b+3);
15    if x>p or not y then
16      p = a+1;
17    endif
18    x = a**p + 1;
19    y = y**p - 1;
20  endwhile
21
22  y = p**a;
23  x = p**n;
24 endfunc

```

que debe producir los errores:

```

Line 9:2 error: Identifier 'c' is undeclared.
Line 12:7 error: Operator '**' with incompatible types.
Line 15:9 error: Operator '>' with incompatible types.
Line 15:15 error: Operator 'not' with incompatible types.
Line 16:9 error: Assignment with incompatible types.
Line 18:10 error: Operator '**' with incompatible types.
Line 19:10 error: Operator '**' with incompatible types.
Line 22:7 error: Operator '**' with incompatible types.
Line 23:7 error: Operator '**' with incompatible types.

```

**Juego de pruebas 3 (0.5 puntos).** El siguiente juego de pruebas comprueba que el resultado de la potencia sea un real.

Una vez hechos los cambios necesarios, el juego de pruebas:

```

1 func f(r:float, b:bool) : char
2   if r > 10 and b then
3     return 'a';
4   else
5     if r < 5 then return 'B';
6     else return '@';
7   endif
8   endif
9 endfunc
10
11 func main()
12   var a,b: int
13   var A: array[2] of float
14   var x,y: float
15   var p : char
16   var n : bool
17
18   x = 3;
19   y = x**a;
20   b = a**x;
21   p = (a+2)**(x-1);
22   while a**b == 2 do
23     A[0] = A[1]**(b+3);
24     A[1] = a**p + y;
25     a = y**p - 1;
26   endwhile
27
28   write f(p**n);
29 endfunc

```

debería producir la salida:

Line 20:4 error: Assignment with incompatible types.  
 Line 20:7 error: Operator '\*\*' with incompatible types.  
 Line 21:4 error: Assignment with incompatible types.  
 Line 21:11 error: Operator '\*\*' with incompatible types.  
 Line 24:13 error: Operator '\*\*' with incompatible types.  
 Line 25:7 error: Assignment with incompatible types.  
 Line 25:10 error: Operator '\*\*' with incompatible types.  
 Line 28:8 error: The number of parameters in the call to 'f' does not match.  
 Line 28:11 error: Operator '\*\*' with incompatible types.

**Juego de pruebas 4 (0.5 puntos).** A continuación empezaremos a generar el código que calcule potencias, que lógicamente debe contener un bucle. Por el momento, sólo trataremos bases reales.

Con esto, el siguiente programa:

```

1 func main()
2   var a,b: int
3   var x,y: float
4
5   x = 3;
6   a = 2;
7   b = 6;
8   y = x**a;
9   write y*2; write "\n";
10
11   while x**b > 3 do
12     write x;
13     write "***";
14     write b;
15     write "=";
16     write x**b; write "\n";
17     b = b-1;
18   endwhile
19
20   write 3 + (y-2.5)**(b+4) - x**(b-1); write "\n";
21 endfunc

```

debe escribir:

```

18
3**6=729
3**5=243
3**4=81
3**3=27
3**2=9
11604.9

```

**Juego de pruebas 5 (0.5 puntos).** Seguidamente, trataremos el caso en que la base es entera. Recordad que el resultado de la potencia sera igualmente un real.

Así pasaremos el juego de pruebas:

```

1 func f(r:float, b:bool) : char
2   if r > 10 and b then
3     return 'R';
4   else
5     if r< 5 then return '$';
6     else return 'k';
7   endif
8   endif
9 endfunc
10
11 func main()
12   var a,b: int
13   var x,y: float
14
15   x = 3;
16   a = 2;
17   b = 6;
18   y = a**a;
19   write y*2; write "\n";
20
21   while a**b > 3 do
22     write a; write "**"; write b; write "=";
23     write a**b; write "\n";
24     b = b-1;
25   endwhile
26   write f(3 + 2**(b+4) - x**(b-1), false); write "\n";
27 endfunc

```

que produce la salida:

```

8
2**6=64
2**5=32
2**4=16
2**3=8
2**2=4
k

```

**Juego de pruebas 6 (0.5 puntos).** El último juego de pruebas comprueba que las prioridades y asociatividades de la potencia sean correctas. Si habéis hecho bien los anteriores es probable que lo paséis directamente.

El último juego de pruebas es:

```

1 func main()
2   var a,b: int
3   var x,y: float
4
5   x = 3;
6   a = 2;
7   b = 6;
8   y = 2**3**4;
9   write y; write "\n";
10
11   y = 2/2.1**3**4;
12   write y; write "\n";
13
14   y = (2/2.1)**3**4;
15   write y; write "\n";
16
17   while a**b > 10 do
18     write 2*a**b/5; write "\n";
19     write (2*a)**b/5; write "\n";
20     write 2*a**(b/5); write "\n";
21     write 2*(a**b)/5; write "\n";
22     write (2*a)**(b/5); write "\n";
23     b = b-1;
24   endwhile
25   write 2**(b+4/a)**(b-1); write "\n";
26 endfunc

```

y escribe la salida:

```

4096
0.000271893
0.556838
25.6
819.2
4
25.6
4
12.8
204.8
4
12.8
4
6.4
51.2
2
6.4
1
1024

```

## 2 Instrucción map (4 puntos)

El segundo ejercicio consiste en dotar al language ASL de la instrucción `map`, que permite aplicar una función a todos los elementos de un vector, obteniendo otro vector con los resultados. Así, podríamos escribir, por ejemplo:

```
1 func vowelnum(c: char) : int
2   if c=='a' then return 1; endif
3   if c=='e' then return 2; endif
4   if c=='i' then return 3; endif
5   if c=='o' then return 4; endif
6   if c=='u' then return 5; endif
7   return 0;
8 endfunc
9
10 func main()
11   var word : array[5] of char
12   var nums : array[5] of int
13   var i : int
14
15   word[0] = 'h';
16   word[1] = 'o';
17   word[2] = 'l';
18   word[3] = 'a';
19   word[4] = '.';
20
21   map word into nums using vowelnum;
22
23   i=0;
24   while i<5 do
25     write nums[i];
26     write " ";
27     i = i+1;
28   endwhile
29   /// will print: 0 4 0 1 0
30 endfunc
```

La sintaxis de la instrucción es: `map A into B using F`, donde `A` y `B` son vectores, y `F` es una función. Además, deben cumplirse las siguientes condiciones:

- `A` y `B` tienen el mismo tamaño.
- `F` tiene un único parámetro.
- El parámetro de `F` es del mismo tipo que los elementos de `A`.
- El resultado de `F` es del mismo tipo que los elementos de `B`.

Cuando una o más de estas condiciones no se cumplen se emitirá un único mensaje d'error. Se trata d'un nuevo mensaje definido en el módulo *SemErrors*.

**Juego de pruebas 7 (0.5 puntos).** El primer juego de pruebas de este ejercicio simplemente comprueba que la gramática reconozca la instrucción `map`.

El primer juego de pruebas:

```
1 func vowelnum(c: char) : int
2   if c=='a' then return 1; endif
3   if c=='e' then return 2; endif
4   if c=='i' then return 3; endif
5   if c=='o' then return 4; endif
6   if c=='u' then return 5; endif
7   return 0;
8 endfunc
9
10 func sq(x: int) : int
11   return x*x;
12 endfunc
13
14 func main()
15   var word : array[5] of char
16   var nums : array[5] of int
17   var i : int
18
19   if nums!=word then
20     word[0] = i<10;
21   endif
22
23   map word into nums using vowelnum;
24   map nums into nums using sq;
25
26   i=0;
27   while nums[i]<5 do
28     write nums[i];
29     i = j+1;
30   endwhile
31 endfunc
```

genera los errores:

```
Line 4:8 error: Identifier 'i' is undeclared.
Line 19:9 error: Operator '!=' with incompatible types.
Line 20:12 error: Assignment with incompatible types.
Line 29:8 error: Identifier 'j' is undeclared.
```

**Juego de pruebas 8 (0.5 puntos).** Para superar el segundo juego de pruebas, deberéis comprobar que los dos primeros argumentos de `map A into B using F` (es decir, A y B) son vectores, y que tienen el mismo tamaño.

Con esto, el programa siguiente:

```
1 func islower(a : char) : bool
2   return a>='a' and a<='z';
3 endfunc
4
5 func f(x: int) : int
6   return x-1;
7 endfunc
8
9 func main()
10  var a : array [5] of char
11  var b : array [5] of bool
12  var c : array [5] of int
13  var x : array [10] of char
14  var y : array [10] of int
15  var z : array [10] of bool
16  var n,m : int
17  var p,q : char
18
19  while i<y[3] do
20    y[3] = y + 1;
21
22    map a into b using islower;
23    map x into n using islower;
24  endwhile
25
26  if not b[2] or x[0] then
27    map p into y using f;
28    map n into m using f;
29  endif
30
31  map a into z using islower;
32  map y into c using f;
33  map c into c using f;
34 endfunc
```

debe dar la salida:

```
Line 19:8 error: Identifier 'i' is undeclared.
Line 20:14 error: Operator '+' with incompatible types.
Line 23:5 error: Instruction 'map' with incompatible arguments.
Line 26:14 error: Operator 'or' with incompatible types.
Line 27:5 error: Instruction 'map' with incompatible arguments.
Line 28:5 error: Instruction 'map' with incompatible arguments.
Line 31:2 error: Instruction 'map' with incompatible arguments.
Line 32:2 error: Instruction 'map' with incompatible arguments.
```

**Juego de pruebas 9 (0.5 puntos).** A continuación comprobaremos que el tercer argumento F es una función, que el tipo de su (único) parámetro coincide con el de los elementos de A, y que el tipo de su resultado coincide con el de los elementos de B.

Este juego de pruebas:

```

1 func islower(a : char) : bool
2   return a>='a' and a<='z';
3 endfunc
4
5 func f(x: int) : int
6   return x-1;
7 endfunc
8
9 func toreal(x: int) : float
10  return x*1.0;
11 endfunc
12
13 func main()
14   var a : array [5] of char
15   var b : array [5] of bool
16   var c : array [5] of int
17   var d : array [5] of float
18   var x : array [10] of char
19   var y : array [10] of int
20   var z : array [10] of bool
21   var n,m : int
22   var p,q : char
23
24   while i<y[3] do
25     y[3] = y + 1;
26
27     map a into b using m;
28     map x into y using islower;
29   endwhile
30
31   map c into d using toreal;
32   map d into d using toreal;
33   map c into c using toreal;
34
35   if not b[2] or x[0] then
36     map p into y using c;
37     map b into c using f;
38   endif
39
40   map a into z using islower;
41   map y into z using f;
42 endfunc

```

debe generar los errores:

```

Line 24:8 error: Identifier 'i' is undeclared.
Line 25:14 error: Operator '+' with incompatible types.
Line 27:5 error: Instruction 'map' with incompatible arguments.
Line 28:5 error: Instruction 'map' with incompatible arguments.
Line 32:2 error: Instruction 'map' with incompatible arguments.
Line 33:2 error: Instruction 'map' with incompatible arguments.
Line 35:14 error: Operator 'or' with incompatible types.
Line 36:5 error: Instruction 'map' with incompatible arguments.
Line 37:5 error: Instruction 'map' with incompatible arguments.
Line 40:2 error: Instruction 'map' with incompatible arguments.
Line 41:2 error: Instruction 'map' with incompatible arguments.

```



**Juego de pruebas 10 (0.5 puntos).** Seguidamente, generaremos código para la instrucción `map`. En este ejemplo, la función `F` devuelve el valor constante 3, que no coincide con el que ya contenía `B` (se recomienda implementar una solución general y no específica para este caso).

<p>La ejecución del programa:</p> <pre> 1 func tres(a: int) : int 2   return 3; 3 endfunc 4 5 func dump(a : array [10] of int) 6   var i: int 7   i=0; 8   while i&lt;10 do 9     write a[i]; write " "; 10    i = i+1; 11  endwhile 12  write "\n"; 13 endfunc 14 15 func main() 16   var a,b: array [10] of int 17   var i : int 18   i=0; 19   while i&lt;10 do 20     a[i] = 3; 21     b[i] = 77; 22     i = i+1; 23   endwhile 24   map a into b using tres; 25   dump(b); 26 endfunc </pre>	<p>producirá la salida:</p> <pre> 3 3 3 3 3 3 3 3 3 3 </pre>
---	--

**Juego de pruebas 11 (0.5 puntos).** En este ejemplo, el vector resultado tomará los mismos valores que el de entrada: la función `F` es la identidad. Si habéis programado una solución general en el caso anterior, pasaréis este juego de pruebas directamente.

<p>La ejecución del programa:</p> <pre> 1 func same(a: int) : int 2   return a; 3 endfunc 4 5 func dump(a : array [10] of int) 6   var i: int 7   i=0; 8   while i&lt;10 do 9     write a[i]; write " "; 10    i = i+1; 11  endwhile 12  write "\n"; 13 endfunc 14 15 func main() 16   var a : array [10] of int 17   var i : int 18   i=0; 19   while i&lt;10 do 20     a[i] = 10*i; 21     i = i+1; 22   endwhile 23   map a into a using same; 24   dump(a); 25 endfunc </pre>	<p>producirá la salida:</p> <pre> 0 10 20 30 40 50 60 70 80 90 </pre>
---	---

**Juego de pruebas 12 (0.5 puntos).** En el siguiente ejemplo, la función **F** puede ser cualquier función. Si habéis programado una solución general en los ejemplos anteriores, pasaréis este juego de pruebas directamente.

<p>La ejecución del programa:</p> <pre>1 func calc(a: int) : float 2   var x: float 3   x = 1 - 3*a/0.75; 4   return 1/x; 5 endfunc 6 7 func dump(a : array [5] of float) 8   var i: int 9   i=0; 10  while i&lt;5 do 11    write a[i]; 12    write "\n"; 13    i = i+1; 14  endwhile 15 endfunc 16 17 func main() 18   var a : array [5] of int 19   var c : array [5] of float 20   var i : int 21 22   i=0; 23   while i&lt;5 do 24     a[i] = 10*i; c[i] = 77; 25     i = i+1; 26   endwhile 27 28   map a into c using calc; 29   dump(c); 30 endfunc</pre>	<p>producirá la salida:</p> <pre>1 -0.025641 -0.0126582 -0.00840336 -0.00628931</pre>
--	---

**Juego de pruebas 13 (1 punto).** El último juego de pruebas admite coerciones de entero a real para los argumentos y resultados de la función. Deberéis modificar el typecheck para admitir estos casos, y generar las instrucciones de coerción en el código donde sea necesario.

La ejecución del programa:

```

1 func dumpi(a : array [10] of int)
2   var i: int
3   i=0;
4   while i<10 do
5     write a[i]; write " "; i = i+1;
6   endwhile
7   write "\n";
8 endfunc
9
10 func dumpf(a : array [10] of float)
11   var i: int
12   i=0;
13   while i<10 do
14     write a[i]; write " "; i = i+1;
15   endwhile
16   write "\n";
17 endfunc
18
19 func toreal(x: int) : float
20   return x*1.3;
21 endfunc
22
23 func getint(x: float) : int
24   if x<10 then return 10; endif
25   if x<20 then return 20; endif
26   if x<50 then return 50; endif
27   if x<100 then return 100; endif
28   return 500;
29 endfunc
30
31 func main()
32   var a : array [10] of int
33   var c : array [10] of float
34   var i : int
35
36   i=0;
37   while i<10 do
38     a[i] = 10*i; c[i] = 77; i = i+1;
39   endwhile
40
41   write "a: "; dumpi(a);
42   map a into c using toreal;
43   write "c: "; dumpf(c);
44   map c into a using getint;
45   write "a: "; dumpi(a);
46
47   map a into c using getint;
48   write "c: "; dumpf(c);
49 endfunc

```

producirá la salida:

```

a: 0 10 20 30 40 50 60 70 80 90
c: 0 13 26 39 52 65 78 91 104 117
a: 10 20 50 50 100 100 100 100 500 500
c: 20 50 100 100 500 500 500 500 500 500

```

## Información importante

**FICHEROS PARA EL EXAMEN:** En el Racó ([examens.fib.upc.edu](http://examens.fib.upc.edu)) encontraréis un fichero `examen.tgz` con el contenido siguiente:

- `final-lab-CL-2021.pdf`: Este documento, con el enunciado y las instrucciones.
- `jps`: Subdirectorio con juegos de pruebas (`jp_chkt_XX.asl`) y `jp_genc_YY.asl`), y su correspondiente salida esperada (`jp_chkt_XX.err`) para los juegos de pruebas de validación semántica, y `jp_genc_YY.in/.out` para los juegos de pruebas de generación de código. En los JPs de generación, no se compara el código generado, sino la salida que produce la tVM al ejecutarlo.
- `common`: Subdirectorio con el módulo auxiliar `SemErrors` ampliado con los errores necesarios para el examen.
- `avalua.sh`: Script que ejecuta todos los juegos de pruebas y dice si se superan o no.
- `empaqueta.sh`: Script que crea un fichero `examen-nombre.apellido.tgz` con vuestra solución. Este es el fichero que debe subirse al Racó.

### PASOS A SEGUIR:

- Haced una **copia** de las carpetas `asl` y `common` de vuestra práctica a un nuevo directorio `examen`.

```
mkdir examen
cp -r practica/asl practica/common examen/
```

- Cambiad al nuevo directorio `examen`, y descomprimid el fichero `examen.tgz` del Racó:

```
cd examen
tar -xzf examen.tgz
```

Esto extraerá el contenido del paquete, **añadiendo** a vuestro directorio `examen` los ficheros mencionados anteriormente.

**IMPORTANTE:** Seguid los pasos anteriores en el orden especificado (primero una copia de la práctica y después descomprimir el `.tgz`). Hacerlo en el orden inverso causará que os falte código y juegos de pruebas necesarios para el examen.

- Trabajad normalmente en la carpeta `examen/asl`.

```
cd asl
make antlr
make
...
```

(También podéis usar el script `fast-make.sh` que compila en un directorio temporal local.)

- Para ver las diferencias entre la salida de vuestro `asl` y la salida esperada de un juego de pruebas concreto de type check, podéis hacer:

```
./asl ../jps/jp_chkt_XX.asl | diff -y - ../jps/jp_chkt_XX.err
```

(Podéis ignorar la línea “There are semantic errors: no code generated” que genera el main)
- Para ver las diferencias entre la salida de vuestro `asl` y la salida esperada de un juego de pruebas concreto de generación de código, podéis hacer:

```
./asl ../jps/jp_genc_XX.asl > jp_XX.t
../tvm/tvm jp_XX.t < ../jps/jp_genc_XX.in | diff -y - ../jps/jp_genc_XX.out
```
- Para ejecutar todos los juegos de pruebas y ver si los superáis, ejecutad `../avalua.sh`.

- Ejecutad `../empaqueta.sh` para crear el fichero de entrega `../examen-nombre.apellido.tgz` para subir al Racó. Los paquetes creados sin usar este script serán calificados com **NO PRESENTADO**.