

Compiladors: Examen final de laboratori.

5 de juny de 2020

ATENCIÓ: Al Racó trobareu els jocs de proves i codi necessari per a fer l'examen.
ABANS DE COMENÇAR A FER RES, llegiu les instruccions del final de l'enunciat per veure com descarregar-lo i instal·lar-lo.

ATENCIÓ: Cal entregar l'examen en un fitxer **.tgz** pujat al Racó. *Llegiu les instruccions del final de l'enunciat per veure com generar-lo.*

PUNTUACIÓ: Els tres primers punts de la nota de laboratori s'obtenen amb els jocs de proves de la pràctica base. La resta s'obtenen superant els jocs de proves específics de l'examen. La correcció és **automàtica**, a través dels jocs de proves d'aquest enunciat, més un conjunt addicional de jocs de proves privats.

IMPORTANT: L'examen consta de dos exercicis independents. Podeu fer-los en qualsevol ordre. Es recomana fer cada exercici incrementalment, resolent cada joc de proves abans de passar al següent.

1 Funció predefinida sum (**3.5** punts)

Volem afegir a l'ASL l'operació **sum**, que calcula la suma dels operands que rep, amb les següents característiques:

- L'operació es pot aplicar a qualsevol nombre de d'operands.
- Els operands han de ser de tipus numèric, **int** o **float**. Poden estar barrejats.
- El resultat sera **float** si algun dels operands ho és, i **int** en *qualsevol altre cas*.
- La suma de zero paràmetres dóna com a resultat zero.

Per exemple:

```
1 func main()
2   var n,m,a : int
3   var x,y : float
4
5   n = sum(32, m+1, 25*n/100);
6   while sum(x, y*2, m+n, 23) <= x+y do
7     x = 3*x - n * sum(4.23, x+y);
8   endwhile
9
10  a = sum(n);
11  m = sum();
12  if m == 0 and n == a then write "yes!"; endif
13 endfunc
```

Joc de proves 1 (0.5 punts). Començarem modificant només la gramàtica per afegir l'operació `sum` al llenguatge, permetent que accepti un nombre qualsevol d'operands.

Es recomana NO aprofitar el codi existent per les crides a funcions, i tractar `sum` com una expressió apart, per no afectar el funcionament de la pràctica.

Com que és un operador, només es pot usar `sum` en una expressió, no com a instrucció.

Un cop fets els canvis, el primer joc de proves:

```
1 func main()
2   var i,z : int
3   var c,d : char
4
5   z = sum(34, z+1);
6   while 0.1 <= sum(z, sum(i*2, z+1), i+1) do
7     c = 3*x - i;
8   endwhile
9
10  y = c + z;
11  if z+3 then
12    write sum(3.7, -5, z*z, 2.3-i);
13    z = sum();
14  endif
15 endfunc
```

hauria de produir la sortida:

```
Line 7:7 error: Assignment with incompatible types.
Line 7:11 error: Identifier 'x' is undeclared.
Line 10:2 error: Identifier 'y' is undeclared.
Line 10:8 error: Operator '+' with incompatible types.
Line 11:2 error: Instruction 'if' requires a boolean condition.
```

Joc de proves 2 (0.5 punts). Ara farem el Typecheck de la nova operació. Caldrà comprovar que tots els operands són numèrics, i calcular el tipus del resultat.

Un cop fets els canvis, el segon joc de proves:

```
1 func main()
2   var i,z : int
3   var x,y : float
4   var c,d : char
5
6   z = sum(34, z+1, 25*i/100, d/4);
7   while i<z do
8     i = z + sum(y+1, 22, x/2);
9     y = z * sum(i+1, y*2, c, d);
10  endwhile
11
12  z = sum('a', d);
13  z = sum(3, 12, k) *sum();
14  c = sum(c, d, 'b');
15 endfunc
```

hauria de produir la sortida:

```
Line 6:30 error: Operator '/' with incompatible types.
Line 8:7 error: Assignment with incompatible types.
Line 9:13 error: Operator 'sum' with incompatible types.
Line 12:6 error: Operator 'sum' with incompatible types.
Line 13:17 error: Identifier 'k' is undeclared.
Line 14:4 error: Assignment with incompatible types.
Line 14:6 error: Operator 'sum' with incompatible types.
```

Joc de proves 3 (0.5 punts). A continuació generarem el codi per a l'operació `sum`, suposant que rep com a molt un operand enter:

Així arribem al tercer joc de proves:

```
1 func main()
2   var i,z : int
3
4   z = sum();
5   i = 10;
6   while i>z do
7     write i;
8     write " ";
9     write sum(2*(i+1));
10    write "\n";
11    i = i-1;
12  endwhile
13 endfunc
```

ha de produir la sortida:

```
10 22
9 20
8 18
7 16
6 14
5 12
4 10
3 8
2 6
1 4
```

Joc de proves 4 (1 punt). A continuació generarem el codi per a l'operació `sum`, suposant que rep un nombre qualsevol d'operands, tots ells enters:

<p>Amb això, el següent joc de proves:</p> <pre> 1 func main() 2 var n,i,z : int 3 var a : array [9] of int 4 5 n = 0; 6 while n<3 do 7 i = 0; 8 while i<9 do 9 read a[i]; 10 i = i + 1; 11 endwhile 12 13 z = sum(a[4]*a[1], a[7]*(1+a[0]), a[3]*4); 14 i = 0; 15 while i<9-1 do 16 write sum(i); 17 write " "; 18 write sum(a[i], 1, -1); 19 write " "; 20 write sum(i*2, a[i+1], (a[i]-1)/2, -12); 21 write "\n"; 22 i = i + 1; 23 endwhile 24 n = n + 1; 25 endwhile 26 endfunc </pre>	<p>amb l'entrada:</p> <pre> 23 54 3 77 -19 6 11 12 -2 44 22 4 -1 9 0 0 22 1 0 0 0 0 0 0 0 0 0 </pre>
	<p>ha d'escriure:</p> <pre> 0 23 53 1 54 19 2 3 70 3 77 13 4 -19 -8 5 6 11 6 11 17 7 12 5 0 44 31 1 22 4 2 4 -8 3 -1 2 4 9 0 5 0 -2 6 0 22 7 22 13 0 0 -12 1 0 -10 2 0 -8 3 0 -6 4 0 -4 5 0 -2 6 0 0 7 0 2 </pre>

Joc de proves 5 (1 punt). Finalment, ampliarem la generació de codi perquè també generi el codi adequat si hi ha operands reals.

Així passarem el darrer joc de proves:	que escriu la sortida:
<pre> 1 func fusio(ai : array [10] of int, 2 af : array [10] of float) : float 3 var j,k : int 4 var s : float 5 6 j = 0; 7 s = 0; 8 while j<10 do 9 s = sum(s, ai[j]); 10 if j<9 then k = ai[j+1]; 11 else if j>0 then k = ai[j-1]; 12 else k = -4; 13 endif 14 endif 15 s = sum(s, af[j], k); 16 write s; write "\n"; 17 j = j+1; 18 endwhile 19 20 return s; 21 endfunc 22 23 func main() 24 var i,z : int 25 var a : array [10] of int 26 var b : array [10] of float 27 var x : float 28 var c,d : char 29 30 i = 0; 31 while i<10 do 32 a[i] = 2*i + 1; 33 b[10-i] = sum(3*i, -1); 34 i = i + 1; 35 endwhile 36 37 i = 0; 38 while i<10 do 39 write b[i]; write " "; 40 i = i + 1; 41 endwhile 42 write "\n"; 43 44 write fusio(a,b); write "\n"; 45 endfunc </pre>	<pre> 0 26 23 20 17 14 11 8 5 2 4 38 73 109 146 184 223 263 304 342 342 </pre>

2 Operació filter (3.5 punts)

El segon exercici consisteix en dotar el llenguatge ASL amb una operació **filter** que comprova quins elements d'un array compleixen certa condició.

La sintaxi és **filter A into B using F**, on A és un array qualsevol, B és un array de **bool** amb el mateix nombre d'elements que A, i F és una funció que rep un únic argument del mateix tipus que els elements de A i retorna un **bool**.

L'efecte de l'operació es que a cada posició B[i] es guarda un booleà amb el resultat d'aplicar F sobre A[i]. Addicionalment, l'operació retorna com a resultat un enter indicant el nombre de valors **true** que s'han guardat a B.

Noteu que **filter** és una *operació* ternaria (combina tres operands A, B, i F per obtenir un resultat enter) que addicionalment té com a efecte lateral la modificació de B.

Com que és un operador, només es pot usar **filter** en una expressió, no com a instrucció.

Un exemple de codi:

```
1 func esvocal(c : char): bool
2     return c=='a' or c=='e' or c=='i' or c=='o' or c=='u';
3 endfunc
4
5 func esmult3(n : int): bool
6     return n%3 == 0;
7 endfunc
8
9 func main()
10     var ac : array [10] of char
11     var bc : array [10] of bool
12     var ai : array [20] of int
13     var bi : array [20] of bool
14     var s,t : int
15
16     s = filter ac into bc using esvocal;
17     t = s*2 - (filter ai into bi using esmult3) * 5;
18
19     if s>t then t = s+1; endif
20 endfunc
```

Joc de proves 6 (0.5 punts). El primer pas és afegir l'operació **filter** a la gramàtica, amb prioritat més alta que qualsevol altra operació.

De moment no us preocupeu del tipus dels operands, només que el resultat sigui enter.

El primer joc de proves:

```
1 func esmult3(n:int) : bool
2   return n %3 == 0;
3 endfunc
4
5 func main()
6   var a1,a2,a3: array [10] of int
7   var b1,b2: array [10] of bool
8
9   var n,m,s : int
10  var x,y : char
11
12  n = filter a1 into b1 using esmult3;
13  m = n + filter a2 into b2 using esmult3;
14  s = m * filter a3 into b1 using esmult3 + 4;
15
16  x = filter a1 into b2 using esmult3;
17  if filter a2 into b1 using esmult3 and
18    y != filter a1 into b2 using esmult3 then
19    s = 0;
20  endif
21 endfunc
```

genera els errors:

```
Line 16:4 error: Assignment with incompatible types.
Line 17:37 error: Operator 'and' with incompatible types.
Line 18:8 error: Operator '!=' with incompatible types.
```

Joc de proves 7 (1 punts). A continuació comprovarem el tipus dels operands de la nova operació. El primer operand ha de ser un array qualsevol. El segon ha de ser un array de `bool` de la mateixa mida que el primer. El darrer operand ha de ser una funció amb resultat `bool` i un únic paràmetre del mateix tipus que els elements del primer array.

El segon joc de proves:

```
1 func esvocal(c : char): bool
2     return c=='a' or c=='e' or c=='i' or c=='o' or c=='u';
3 endfunc
4
5 func esmult3(n : int): bool
6     return n%3 == 0;
7 endfunc
8
9 func main()
10     var ac : array [10] of char
11     var bc : array [10] of bool
12     var ai : array [20] of int
13     var bi : array [20] of bool
14
15     var s,t : int
16
17     s = filter ac into bc using esvocal;
18
19     t = s*2 - (filter ac into bi using esvocal) * 5;
20
21     if filter ac into bc using esmult3 then
22         s = 4 * (filter ac into bi using esmult3 + 1);
23     endif
24
25     ac = filter s into t using r;
26 endfunc
```

genera els errors:

```
Line 19:13 error: Operator 'filter' with incompatible types.
Line 21:2 error: Instruction 'if' requires a boolean condition.
Line 21:5 error: Operator 'filter' with incompatible types.
Line 22:14 error: Operator 'filter' with incompatible types.
Line 25:5 error: Assignment with incompatible types.
Line 25:7 error: Operator 'filter' with incompatible types.
Line 25:29 error: Identifier 'r' is undeclared.
```


Joc de proves 8 (0.5 punts). El següent pas és la generació de codi. Podeu començar suposant que la mida de l'array és 1 i que la funció sempre retorna **true**:

Amb això, el programa següent:

```
1 func ff(n : int): bool
2   return true;
3 endfunc
4
5 func main()
6   var a : array [1] of int
7   var b : array [1] of bool
8   var t : int
9
10  a[0] = 23;
11  t = 12;
12
13  write a[0];
14  write "\n";
15  write t + filter a into b using ff * 3;
16  write "\n";
17
18  if b[0] then
19    write "yes\n";
20  endif
21 endfunc
```

ha de donar la sortida:

```
23
15
yes
```

Joc de proves 9 (0.5 punts). En el següent joc de proves els arrays són encara de 1 element, però la funció booleana no té un resultat fix.

El codi d'aquest joc de proves:

```
1 func ff(n : int): bool
2   return n*2 % 3 != 1;
3 endfunc
4
5 func main()
6   var a : array [1] of int
7   var b : array [1] of bool
8   var t : int
9
10  t = 12;
11  a[0] = 23*t;
12
13  write a[0];
14  write "\n";
15  write (t - 4) * (filter a into b using ff + 2);
16  write "\n";
17
18  if not b[0] then
19    write "no\n";
20  else
21    write "yes\n";
22  endif
23
24  a[0] = a[0] + 2;
25
26  write (t - 4) * (filter a into b using ff + 2);
27  write "\n";
28
29  if b[0] then
30    write "no\n";
31  else
32    write "yes\n";
33  endif
34 endfunc
```

ha de generar la sortida:

```
276
24
yes
16
yes
```

Joc de proves 10 (1 punt). En el darrer joc de proves, tractarem arrays de qualsevol mida.

<p>El codi d'aquest joc de proves:</p> <pre> 1 func esvocal(c : char): bool 2 return c=='a' or c=='e' or 3 c=='i' or c=='o' or c=='u'; 4 endfunc 5 6 func esmult3(n : int): bool 7 return n%3 == 0; 8 endfunc 9 10 func main() 11 var ac : array [20] of char 12 var bc : array [20] of bool 13 var ai : array [10] of int 14 var bi : array [10] of bool 15 var i,n,s,t : int 16 17 // fill arrays ac and ai 18 i = 0; 19 while i<10 do 20 read ai[i]; 21 read ac[2*i]; 22 read ac[2*i+1]; 23 i = i + 1; 24 endwhile 25 26 // use filter on the arrays, 27 // with changing contents 28 n = 0; 29 while n<10 do 30 s = filter ac into bc using esvocal; 31 t = s*2 - filter ai into bi using esmult3*5; 32 write s; write " "; write t; write "\n"; 33 34 // change array content for next iteration 35 ai[n] = 33 * s - 10; 36 if t%2 == 0 then 37 ac[2*n] = 'k'; ac[2*n+1] = 'x'; 38 else 39 ac[2*n]='e'; ac[2*n+1] = 'o'; 40 endif 41 n = n + 1; 42 endwhile 43 44 if s>t then write "bad\n"; 45 else write "very good\n"; 46 endif 47 endfunc </pre>	<p>al llegir l'entrada següent:</p> <pre> 23 h u -12 o i 3 b f 0 o o -1 k k 2 a h 99 l e 4 p i -19 a n 1 o l </pre> <p>ha de generar la sortida:</p> <pre> 10 0 9 -2 7 -1 9 8 7 9 9 13 10 15 11 22 10 20 9 18 very good </pre>
--	--

Informació important

FITXERS PER A L'EXAMEN: Al Racó (`examens.fib.upc.edu`) trobareu un fitxer `examen.tgz` amb el següent contingut:

- `final-lab-CL-2020.pdf`: Aquest document, amb l'enunciat i les instruccions.
- `jps`: Subdirectori amb jocs de proves (`jp_chkt_XX.asl`) i `jp_genc_YY.asl`, i la seva corresponent sortida esperada (`jp_chkt_XX.err`) per als jocs de proves de validació semàntica, `jp_genc_YY.in/.out` per als jocs de proves de generació de codi). En els JPs de generació, no es compara el codi generat, sinó la sortida que produeix la tVM en executar-lo.
- `avalua.sh`: Script que executa tots els jocs de proves i diu si se superen o no.
- `empaqueta.sh`: Script que crea un fitxer `examen-nom.cognom.tgz` amb la vostra solució. Aquest és el fitxer que cal pujar al Racó.

PASSOS A SEGUIR:

- Feu una còpia de les carpetes `asl` i `common` de la vostra pràctica a un directori `examen`.

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

- Canvieu al nou directori `examen`, i descomprimiu-hi el fitxer `examen.tgz` del Racó:

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

Això extreurà el contingut del paquet, **afegint** al vostre directori `examen` els fitxers llistats anteriorment.

IMPORTANT: Feu-ho en l'ordre especificat (primer una còpia de la vostra pràctica i després descomprimir el `.tgz`). Fer-ho en l'ordre invers causarà que us falti codi necessari a `common` i que els JPs no siguin els adequats.

- Trebal·leu normalment a la carpeta `examen/asl`.

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

- Per executar tots els jocs de proves i veure si els passeu, executeu `../avalua.sh`.
- Per veure les diferències entre la sortida del vostre `asl` i la sortida esperada en un joc de proves concret de type check, podeu fer:

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

(Podeu ignorar la línia “There are semantic errors: no code generated” que genera el main)

- Per veure les diferències entre la sortida del vostre `asl` i la sortida esperada en un joc de proves concret de generació de codi, podeu fer:

```
./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
```

- Executeu `../empaqueta.sh` per crear el fitxer d'entrega `../examen-USERNAME.tgz` que cal pujar al Racó. Els paquets creats sense usar aquest script seran qualificats com **NO PRESENTAT**.