

Compiladors: Examen final de laboratori.

31 de maig de 2022

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 Operació factorial (3 punts)

Volem afegir a l'ASL l'operador `n!` que calcula el `factorial` d'un natural. L'operació tindrà les següent propietats:

- Es un operador postfix: `n!`
- Només accepta operands enters, i el resultat és sempre enter.
- Té la màxima prioritat, superior a qualsevol altre operador.
- Si l'operand és un enter negatiu, es produeix un error d'execució.

Per exemple:

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

Joc de proves 1 (0.5 punts). Començarem modificant només la gramàtica per afegir l'operador ! al llenguatge, amb la màxima prioritat.

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

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

hauria de produir la sortida:

Line 8:9 error: Operator '+' with incompatible types.
Line 12:18 error: Identifier 'f' is undeclared.
Line 15:3 error: Basic type required in 'write'.

Joc de proves 2 (0.5 punts). Ara farem el Typecheck de la nova operació. Caldrà comprovar que l'operand és enter, i afegir el codi necessari per calcular el tipus del resultat.

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

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

hauria de produir la sortida:

Line 8:12 error: Operator '!' with incompatible types.
Line 10:12 error: Operator '!' with incompatible types.
Line 12:18 error: Identifier 'f' is undeclared.
Line 15:3 error: Basic type required in 'write'.

Joc de proves 3 (1 punt). A continuació generarem el codi necessari per calcular el factorial de l'operand corresponent. Recordeu que $0! = 1! = 1$.

<p>Amb això, el següent joc de proves:</p> <pre> 1 func main () 2 var a, b : int 3 var A : array [2] of int 4 var x , y : float 5 6 x = 3; 7 a = 1; 8 read b; 9 y = b! + x; 10 write y; write "\n"; 11 A[0] = 8; 12 A[1] = 9; 13 y = a! * x; 14 if (-b! + x <= (A[0]/2)!*a!) then 15 x = A[1]! - b; 16 A[1] = (b-2)!!; 17 write "A[1]="; write A[1]; 18 write "\n"; 19 endif 20 write a; write " "; 21 write b; write "\n"; 22 write "A: "; 23 write A[0]; write " "; 24 write A[1]; write "\n"; 25 write x; write " "; 26 write y; write "\n"; 27 endfunc </pre>	<p>al llegir l'entrada següent:</p> <p>5</p>
	<p>ha d'escriure:</p> <p>123 A[1]=720 1 5 A: 8 720 362875 3</p>

Joc de proves 4 (1 punt). Finalment, generarem codi que comprovi el signe de l'operand, i avorti l'execució si és negatiu.

Per avortar l'execució amb l'error apropiat, cal que afegiu al codi generat la instrucció `instruction::HALT(code::INVALID_INTEGER_OPERAND)`.

Així passarem el darrer joc de proves:

```
1 func main ()
2   var a, b : int
3   var A : array [2] of int
4   var x , y : float
5
6   x = 3;
7   read b;
8   a = b! / b;
9   A[0] = 8;
10  A[1] = 9;
11  y = a! * x;
12  while x < 100 do
13    if ( -b! + x > (A[0]/2)!*a! ) then
14      x = A[1]! - b;
15      A[1] = (b+1)!!;
16    else
17      read x;
18    endif
19    write "loop ";
20    write A[1]; write " ";
21    write x; write "\n";
22  endwhile
23  write b-12; write "\n";
24  a = (b-12)! * 5;    // Here execution must halt
25  write "end ";
26  write y; write "\n";
27 endfunc
```

que amb les dades d'entrada:

```
2
40
```

produeix la sortida:

```
loop 9 40
loop 720 362878
-10
VM_CRASH: Execution halted: Invalid integer value in math operation.
```

2 Tipus matrix (4 punts)

El segon exercici consisteix en dotar el llenguatge ASL del tipus `matrix`, que emmagatzema arrays de dues dimensions.

Les matrius es declaren anàlogament als arrays, pero amb tipus `matrix` enlloc d'`array`, i especificant dues dimensions enlloc d'una. Els accessos són també anàlegs als dels arrays, pero especificant dues dimensions.

Igual que en els arrays, només s'accepten matrius de tipus bàsics.

Per exemple:

```
1 func main ()
2   var i,j : int
3   var A : array [2] of int
4   var M : matrix [10,20] of float
5
6   A[0] = 4;
7   A[1] = 7;
8   i = 0;
9   while i<10 do
10    j = 0;
11    while j<20 do
12      M[i,j] = A[i%2] + j;
13      j = j+1;
14    endwhile
15    i = i+1;
16  endwhile
17
18  write M[5,8]; write "\n";
19 endfunc
```

Joc de proves 5 (0.5 punts). El primer pas és afegir el tipus `matrix` a la gramàtica, i gestionar la declaració de variables d'aquest tipus.

Al mòdul auxiliar *TypesMgr* proporcionat amb l'exàmen trobareu els mètodes necessaris per crear i consultar el tipus `matrix`.

El primer joc de proves:

```
1 func main ()
2   var a,b, i,j : int
3   var t : bool
4   var A : array [2] of int
5   var M : matrix [10,20] of float
6
7   A[0] = t + 1;
8   A[1] = 7;
9   i = 0;
10  while i<10 do
11    j = not b;
12    while j<20 do
13      M[i,j] = A[i%2] + j;
14      j = j+1;
15    endwhile
16    i = i+1;
17    a[i+j] = 12;
18  endwhile
19
20  write M[5,8]; write "\n";
21 endfunc
```

genera els errors:

Line 7:12 error: Operator '+' with incompatible types.
Line 11:8 error: Assignment with incompatible types.
Line 11:10 error: Operator 'not' with incompatible types.
Line 17:6 error: Array access to a non array operand.

Joc de proves 6 (1 punts). A continuació farem la comprovació de tipus. Caldrà comprovar que els índexs en un accés a una matriu són enters, que la variable base és de tipus `matrix`, i obtenir el tipus dels elements de la matriu com a tipus resultant de l'accés. Observeu que totes les etapes son anàlogues a les del tractament dels arrays.

El segon joc de proves:

```

1 func main ()
2   var a,b, i,j : int
3   var t : bool
4   var A : array [2] of int
5   var B : array [2] of bool
6   var M : matrix [10,20] of float
7
8   A[0] = t;
9   A[1] = 7;
10  i = 0;
11  while i<10 do
12    j = 0;
13    while j<20 do
14      M[i,j] = A[i%2] + B[j%2];
15      t = M[i-1,j+1] and f;
16      j = j+1;
17    endwhile
18    i = i+1;
19    a[i+j] = M[a,t] + (B[i-1] or not b);
20  endwhile
21
22  write M[5,8]; write "\n";
23  write M[t,12]*A[8,1]; write "\n";
24 endfunc

```

genera els errors:

```

Line 8:8 error: Assignment with incompatible types.
Line 14:25 error: Operator '+' with incompatible types.
Line 15:24 error: Operator 'and' with incompatible types.
Line 15:28 error: Identifier 'f' is undeclared.
Line 19:6 error: Array access to a non array operand.
Line 19:19 error: Matrix access with non integer index.
Line 19:22 error: Operator '+' with incompatible types.
Line 19:35 error: Operator 'not' with incompatible types.
Line 23:11 error: Matrix access with non integer index.
Line 23:17 error: Matrix access to a non matrix operand.

```

Joc de proves 7 (1 punt). El següent pas consistirà en generar codi per a l'accés a una matriu dins d'una expressió.

A la memòria de la màquina virtual, la matriu s'emmagatzemarà seqüencialment (és a dir, el darrer element de la fila i va seguit del primer element de la fila $i + 1$), i l'índex de la primera fila i columna és el zero. Per tant, per calcular la posició relativa de l'element $A[i, j]$ respecte l'adreça de la variable A , cal usar la fórmula $offset = i * M + j$, on M es el número d'elements en cada fila de la matriu.

El codi d'aquest joc de proves:	ha de generar la sortida:
<pre> 1 func main () 2 var nz, i, j : int 3 var sum: float 4 var A : array [2] of int 5 var M : matrix [10,20] of float 6 7 nz = 0; 8 sum = 0; 9 i = 0; 10 while i<10 do 11 j = 0; 12 while j<20 do 13 if M[i,j] == 0 then 14 nz = nz + 1; 15 endif 16 sum = sum + M[i,j]; 17 if (10*i+j)%12 == 0 then 18 write nz; 19 write "\n"; 20 endif 21 j = j + 1; 22 endwhile 23 i = i+1; 24 endwhile 25 26 A[0] = nz; 27 A[1] = 23; 28 29 write "nz="; write nz; write "\n"; 30 write "sum="; write sum; write "\n"; 31 write "A: ["; write A[0]; write ","; 32 write A[1]; write "]\n"; 33 endfunc </pre>	<pre> 1 13 23 35 45 57 67 79 89 111 121 133 143 155 165 177 187 199 nz=200 sum=0 A: [200,23] </pre>

Joc de proves 8 (0.5 punt). Seguidament, tractarem de manera anàloga l'accés a una matriu com a *left-value*.

<p>El codi d'aquest joc de proves:</p> <pre> 1 func main () 2 var nz,a,b, i,j : int 3 var sum: float 4 var A : array [2] of int 5 var M : matrix [10,20] of float 6 7 read a; 8 read b; 9 10 i = 0; 11 while i<10 do 12 j = 0; 13 while j<20 do 14 M[i,j] = a; 15 a = a - b; 16 j = j+1; 17 endwhile 18 i = i+1; 19 endwhile 20 21 nz = 0; 22 sum = 0; 23 i = 0; 24 while i<10 do 25 j = 0; 26 while j<20 do 27 if M[i,j] == 100 then 28 nz = nz + 1; 29 endif 30 sum = sum + M[i,j]; 31 if (10*i+j)%12 == b then 32 write sum; write "\n"; 33 endif 34 j = j + 1; 35 endwhile 36 i = i+1; 37 endwhile 38 39 A[0] = nz; 40 A[1] = 23; 41 42 write "nz="; write nz; write "\n"; 43 write "sum="; write sum; write "\n"; 44 write "A: ["; write A[0]; write ","; 45 write A[1]; write "]\n"; 46 endfunc </pre>	<p>al llegir l'entrada següent:</p> <p>550 5</p>
	<p>ha de generar la sortida:</p> <p>3225 9135 13510 18100 21375 26820 28495 29845 30420 30450 29925 28635 27010 24400 21675 13920 nz=1 sum=10500 A: [1,23]</p>

Joc de proves 9 (0.5 punt). En aquest joc de proves permetrem l'assignació de matrius. Donat que les matrius s'emmagatzemen seqüencialment, el t-codi necessari per copiar una matriu [N,M] és exactament el mateix que el que copia un array de mida N*M. Per tant, simplement cal assegurar-se que aquesta part de la generació de codi s'executi també en el cas de les matrius. Recordeu que el *TypesMgr* té un mètode per consultar la mida d'un tipus.

<p>El codi d'aquest joc de proves:</p> <pre> 1 func main () 2 var i,j: int 3 var M,K : matrix [10,15] of float 4 var t : float 5 6 // fill 7 i = 0; 8 while i<10 do 9 j = 0; 10 while j<15 do 11 read M[i,j]; 12 j = j+1; 13 endwhile 14 i = i+1; 15 endwhile 16 17 // copy 18 K = M; 19 20 // transpose submatrix 21 i = 3; 22 while i<7 do 23 j = 3; 24 while j<i do 25 t = K[i,j]; 26 K[i,j] = K[j,i]; 27 K[j,i] = t; 28 j = j + 1; 29 endwhile 30 i = i+1; 31 endwhile 32 33 // print 34 i = 0; 35 while i<10 do 36 j = 0; 37 while j<15 do 38 write K[i,j]; 39 write " "; 40 j = j + 1; 41 endwhile 42 write "\n"; 43 i = i+1; 44 endwhile 45 46 endfunc </pre>	<p>al llegir l'entrada següent:</p> <pre> 1 0 9 9 9 1 1 1 1 1 1 1 1 1 1 1 6 0 9 9 1 1 1 1 1 1 1 1 1 1 1 6 6 0 9 1 1 1 1 1 1 1 1 1 1 1 6 6 6 0 1 </pre> <p>ha de generar la sortida:</p> <pre> 1 0 6 6 6 1 1 1 1 1 1 1 1 1 1 1 9 0 6 6 1 1 1 1 1 1 1 1 1 1 1 9 9 0 6 1 1 1 1 1 1 1 1 1 1 1 9 9 9 0 1 </pre>
--	--

Joc de proves 10 (0.5 punt). En el darrer joc de proves afegirem comprovació dinàmica de l'accés a matrius: El codi generat ha de comprovar si els valors dels índexs estan dins dels límits de les dimensions de la matriu, i avortar l'execució si no és així.

Per avortar l'execució amb el missatge d'error apropiat, cal que afegiu al codi generat la instrucció `instruction::HALT(code::INDEX_OUT_OF_RANGE)`.

El codi d'aquest joc de proves:	ha de generar la sortida:
<pre> 1 func main () 2 var i,j: int 3 var M: matrix [10,15] of float 4 var t : float 5 6 // fill 7 i = 0; 8 while i<10 do 9 j = 0; 10 while j<15 do 11 M[i,j] = i+j; 12 j = j+1; 13 endwhile 14 i = i+1; 15 endwhile 16 17 // print 18 i = 0; 19 while i<15 do 20 j = 0; 21 while j<10 do 22 write M[i,j]; 23 write " "; 24 j = j + 1; 25 endwhile 26 write "\n"; 27 i = i+1; 28 endwhile 29 30 endfunc </pre>	<pre> 0 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 10 2 3 4 5 6 7 8 9 10 11 3 4 5 6 7 8 9 10 11 12 4 5 6 7 8 9 10 11 12 13 5 6 7 8 9 10 11 12 13 14 6 7 8 9 10 11 12 13 14 15 7 8 9 10 11 12 13 14 15 16 8 9 10 11 12 13 14 15 16 17 9 10 11 12 13 14 15 16 17 18 VM_CRASH: Execution halted: Container index out of range. </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-2022.pdf`: Aquest document, amb l'enunciat i les instruccions.
- `jps`: Subdirectori amb jocs de proves (`jp_chk_XX.asl`) i `jp_genc_YY.asl`, i la seva corresponent sortida esperada (`jp_chk_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.
- `common`: Subdirectori amb els mòduls auxiliars `SemErrors`, `TypesMgr` i `code` ampliat amb el codi necessari per a l'examen.
- `tvm`: Subdirectori amb la màquina virtual ampliada amb la instrucció `HALT`.
- `evalua.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  
...
```

(Si la compilació és lenta per sobrecàrrega del servidor, podeu executar l'script `fast-make.sh`)

- Per executar tots els jocs de proves i veure si els passeu, executeu `../evalua.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_chk_XX.asl | diff -y - ../jps/jp_chk_XX.err
```

(Podeu ignorar la línia "There are semantic errors: no code generated")
- 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**.