

Compiladors: Examen final de laboratori.

6 de juny de 2024

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 Instrucció swap (**3.5** punts)

Volem afegir a l'ASL la instrucció **swap** que intercanvia els valors de dos expressions referenciables (*left-expressions*). Els seus arguments poden ser de tipus bàsics i també arrays. Per exemple:

```
1 func main()
2   var n: int
3   var x, y: float
4   var A, B: array[10] of float
5
6   swap(x, y);           // swap de tipus basics
7   n = 10;
8   swap(x, A[n-1]);
9   swap(A[0], A[n-1]);
10  swap(A, B);           // swap dels arrays sencers
11 endfunc
```

Joc de proves 1 (0.25 punts). Començarem modificant només la gramàtica per afegir la instrucció `swap` al llenguatge.

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

```
1 func main()
2   var i, j: int
3   var c1, c2: char
4   var x, y: float
5   var Ai: array[10] of int
6   var Af: array[10] of float
7
8   swap(x+1, y);
9
10  swap(x, y);
11  swap(z, A[0]);
12  swap(A[0], A[n-1]);
13
14  swap(3, A[2]);
15
16 endfunc
```

hauria de produir els següents errors sintàctics:

```
line 8:8 mismatched input '+' expecting ','
line 14:7 mismatched input '3' expecting ID
Lexical and/or syntactical errors have been found.
```

Joc de proves 2 (0.5 punts). Ara farem el Typecheck de la nova instrucció. Caldrà comprovar que els arguments són de tipus compatibles.

Al mòdul auxiliar *SemErrors* proporcionat amb l'exàmen trobareu els errors específics d'aquesta nova instrucció.

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

```
1 func main()
2   var i, j, n: int
3   var c1, c2: char
4   var x, y: float
5   var Ai: array[10] of int
6   var Af: array[10] of float
7
8   // ok:
9   swap(x, y);
10  swap(c1, c2);
11  swap(Ai[0], Ai[n-1]);
12
13  // error en swap:
14  swap(c1, x);
15  swap(x, Ai[2]);
16
17  // errors en operands:
18  swap(x, x[3]);
19  swap(zzzz, Ai[3]);
20
21 endfunc
```

hauria de produir la sortida:

```
Line 14:2 error: Incompatible arguments in swap.
Line 15:2 error: Incompatible arguments in swap.
Line 18:10 error: Array access to a non array operand.
Line 19:7 error: Identifier 'zzzz' is undeclared.
```

Joc de proves 3 (0.75 punts). A continuació generarem el codi necessari per realitzar l'intercanvi dels valors de dues variables de tipus bàsics.

<p>Amb això, el següent joc de proves:</p> <pre> 1 func main() 2 var i, j: int 3 var c1, c2: char 4 var x, y: float 5 var Ai: array[10] of int 6 var Af: array[10] of float 7 8 // swap(ID1, ID2) amb ID de tipus basic 9 x = 55; y = 99; 10 swap(x, y); 11 write x; write " "; write y; write "\n"; 12 13 endfunc </pre>	<p>que ha d'escriure:</p> <p>99 55</p>
---	--

Joc de proves 4 (1 punt). Ara també considerarem casos de *left-expressions* que són accessos a arrays.

<p>Amb això, el següent joc de proves:</p> <pre> 1 func main() 2 var i, j, k, n: int 3 var c1, c2: char 4 var x, y: float 5 var Ai: array[10] of int 6 var Af: array[10] of float 7 8 n = 10; 9 i = 0; 10 while i < n do 11 Ai[i] = i*i + 1; 12 i = i + 1; 13 endwhile 14 15 read k; 16 j = 5; 17 // swap(left_expr1, left_expr2) 18 swap(k, Ai[j+1]); 19 write "A. "; write k; write " "; 20 write Ai[j+1]; write "\n"; 21 22 swap(Ai[0], Ai[n-1]); 23 write "B. "; write Ai[0]; write " "; 24 write Ai[n-1]; write "\n"; 25 26 swap(Ai[0], k); 27 write "C. "; write Ai[0]; write " "; 28 write k; write "\n"; 29 30 i = 0; 31 while i < n do 32 write "Ai["; write i; write "] = "; 33 write Ai[i]; write "\n"; 34 i = i + 1; 35 endwhile 36 37 endfunc </pre>	<p>al llegir l'entrada següent:</p> <p>1234</p> <p>ha d'escriure:</p> <p>A. 37 1234 B. 82 1 C. 37 82 Ai[0] = 37 Ai[1] = 2 Ai[2] = 5 Ai[3] = 10 Ai[4] = 17 Ai[5] = 26 Ai[6] = 1234 Ai[7] = 50 Ai[8] = 65 Ai[9] = 1</p>
---	---

Joc de proves 5 (1 punt). Finalment, generarem codi que intercanviï tots els valors de dos arrays.

Així passarem el darrer joc de proves:

```
1 func main()
2   var i, n: int
3   var x, y: float
4   var A, B: array[10] of float
5
6   n = 10;
7   i = 0;
8   while i < n do
9     A[i] = i*i + 1;
10    B[i] = 1000-i;
11    i = i + 1;
12  endwhile
13
14  i = 0;
15  while i < n do
16    write "before. A["; write i; write "] = "; write A[i];
17    write " - B["; write i; write "] = "; write B[i]; write "\n";
18    i = i + 1;
19  endwhile
20
21  swap(A, B);
22
23  i = 0;
24  while i < n do
25    write "after. A["; write i; write "] = "; write A[i];
26    write " - B["; write i; write "] = "; write B[i]; write "\n";
27    i = i + 1;
28  endwhile
29
30 endfunc
```

que produeix la sortida:

```
before. A[0] = 1 - B[0] = 1000
before. A[1] = 2 - B[1] = 999
before. A[2] = 5 - B[2] = 998
before. A[3] = 10 - B[3] = 997
before. A[4] = 17 - B[4] = 996
before. A[5] = 26 - B[5] = 995
before. A[6] = 37 - B[6] = 994
before. A[7] = 50 - B[7] = 993
before. A[8] = 65 - B[8] = 992
before. A[9] = 82 - B[9] = 991
after. A[0] = 1000 - B[0] = 1
after. A[1] = 999 - B[1] = 2
after. A[2] = 998 - B[2] = 5
after. A[3] = 997 - B[3] = 10
after. A[4] = 996 - B[4] = 17
after. A[5] = 995 - B[5] = 26
after. A[6] = 994 - B[6] = 37
after. A[7] = 993 - B[7] = 50
after. A[8] = 992 - B[8] = 65
after. A[9] = 991 - B[9] = 82
```

2 Instrucció switch (3.5 punts)

El segon exercici consisteix en dotar el llenguatge ASL de la instrucció **switch**, amb una semàntica similar a la del llenguatge C++.

El **switch** conté una expressió i una sèrie de casos *valor-instruccions*. L'expressió és avaluada i comparada amb els valors dels casos. S'executen les instruccions corresponents al **primer** cas amb valor igual al de l'expressió.

Opcionalment, pot haver-hi un cas **default** amb instruccions que s'executen quan cap valor coincideix.

Per exemple:

```
1 func main()
2   var i, j : int
3
4   i = 3;
5   write "case value = ";
6   switch i*i:
7     case 1: write 1; write "\n"; endcase
8     case 4: write 4; write "\n"; endcase
9     case 9: write 9; write "\n"; endcase
10    case 16: write 16; write "\n"; endcase
11    default: write "default"; write "\n";
12    endswitch
13
14 endfunc
```

Joc de proves 6 (0.5 punts). El primer pas és afegir la instrucció **switch** a la gramàtica.

PISTA: Tot i que tots els casos d'un **switch** venen donats per *valors*, en la gramàtica pot resultar més fàcil acceptar-los de forma més genèrica, com a *expressions* (que inclouen els *valors*). En qualsevol cas, tots els jocs de proves fan servir només *valors* en els casos.

El primer joc de proves:

```
1 func main()
2   var i , j : int
3   var c1 , c2 : char
4   var A: array[10] of int
5
6   if i > c2 then
7     switch c1:
8       case 'a': write 1; endcase
9     endswitch
10    A[c1] = 19;
11  endif
12
13  j = c1[3] + k;
14
15  switch i*j:
16    case 3: i = A[i+1]; j = j-1; endcase
17    case 1: j = 3-1; endcase
18    default: i = 4; write 2*j; write "\n";
19    endswitch
20
21 endfunc
```

genera els errors:

Line 6:7 error: Operator '>' with incompatible types.
Line 10:6 error: Array access with non integer index.
Line 13:6 error: Array access to a non array operand.
Line 13:14 error: Identifier 'k' is undeclared.

Joc de proves 7 (0.5 punts). A continuació farem el Typecheck de la nova instrucció. Caldrà comprovar que el tipus de l'expressió és comparable amb el de cadascun dels valors dels casos. Feu servir el mètode `comparableTypes(tE, tV, "=")` del mòdul *TypesMgr*.

Al mòdul auxiliar *SemErrors* proporcionat amb l'exàmen trobareu els errors específics d'aquesta nova instrucció.

El segon joc de proves:

```

1 func main()
2   var i , j : int
3   var c1 , c2 : char
4   var A: array[10] of float
5
6   if i > j then
7     switch c1:
8       case 'a': write(1); endcase
9     endswitch
10
11     switch p:
12       case 33: i = p; endcase
13     endswitch
14   endif
15
16   switch i*j:
17     case false: i = 0; endcase
18     case 3: i = A[i]+1; j = A-1; endcase
19     case 2.5: i = 2; endcase
20     case 1: j = 3 > c1; endcase
21     case 'a': i = i+1; endcase
22     default: k = 4;
23   endswitch
24
25 endfunc

```

genera els errors:

```

Line 11:11 error: Identifier 'p' is undeclared.
Line 12:17 error: Identifier 'p' is undeclared.
Line 17:7 error: Incompatible value in switch in case 'false'.
Line 18:12 error: Assignment with incompatible types.
Line 18:27 error: Operator '-' with incompatible types.
Line 20:12 error: Assignment with incompatible types.
Line 20:16 error: Operator '>' with incompatible types.
Line 21:7 error: Incompatible value in switch in case ''a''.
Line 22:11 error: Identifier 'k' is undeclared.

```

Joc de proves 8 (1 punt). El següent pas consistirà en generar codi per la instrucció **switch**.

En primer lloc tractarem un cas molt simple on el valor del primer *case* ja és igual al de la expressió.

El codi d'aquest joc de proves:	ha de generar la sortida:
<pre> 1 func main() 2 var i, j : int 3 4 i = 2; 5 switch i*i: 6 case 4: write "case value = 4\n"; endcase 7 case 0: write "case value = 0\n"; endcase 8 case 1: write "case value = 1\n"; endcase 9 endswitch 10 endfunc </pre>	<pre> case value = 4 </pre>

Joc de proves 9 (1 punt). Seguidament, tractarem **switch**'s on qualsevol cas és possible, i també pot haver-hi cap o més d'un valor coincident.

El codi d'aquest joc de proves:	al llegir l'entrada següent:
<pre> 1 func main() 2 var i, j : int 3 4 read i; 5 switch i*i: 6 case 1: write "case value = 1\n"; endcase 7 case 4: write "case value = 4\n"; endcase 8 case 9: write "case value = 9\n"; endcase 9 case 16: write "case value = 16\n"; endcase 10 case 9: write "case value = 9 err!\n"; endcase 11 endswitch 12 13 switch i: 14 case 33: i = 33; endcase 15 endswitch 16 17 write i; write "\n"; 18 endfunc </pre>	<pre> 3 </pre>
	ha de generar la sortida:
	<pre> case value = 9 3 </pre>

Joc de proves 10 (0.5 punts). En el darrer joc de proves apareixen **switch**'s que contenen un darrer cas **default**.

El codi d'aquest joc de proves:

```
1 func main()
2   var i, j : int
3
4   i = 1;
5   while i < 4 do
6     write "i="; write i; write "    i*i="; write i*i;
7     switch i*i:
8       case 0: write "    case value = 0\n"; endcase
9       case 1: write "    case value = 1\n"; endcase
10      case 4: write "    case value = 4\n"; endcase
11      default: write "    case default\n";
12    endswitch
13    i = i + 1;
14  endwhile
15 endfunc
```

ha de generar la sortida:

```
i=1    i*i=1    case value = 1
i=2    i*i=4    case value = 4
i=3    i*i=9    case default
```


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-2024.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.
- `common`: Subdirectori amb el mòdul auxiliar `SemErrors` ampliat amb el codi necessari per a l'examen.
- `tvm`: Subdirectori amb la maquina virtual.
- `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.

- Treballeu normalment a la carpeta `examen/asl`.

```
cd asl
make pristine
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_chkt_XX.asl | diff -y - ../jps/jp_chkt_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**.