

Compiladors: Examen parcial de laboratori.

26 d'abril de 2023

ATENCIÓ: Al Racó trobareu els jocs de proves i codi necessari per a fer l'examen. El paquet conté versions modificades dels mòduls `common/SemErrors.*` i `common/TypesMgr.*` amb els mètodes necessaris per donar els nous errors o gestionar els nous tipus que apareixen als exercicis de 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 Punters (3.5 punts)

Volem afegir a l'ASL la possibilitat de tenir punters, semblants als de C.

Els punters es declaren amb el tipus `pointer to <tipus>`, on `<tipus>` pot ser qualsevol tipus del llenguatge, bàsic o no, incloent el tipus punter.

Els punters tenen les operacions:

- **Dereferenciació:** Obtenir el contingut apuntat pel punter: `*p`
L'operació de dereferenciació es pot aplicar a qualsevol expressió de tipus punter. El resultat es del tipus apuntat per l'expressió dereferenciada.
- **Creació:** Obtenir el punter a una variable: `&x`
Es pot obtenir el punter de qualsevol expressió referenciable (es a dir, que sigui *left expression*). El resultat és de tipus punter al tipus de l'expressió. El resultat de l'operació **no** és referenciable.

Adicionalment, existeix la constant `null` que es comporta com un `(void*)` de C, i es pot assignar o comparar amb un punter a qualsevol tipus. Els mètodes `comparableTypes` i `copyableTypes` han estat adaptats.

Un exemple de codi:

```
1 func main()
2   var i,j : int
3   var a : array [10] of bool
4   var p : pointer to int
5   var q : pointer to array [10] of bool
6   var s : pointer to pointer to int
7   var t : pointer to bool
8
9   p = &i;
10  *p = 3;    // i is now 3
11  i = i+1;
12  *p = *p+1;
13  write i;   // will write 5
14
15  p = &j;
16  s = &p;
17  **s = 2;   // j is now 2
18
19  q = &a;
20  t = &a[j+1];
21  p = null;
22  if (*q)[i] and not *t and q!=null then
23    write "that's it";
24  endif
25 endfunc
```

Joc de proves 1 (1 punt). El primer pas és estendre la gramàtica amb el nou tipus, el token `null`, i les operacions `*` i `&` sobre punters, les quals tenen alta prioritat, just per sota de l'operació d'accés a array. Per evitar problemes amb el compilador de C++, doneu un nom diferent de `NULL` al token corresponent al nou valor `null`.

De moment, no farem encara cap comprovació de tipus.

El primer joc de proves:

```
1 func f(a:int): bool
2   return true;
3 endfunc
4
5 func main()
6   var i,j : int
7   var y: float
8   var p : pointer to int
9   if p == null then
10    p = &i;
11  endif
12  x = 3;
13  j = *p + 1;
14  y = 'a' or not 23;
15  if not f(y) then
16    write "meeeck!"    // syntax error!! no TypeCheck is done
17  endif
18 endfunc
```

genera el següent error:

```
line 17:2 missing ',' at 'endif'
Lexical and/or syntactical errors have been found.
```

Joc de proves 2 (1 punt). A continuació començarem a comprovar el tipus de les noves operacions. En aquest joc de proves, cal adaptar les declaracions de variables per tal que els punters es declarin correctament. També cal adaptar la comprovació de tipus per tal que l'operació `*` doni un error si l'operand no és un punter. Finalment, cal que `*` propagui la decoració del tipus resultant de l'operació.

Al `common` modificat per a l'examen trobareu que la classe `TypesMgr` disposa de mètodes per tractar els punters. També trobareu mètodes per generar errors relatius als punters a la versió modificada de `SemErrors`.

Així, el segon joc de proves:

```
1 func main()
2   var i,j : int
3   var y: float
4   var p : pointer to int
5   var pb : pointer to bool
6
7   j = *p + (x>0);
8   j = *i;
9   y = 2 * (y+1);
10  p = p + 1;
11  y = *p + 1;
12  if not *pb or y < 0 then
13    write *p + *(p/2);
14    i = **pb / 2;
15  endif
16 endfunc
```

genera els errors:

```
Line 7:9 error: Operator '+' with incompatible types.
Line 7:12 error: Identifier 'x' is undeclared.
Line 8:6 error: Pointer access to a non pointer operand.
Line 10:4 error: Assignment with incompatible types.
Line 10:8 error: Operator '+' with incompatible types.
Line 13:15 error: Pointer access to a non pointer operand.
Line 13:18 error: Operator '/' with incompatible types.
Line 14:8 error: Pointer access to a non pointer operand.
```

Joc de proves 3 (0.5 punts). Ara farem el typecheck de l'operació `&`, i del valor `null`. Cal que `&` propagui la decoració del tipus resultant de l'operació (aquesta decoració pot ser **error**).

Així, el segon joc de proves:

```
1 func main()
2   var i,j : int
3   var y: float
4   var p : pointer to int
5   if p == null then
6     p = &i;
7   endif
8   j = *p + (x>0);
9   i = &j;
10  j = *i;
11  y = null;
12  y = 2 * (y+1);
13  p = p + 1;
14  if y == null then
15    write "is null";
16  endif
17 endfunc
```

genera els errors:

```
Line 8:9 error: Operator '+' with incompatible types.
Line 8:12 error: Identifier 'x' is undeclared.
Line 9:4 error: Assignment with incompatible types.
Line 10:6 error: Pointer access to a non pointer operand.
Line 11:4 error: Assignment with incompatible types.
Line 13:4 error: Assignment with incompatible types.
Line 13:8 error: Operator '+' with incompatible types.
Line 14:7 error: Operator '==' with incompatible types.
```

Joc de proves 4 (0.5 punts). Seguint amb les comprovacions de tipus, cal gestionar la referenciabilitat en les operacions amb punters: l'operand de `&` ha de ser referenciable, i el resultat de l'operació `*` és sempre referenciable però el de l'operació `&` no ho és.

El tercer joc de proves:

```
1 func f(x :int) : bool
2     return x!=0;
3 endfunc
4
5 func main()
6     var i,j : int
7     var y: float
8     var p : pointer to int
9     var q : pointer to pointer to int
10    if p == null then
11        p = &i;
12    endif
13    j = *p + (x>0);
14    *p = *p/(*p+1);
15    *p = *p*( *p);
16    *p = *p * *p;
17    q = &p;
18    i = &&q;
19    q = &f;
20    p = q + 1;
21    p = *q;
22    p = *q + 1;
23    j = *q;
24    i = 3 + **q;
25    j = 5 * **q;
26    i = ***q;
27 endfunc
```

ha de produir els errors:

```
Line 13:9 error: Operator '+' with incompatible types.
Line 13:12 error: Identifier 'x' is undeclared.
Line 18:4 error: Assignment with incompatible types.
Line 18:6 error: Referenceable expression required in '&'.
Line 19:6 error: Referenceable expression required in '&'.
Line 20:4 error: Assignment with incompatible types.
Line 20:8 error: Operator '+' with incompatible types.
Line 22:4 error: Assignment with incompatible types.
Line 22:9 error: Operator '+' with incompatible types.
Line 23:4 error: Assignment with incompatible types.
Line 26:6 error: Pointer access to a non pointer operand.
```

Joc de proves 5 (0.5 punts). Fins ara només hem provat punters a tipus bàsics. El següent joc de proves valida el funcionament dels punters a arrays: ara hi han accessos a arrays apuntats per punters. Si heu fet solucions generals a la gramàtica i als passos anteriors, possiblement el passareu directament; si no, haureu d'adaptar els accessos a arrays.

Segons com enfoqueu aquest exercici, els anteriors poden deixar de funcionar. Es recomana que feu una entrega abans de fer les modificacions relacionades amb aquest joc de proves.

El quart joc de proves:

```
1 func main()
2   var A,B : array[10] of bool
3   var pA, pB : pointer to array[10] of bool
4   var ppA, ppB : pointer to pointer to array[10] of bool
5   var i: int
6   var b: bool
7   var p, q: pointer to bool
8
9   pA = &A;
10  pB = &B;
11  *pA = *pB;
12  A[0] = not ((*pA)[1]);
13  i = 3.5;
14  A[b+1] = b[i] or 2;
15
16  ppA = &pA;
17  ppB = &pB;
18  (**ppA)[1] = (*pA)[0] or not B[1];
19  **ppA[1] = *pA[i+1] or not B[2];
20  (*ppA)[i-1] = true;
21 endfunc
```

produirà els errors:

```
Line 13:4 error: Assignment with incompatible types.
Line 14:5 error: Operator '+' with incompatible types.
Line 14:11 error: Array access to a non array operand.
Line 14:16 error: Operator 'or' with incompatible types.
Line 17:6 error: Assignment with incompatible types.
Line 17:8 error: Referenceable expression required in '&'.
Line 19:14 error: Array access to a non array operand.
Line 20:2 error: Array access to a non array operand.
```

2 Estructura *case* (3.5 punts)

El segon exercici consisteix en afegir la instrucció **case** al llenguatge ASL. Així podríem escriure per exemple:

```
1 func main()
2   var b : bool
3   var n, i: int
4   var a : char
5
6   i = 4;
7   read n;
8   case i-2*n of
9     12 : n = n+1;
10       write n;
11     10 : i = i*2-n;
12     2,8,6,14:
13       b = true;
14       if not b or i>4 then
15         n = 22;
16       endif
17     default:
18       b = false;
19       i = 0;
20   endcase
21 endfunc
```

Les condicions per a la correcta formació del **case** són les següents:

- L'expressió només pot ser de tipus enter o caràcter.
- Tots els valors dels casos han de ser del mateix tipus que l'expressió.
- No hi pot haver dos casos que incloguin el mateix valor.
- El cas **default** és opcional, però si apareix ha de ser l'últim.

Joc de proves 6 (0.5 punts). El primer pas és modificar la gramàtica per tal d'afegir els tokens necessaris i la instrucció `case`, amb el `default` opcional.

Sense fer encara comprovacions de tipus, podem superar el primer joc de proves. Per ara, no considerarem llistes de valors als casos, sinó simplement un valor per cas.

Aquest joc de proves:

```
1 func main()
2   var b : bool
3   var n, i: int
4   var a : char
5
6   i = 4;
7   read n;
8
9   case i-2*n of
10     12 : n = n+1;
11         write n;
12
13     10 : i = i*2-n;
14
15     2 :
16         b = true;
17         if not b or i>4 then
18             n = 22;
19             case a of
20                 'X' : b = true;
21                 'Y' : n = 0;
22             endcase
23         endif
24
25     default:
26         b = false;
27         i = 0;
28   endcase
29
30   a = b>0 or n/a;
31
32 endfunc
```

genera els errors:

```
Line 30:4 error: Assignment with incompatible types.
Line 30:7 error: Operator '>' with incompatible types.
Line 30:10 error: Operator 'or' with incompatible types.
Line 30:14 error: Operator '/' with incompatible types.
```

Joc de proves 7 (0.5 punts). El següent joc de proves ha de comprovar que l'expressió del `case` sigui de tipus enter o caràcter. En cas que no ho sigui, es redecorarà amb el tipus `error`.

Per superar aquest joc de proves heu de visitar les instruccions de tots els casos.

Trobareu mètodes per generar els errors específics per al `case` a la versió adaptada de `SemErrors`.

Passarem aquest joc de proves:

```
1 func main()
2   var b : bool
3   var n, i: int
4   var a : char
5   var x : float
6
7   i = 4;
8   read n;
9
10  case not b or x>1 of
11    'a' : b = true;
12    default : b = not b;
13  endcase
14
15  case f*3 - 1 of
16    12 : n = n+1;
17    write n;
18
19    10 : i = 0;
20      case a of
21        'X' : b = true;
22        'Y' : n = 0;
23      endcase
24
25    8 : b = true;
26      while not x or i>4 do
27        n = 22;
28        case n-1 of
29          0 : n = n+1;
30          1 : n = n-1;
31          10 : b = false;
32        endcase
33      endwhile
34
35  default:
36    b = true;
37    x = false;
38    i = x;
39  endcase
40 endfunc
```

generant els errors:

```
Line 10:2 error: Case instruction requires an expression of type int or char.
Line 15:7 error: Identifier 'f' is undeclared.
Line 26:16 error: Operator 'not' with incompatible types.
Line 37:11 error: Assignment with incompatible types.
Line 38:11 error: Assignment with incompatible types.
```

Joc de proves 8 (1 punt). En aquest pas comprovarem que els valors dels casos són del mateix tipus que l'expressió: per a cada valor que no sigui `int` o `char`, o tingui un tipus diferent al de l'expressió, cal donar l'error corresponent.

Si l'expressió és de tipus `error` (ja sigui perquè ho era originalment o perquè s'ha rede-corat per no ser `int` o `char`) llavors el primer valor *vàlid* (`int` o `char`) determina el tipus de “referència” del `case`: guardeu el tipus d'aquest valor, i reporteu els que siguin diferents.

Amb això, el programa següent:

```

1
2 func main()
3   var b : bool
4   var n, i: int
5   var a : char
6   var x : float
7
8   i = 4;
9   read n;
10
11  case not b or x>1 of
12    12 : n = n+1;
13      write n;
14
15    6 : i = 0;
16      case a of
17        'X' : b = true;
18        2.3 : x = x/3;
19        0 : n = 0;
20      endcase
21
22    8 : b = true;
23      while not x or i>4 do
24        n = 22;
25        case n-1 of
26          0 : n = n+1;
27          true : n = n-1;
28          'X' : b = false;
29        endcase
30      endwhile
31
32  default:
33    b = true;
34    x = false;
35    i = x;
36  endcase
37
38 endfunc

```

ha de donar la sortida:

```

Line 11:2 error: Case instruction requires an expression of type int or char.
Line 18:13 error: Incompatible value in case instruction.
Line 19:14 error: Incompatible value in case instruction.
Line 23:16 error: Operator 'not' with incompatible types.
Line 27:16 error: Incompatible value in case instruction.
Line 28:16 error: Incompatible value in case instruction.
Line 34:11 error: Assignment with incompatible types.
Line 35:11 error: Assignment with incompatible types.

```

Joc de proves 9 (1 punt). A continuació comprovarem si algun dels valors està repetit. Per fer-ho, cal que des del node `case` s'accedeixi als valors dels fills i es vagin guardant en un `std::map<std::string,bool>` o `std::set<std::string>` per tal de detectar i reportar els que puguin aparèixer repetits. Feu servir el mètode `getText()` per accedir al text de cada valor.

Trobareu errors específics per al `case` a la versió adaptada de `SemErrors`.

En aquest joc de proves, seguim suposant que tots els casos tenen només un valor.

Amb això, el programa següent:

```
1 func main()
2   var b : bool
3   var n, i: int
4   var a : char
5   var x : float
6
7   i = 4;
8   read n;
9
10  case not b or x>1 of
11    12 : n = n+1;
12      write n;
13
14    6 : i = 0;
15      case a of
16        'X' : b = true;
17        'Z' : n = 2;
18        'x' : n = 0;
19        'Z' : b = false;
20      endcase
21
22    12 : b = true;
23      while not x or i>4 do
24        n = 22;
25        case n-1 of
26          0 : n = n+1;
27          9 : n = n-1;
28          '<' : b = false;
29        endcase
30      endwhile
31
32  default:
33    b = true;
34    x = false;
35    i = x;
36  endcase
37
38 endfunc
```

ha de donar la sortida:

```
Line 10:2 error: Case instruction requires an expression of type int or char.
Line 19:13 error: Repeated value in case instruction.
Line 22:6 error: Repeated value in case instruction.
Line 23:16 error: Operator 'not' with incompatible types.
Line 28:16 error: Incompatible value in case instruction.
Line 34:11 error: Assignment with incompatible types.
Line 35:11 error: Assignment with incompatible types.
```

Joc de proves 10 (0.5 punts). Finalment, ampliarem les comprovacions de tipus a quan els casos contenen una llista de valors.

Segons com enfoqueu aquest exercici, els anteriors poden deixar de funcionar. Es recomana que feu una entrega abans de fer les modificacions relacionades amb aquest joc de proves.

Per passar aquest darrer cas, heu de:

- Estendre la gramàtica perquè cada cas pugui tenir una llista de valors enlloc d'un de sol.
- Aplicar les mateixes comprovacions de tipus dels exercicis anteriors a tots els valors de cada llista.

Amb això, el programa següent:

```
1 func main()
2   var b : bool
3   var n, i: int
4   var a : char
5   var x : float
6
7   i = 4;
8   read n;
9
10  case not b or x>1 of
11    12 : n = n+1;
12    write n;
13
14    6 : i = 0;
15    case a of
16      'X','Y' : b = true;
17      'Z'      : n = 2;
18      'x','X' : n = 0;
19    endcase
20
21    4,6,'a',12 : b = true;
22    while not x or i>4 do
23      n = 22;
24      case n-1 of
25        0,2,4,6,8 : n = n+1;
26        1,3,false,7,9 : n = n-1;
27        8,10,11 : b = false;
28      endcase
29    endwhile
30
31    default:
32      b = true;
33      x = false;
34      i = x;
35  endcase
36
37 endfunc
```

ha de donar la sortida:

```
Line 10:2 error: Case instruction requires an expression of type int or char.
Line 18:17 error: Repeated value in case instruction.
Line 21:7 error: Repeated value in case instruction.
Line 21:9 error: Incompatible value in case instruction.
Line 21:13 error: Repeated value in case instruction.
Line 22:16 error: Operator 'not' with incompatible types.
Line 26:20 error: Incompatible value in case instruction.
Line 27:16 error: Repeated value in case instruction.
Line 33:11 error: Assignment with incompatible types.
Line 34:11 error: Assignment with incompatible types.
```

Informació important

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

- `parcial-lab-CL-2023.pdf`: Aquest document, amb l'enunciat i les instruccions.
- `jps`: Subdirectori amb jocs de proves (`jp_chkt_XX.asl`), i la seva corresponent sortida esperada (`jp_chkt_XX.err`).
- `common`: Subdirectori amb els mòduls auxiliars `SemErrors` i `TypesMgr` ampliat amb el codi necessari per a l'examen.
- `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 nou directori `examen`.

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

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

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

Això extraurà 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 -j4
...
```

(Si la compilació és lenta per sobrecàrrega del servidor, podeu executar l'script `fast-make.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 executar tots els jocs de proves i veure si els passeu, executeu `../avalua.sh`.
- 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**.