

Compiladors: Examen de laboratori.

11 de juny de 2018

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.

ATENCIÓ: Al Racó trobareu els jocs de proves i codi necessari per a fer l'examen. Llegiu les instruccions del final de l'enunciat per veure d'on descarregar-lo.

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 següents.

1 Tipus pair (1.5 punts)

Volem afegir a l'ASL el tipus **pair**, que permet crear tuples de dos elements de tipus bàsics. La tupla es delimita amb les paraules clau **pair** i **endpair**, entre les quals hi ha dos tipus bàsics separats per coma. Els elements del pair es poden accedir amb els operadors **.first** i **.second**. Per exemple:

```
func main()
  var p : pair int, bool endpair
  var x : int

  p.first = 5; p.second = true;
  x = 10;
  while (p.first <= x*2) do
    p.second = not p.second;
    p.first = p.first + 1;
  endwhile
endfunc
```

Joc de proves 1 (0.5 punts). Començarem modificant la gramàtica per tal d'admetre declaracions de variables de tipus **pair**. També caldrà modificar el **SymbolListener** perquè creï els tipus adequats usant el **TypeManager** modificat que us proporcionem, i inserti les noves variable a la taula de símbols. El nou **TypeManager** inclou els mètodes:

```
TypeId createPairTy      (TypeId firstType, TypeId secondType);
bool   isPairTy          (TypeId tid) const;
TypeId getFirstPairType  (TypeId tid) const;
TypeId getSecondPairType (TypeId tid) const;
```

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

```
func main()
  var p : pair int, bool endpair
  var x : int

  x = 10;
  if (p+1 != x/10) then
    x = p;
  endif
  p = 25;
endfunc
```

ha de produir els errors:

```
Line 6:7 error: Operator '+' with incompatible types.
Line 7:7 error: Assignment with incompatible types.
Line 9:4 error: Assignment with incompatible types.
```

Joc de proves 2 (0.5 punts). A continuació afegirem les operacions `.first` i `.second` per accedir als elements del parell: Cal ampliar la gramàtica per acceptar aquests operadors darrera un identificador, i afegir al `TypeCheckListener` les comprovacions i decoracions de l'arbre necessàries.

Us suggerim plantejar-ho amb una estructura similar a la del l'accés a un element d'un array, tant a la gramàtica com a la comprovació de tipus.

Així arribem al segon joc de proves: Que te com a sortida esperada:

| | |
|---|--|
| <pre>func main() var p : pair int, bool endpair var q : pair int, bool endpair var x : int var b : bool x = 10; while (p.first != not b) do x = 20*p.first + 2; b = p.second and not (x<0); if (p.second+3==0 and not p.first) then write "ok"; q = p; endif endwhile endfunc</pre> | <pre>Line 8:17 error: Operator '!=' with incompatible types. Line 11:17 error: Operator '+' with incompatible types. Line 12:27 error: Operator 'not' with incompatible types.</pre> |
|---|--|

Joc de proves 3 (0.5 punts). Finalment, ens queda tractar el cas en que l'accés a un element del parell s'usa com a valor referenciable. Igual que abans, podeu tractar-ho de forma anàloga a l'accés a un element d'un array, i resoldre el tercer joc de proves:

| | |
|--|--|
| <p>Tercer joc de proves:</p> <pre>func main() var p : pair int, bool endpair var x : int var b : bool x = 10; while (p.first != not b) do p.first = x + 2*p.first; if (p.second and x==10) then p.second = false; p.first = p.second or b; write "ok"; else p.second = p.second + 3; endif endwhile endfunc</pre> | <p>produeix els errors:</p> <pre>Line 7:17 error: Operator '!=' with incompatible types. Line 11:16 error: Assignment with incompatible types. Line 14:28 error: Operator '+' with incompatible types. Line 14:17 error: Assignment with incompatible types.</pre> |
|--|--|

Atenció: No es demana la generació de codi relativa al tipus `pair`.

2 Producte escalar (5.5 punts)

El segon exercici consisteix en dotar a l'ASL del producte escalar de vectors de nombres. El producte escalar de dos vectors a, b de mida n és $\sum_{i=1}^n a_i * b_i$. Només permetrem productes escalars de vectors amb el mateix tipus bàsic (és a dir, o bé ambdós vectors contenen enters, o bé ambdós contenen reals). Obviament, el resultat del producte serà del mateix tipus que els elements dels vectors.

Un exemple de codi:

```
func main()
  var a : array [10] of int
  var b : array [10] of int
  var c : array [10] of int
  var x : int
  var y : float

  x = a*b + 3
  y = x - b*c/(y-1)
endfunc
```

Observeu que denotem el producte escalar amb l'operador '*', que per tant esdevé un operador sobrecarregat. Això fa que no sigui necessari modificar la gramàtica, i que només calgui fer canvis a la comprovació de tipus.

Joc de proves 1 (0.5 punts). El primer pas serà tractar la sobrecàrrega: Quan en el `TypeChangeListener` trobem un operador de multiplicació, abans de res cal decidir si és una multiplicació normal o un producte de vectors. En el primer cas, procedirem normalment. En canvi, si es tracta d'un producte de vectors, cal comprovar que tots dos operands són vectors i ténen la mateixa mida.

| | |
|---|--|
| El primer joc de proves: | genera els errors: |
| <pre>func main() var a : array[10] of int var b : array[10] of int var c : array[12] of int var n : int n = a*b+1; n = 2*a*b; n = a*n; n = a*c; endfunc</pre> | <pre>Line 7:7 error: Operator '*' with incompatible types. Line 7:9 error: Operator '*' with incompatible types. Line 8:7 error: Operator '*' with incompatible types. Line 9:7 error: Operator '*' with incompatible types.</pre> |

Joc de proves 2 (1 punts). A continuació afegirem la comprovació que els elements dels vectors son numèrics i del mateix tipus (ambdós int o ambdós float).

| | |
|---|---|
| Usarem el següent codi: | que ha de donar la sortida: |
| <pre>func main() var a : array[10] of int var b : array[10] of int var c : array[10] of float var d : array[10] of float var e : array[10] of char var f : array[10] of char var n : float n = a*b; n = a*c*2; n = c*d - 1; n = e*f; n = a*e; endfunc</pre> | <pre>Line 10:7 error: Operator '*' with incompatible types. Line 12:7 error: Operator '*' with incompatible types. Line 13:7 error: Operator '*' with incompatible types.</pre> |

Joc de proves 3 (1 punts). Finalment, queda assegurar-nos que el tipus del resultat del producte escalar (int o float segons siguin els elements dels vectors), es propaga correctament en les decoracions:

| | |
|--|--|
| El codi d'aquest joc de proves: | ha de generar els errors: |
| func main() var a : array[10] of int var b : array[10] of int var c : array[10] of float var d : array[10] of float var n : int var x : float n = a*b*2; n = 2*a*b; x = c*d - 1; n = a*c; x = b*d; endfunc | Line 9:7 error: Operator '*' with incompatible types. Line 9:9 error: Operator '*' with incompatible types. Line 11:7 error: Operator '*' with incompatible types. Line 12:7 error: Operator '*' with incompatible types. |

Joc de proves 4 (1 punts). Un cop ja tenim les comprovacions semàntiques del producte escalar, podem passar a generar el codi necessari. Començarem per generar codi que faci el producte de vectors d'enters. Una operació de producte escalar en ASL generarà un t-codi consistent en un bucle que multiplica els elements dos a dos i en va acumulant la suma.

| | |
|--|--|
| Per tant, per al joc de proves: | quan la tVM executi el codi generat, obtindrem la sortida: |
| func dotprod(x : array[10] of int, y : array[10] of int) : int var i,s : int i = 0; s = 0; while i < 10 do s = s + x[i]*y[i]; i = i+1; endwhile return s; endfunc func main() var a : array[10] of int var b : array[10] of int var i : int // omplir vectors // a = [1,2,3,4,5,6,7,8,9,10] // b = [20,18,16,14,12,10,8,6,4,2] i = 0; while i<10 do a[i]=i+1; b[9-i]=(i+1)*2; i = i+1; endwhile // calcular producte escalar "manualment" write "dotprod manual = "; write dotprod(a,b); write "\n"; // calcular producte escalar amb la nova operació write "dotprod operador = "; write a*b; write "\n"; endfunc | dotprod manual = 440 dotprod operador = 440 |

Joc de proves 5 (1 punts). A continuació extendrem la generació de codi per tractar també vectors de reals. Simplement cal tenir en compte que cal generar instruccions de suma i multiplicació real o entera segons sigui el tipus dels elements dels vectors.

Així, amb el codi:

```
func dotprod(x : array[10] of float,
            y : array[10] of float) : float
    var i : int
    var s : float
    i = 0;
    s = 0;
    while i < 10 do
        s = s + x[i]*y[i];
        i = i+1;
    endwhile
    return s;
endfunc

func main()
    var a : array[10] of float
    var b : array[10] of float
    var i : int

    // omplir vectors
    // a = [0.4,0.8,1.2,1.6,2,2.4,2.8,3.2,3.6,4]
    // b = [7,6.3,5.6,4.9,4.2,3.5,2.8,2.1,1.4,0.7]
    i = 0;
    while i<10 do
        a[i] = (i+1)/2.5;
        b[9-i]=(i+1)*0.7;
        i = i+1;
    endwhile
    write "\n";

    // calcular producte escalar "manualment"
    write "dotprod manual = ";
    write dotprod(a,b);
    write "\n";

    // calcular producte escalar amb nou operador
    write "dotprod operador = ";
    write a*b;
    write "\n";
endfunc
```

quan la tVM executi el codi generat, obtindrem la sortida:

```
dotprod manual = 61.6
dotprod operador = 61.6
```

Joc de proves 6 (1 punts). Per acabar, resta fer que la generació de codi de l'operador del producte escalar tingui en compte si els vectors són locals a la funció o són referències a vectors externs que s'han rebut com a paràmetre.

| | |
|---|---|
| Per fer-ho, useu el joc de proves: | Quan executem amb la tVM el codi generat, donant-li l'entrada: |
| <pre>func i_dotprod(x : array[10] of int, y : array[10] of int) : int return x*y; endfunc func f_dotprod(x : array[10] of float, y : array[10] of float) : float return x*y; endfunc func i_readvec(x : array[10] of int) var i : int; i = 0; while i<10 do read x[i]; i = i+1; done endfunc func f_readvec(x : array[10] of float) var i : int; i = 0; while i<10 do read a[i]; i = i+1; done endfunc func main() var a : array[10] of int var b : array[10] of int var c : array[10] of float var d : array[10] of float // omplir vectors i_readvec(a); i_readvec(b); f_readvec(c); f_readvec(d); // calcular producte escalar dins una funcio write "dotprod int funcio = "; write i_dotprod(a,b); write "\n"; write "dotprod float funcio = "; write f_dotprod(c,d); write "\n"; // calcular producte escalar de vectors locals write "dotprod int local = "; write a*b; write "\n"; write "dotprod float local = "; write c*d; write "\n"; endfunc</pre> | <pre>1 2 3 4 5 6 7 8 9 10 2 4 6 8 9 8 7 6 5 1 1.1 2.2 3.4 4.4 5.5 6.6 7 8 9 11 2.1 4.3 5.4 6.6 1.1 2.2 7 9 1 2.8</pre> <p>obtindrem el resultat:</p> <pre>dotprod int funcio = 305 dotprod float funcio = 240.54 dotprod int local = 305 dotprod float local = 240.54</pre> |

Informació important

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

- Codi necessari per a l'examen
 - **TypesMgr.cpp** i **TypesMgr.h** : Gestor de tipus ampliat amb el tipus necessaris per a l'examen. Copieu-lo al directori **common** i no el modifiqueu.
 - **SemErrors.cpp** i **SemErrors.h** : Gestor d'errors modificat amb els errors necessaris per a l'examen. Copieu-lo al directori **common** i no el modifiqueu.
- **jps**: Subdirectori amb jocs de proves (**jpXX.asl**), i la seva corresponent sortida esperada (**jpXX.err** per als jocs de proves de validació semàntica, **jpXX.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 a l'executar-lo.
- **evaluator.sh**: Script que executa tots els jocs de proves i compara els resultats amb la sortida esperada.
- **empaqueta.sh**: Script que crea un fitxer **examen-nom.cognom.tgz** amb la vostra solució. Aquest és el fitxer que cal pujar al Racó.