

Compilers: midterm laboratory exam.

April 9th 2024

ATTENTION: In Racó you will find the test code examples required to take the exam. The package contains modified versions of the modules `common/SemErrors.*` and `common/TypesMgr.*` with the methods needed to give new errors or manage the new types appearing in the exam exercises. *BEFORE YOU START DOING ANYTHING, read the instructions at the end of the statement to see how to download and install it.*

ATTENTION: The exam must be submitted in a `.tgz` file uploaded to the Racó. *Read the instructions at the end of the statement to see how to generate it.*

SCORE: The **first three points** of the lab exam grade are obtained with the test examples of the base practice. The rest are obtained by overcoming the exam-specific test examples. Correction is **automatic**, through the testing examples of this statement, plus an additional set of private testing examples.

IMPORTANT: The exam consists of two independent exercises. You can do them in any order. It is recommended to perform each exercise incrementally, resolving each test example before moving on to the next one.

1 Exception handling (3.5 points)

We want to add to the ASL a try-catch exception handling structure, which the following characteristics:

The structure encloses a group of statements in the `try` section. Statements inside `try` may include the statement `throw` followed by an expression.

- A `try` section including a sequence of statements.
- Statements inside `try` section may include the `throw` statement at any point.
- A `throw` statement is followed by an expression of a primitive type.
- A `catch` section including a list of cases. Each case consists of an expression and a sequence of statements.
- The expressions for cases in the `catch` section must be of primitive type, and all of them of the same type (int and float are allowed to be mixed though)

An example code:

```
1 func main()
2   var i,j : int
3   var x,y,z : float
4   var a : array [10] of float
5
6   try
7     i = 0;
8     while i<10 do
9       if y == 0 then
10        throw x*2;
11      else
12        a[i] = x/y;
13      endif
14    endwhile
15    if a[0]>a[9] then
16      throw -3;
17    else
18      if x!=y then
19        throw j+1;
20      endif
21    endif
22  catch
23    4*j-1 : a[0]=a[9];
24          x=0;
25    z-2 : i=1;
26          if a[i]!=2 then
27            x =1;
28          endif
29    -3 : write "ok";
30  endtry
31
32 endfunc
```

Test code 1 (1 point). The first step is to extend the grammar with the new structure try-catch and the new statement **throw**.

At the moment, we are not going to do any type checking on the **throw** and **catch** expressions yet, but the instructions inside the structure must be normally checked.

The first test code:

```

1 func main()
2   var i,j : int
3   var x,y,z : float
4   var a : array [10] of float
5
6   try
7     i = 0;
8     while i<(10 and j!='a') do
9       if y+1 then
10        throw x*2;
11      else
12        a[x] = z/y;
13      endif
14    endwhile
15    if a[0]>a[9] then
16      throw -3;
17    else
18      if x[i]!=z+1 then
19        throw j+1;
20      endif
21    endif
22  catch
23    j-1 : a[0]=a[9]-k*z;
24    x=0;
25    i*2 : i=1;
26    if a[i-w*2]!=2 then
27      x = 1;
28    endif
29    -3 : write "ok";
30  endtry
31 endfunc

```

generates the following errors:

```

Line 8:12 error: Operator '<' with incompatible types.
Line 8:17 error: Operator 'and' with incompatible types.
Line 8:22 error: Operator '!=' with incompatible types.
Line 9:8 error: Instruction 'if' requires a boolean condition.
Line 12:13 error: Array access with non integer index.
Line 18:11 error: Array access to a non array operand.
Line 23:21 error: Identifier 'k' is undeclared.
Line 26:18 error: Identifier 'w' is undeclared.

```

Test code 2 (1 point). Next we will check that the expression in `throw` statements are of primitive type.

In the modified `common/SemErrors.*` for the exam you will find that the method `throwRequiresBasicType` that will emit the corresponding error message.

Thus, the second test code:

```
1 func main()
2   var i,j : int
3   var x,y,z : float
4   var a : array [10] of float
5
6   try
7     i = 0;
8     while i<10 and j!='a' do
9       if y+1 then
10        throw a;
11      else
12        a[x] = z/y;
13      endif
14    endwhile
15    if a[0]>a[j-2]*y then
16      throw (x>0 or y!=z+1);
17    else
18      if x[i]!=z+1 then
19        throw j+1/a[z-2];
20      endif
21      throw m;
22    endif
23  catch
24    'a' : a[0]=a[9]-k*z;
25        x=0;
26    'b' : i=1;
27        if a[i-w*2]!=2 then
28          x = 1;
29        endif
30    'Z' : write "ok";
31  endtry
32
33 endfunc
```

generates the error messages:

```
Line 8:21 error: Operator '!=' with incompatible types.
Line 9:8 error: Instruction 'if' requires a boolean condition.
Line 10:4 error: Basic type required in 'throw'.
Line 12:13 error: Array access with non integer index.
Line 18:11 error: Array access to a non array operand.
Line 19:23 error: Array access with non integer index.
Line 21:7 error: Identifier 'm' is undeclared.
Line 24:21 error: Identifier 'k' is undeclared.
Line 27:18 error: Identifier 'w' is undeclared.
```

Test code 3 (1 point). We will now typecheck the expressions in the cases inside the `catch` section. All must be of a primitive type, and all must be of the *same* type.

In the modified `common/TypesMgr.*` for the exam you will find methods `allPrimitiveType` and `allSameType` that will check a list of `TypeIds`.

Important: To achieve that these errors are reported in the appropriate line, give the `catch` token position as argument to the error methods you'll find in `common/SemErrors.*`; that is, use `catchCasesRequireBasicTypes(ctx->CATCH())` and `catchCasesRequireCompatibleTypes(ctx->CATCH())`.

Thus, the third test code:

```

1 func f(x: int)
2   var b : array [5] of bool
3   try
4     b[1] = x-1;
5     throw b[x-2];
6   catch
7     b : write "error";
8     b[5] : write "crash";
9   endtry
10
11   return x*2;
12 endfunc
13
14 func main()
15   var i,j : int
16   var x,y,z : float
17   var a : array [10] of float
18
19   try
20     i = 0;
21     if a then
22       if y+1 then
23         throw a + b[0];
24       else
25         a[x] = z/y;
26         throw k;
27       endif
28     endif
29     if a[0]>a[j-2]*y then
30       throw (x>0 or y)!=z+1;
31     else
32       if x[i]!=z+1 then
33         throw f;
34       endif
35     endif
36   catch
37     'a' : a[0]=a[9]-k*z;
38         x=0;
39     2*j+1 : i=1;
40           if a[i-w*2]!=2 then
41             x = 1;
42           endif
43     'Z' : write "ok";
44   endtry
45 endfunc

```

produces the messages:

```

Line 4:11 error: Assignment with incompatible types.
Line 6:3 error: Basic types required in 'catch' cases.
Line 11:3 error: Return with incompatible type.
Line 21:6 error: Instruction 'if' requires a boolean condition.
Line 22:9 error: Instruction 'if' requires a boolean condition.
Line 23:12 error: Operator '+' with incompatible types.
Line 23:14 error: Identifier 'b' is undeclared.
Line 25:14 error: Array access with non integer index.
Line 26:11 error: Identifier 'k' is undeclared.
Line 30:20 error: Operator 'or' with incompatible types.
Line 30:25 error: Operator '!=' with incompatible types.
Line 32:12 error: Array access to a non array operand.
Line 33:12 error: Basic type required in 'throw'.
Line 36:3 error: Compatible types required in 'catch' cases.
Line 37:24 error: Identifier 'k' is undeclared.
Line 40:21 error: Identifier 'w' is undeclared.

```

Test code 4 (0.5 points). Finally, we are going to allow that `int` and `float` expressions are mixed in the `catch` section.

The modified `common/TypesMgr.*` for the exam contains a new method `allNumericType` which might be useful for this exercise.

The next test code:

```

1 func f(x: int)
2   var b : array [5] of bool
3   try
4     b[1] = x-1;
5     throw b[x-2];
6   catch
7     b : write "error";
8     b[5] : write "crash";
9   endtry
10
11   return x*2;
12 endfunc
13
14 func main()
15   var i,j : int
16   var x,y,z : float
17   var a : array [10] of float
18
19   try
20     i = 0;
21     if a then
22       if y+1 then
23         throw a + b[0];
24       else
25         a[x] = z/y;
26         throw k;
27       endif
28     endif
29     if a[0]>a[j-2]*y then
30       throw (x>0 or y)!=z+1;
31     else
32       if x[i]!=z+1 then
33         throw f;
34       endif
35     endif
36   catch
37     z/2 : a[0]=a[9]-k*z;
38         x=0;
39     j+1 : i=1;
40         if a[i-w*2]!=2 then
41           x = 1;
42         endif
43     a[i]+j : write "ok";
44   endtry
45 endfunc

```

should produce the errors:

```

Line 4:11 error: Assignment with incompatible types.
Line 6:3 error: Basic types required in 'catch' cases.
Line 11:3 error: Return with incompatible type.
Line 21:6 error: Instruction 'if' requires a boolean condition.
Line 22:9 error: Instruction 'if' requires a boolean condition.
Line 23:12 error: Operator '+' with incompatible types.
Line 23:14 error: Identifier 'b' is undeclared.
Line 25:14 error: Array access with non integer index.
Line 26:11 error: Identifier 'k' is undeclared.
Line 30:20 error: Operator 'or' with incompatible types.
Line 30:25 error: Operator '!=' with incompatible types.
Line 32:12 error: Array access to a non array operand.
Line 33:12 error: Basic type required in 'throw'.
Line 37:24 error: Identifier 'k' is undeclared.
Line 40:21 error: Identifier 'w' is undeclared.

```

2 Array initialization expressions (3.5 points)

The second exercise consists of adding array expressions that can be used in an assignment to initialize a array. Correct array expression can be built as follows:

- a) A list of expressions in curly brackets, separated by commas, is an array expression (e.g. $\{4, i+1, a[i]-2\}$).
- b) A list in curly brackets can be repeated with the operator \wedge and an integer constant (e.g. $\{a, b+1\}^3$ is equivalent to $\{a, b+1, a, b+1, a, b+1\}$)
- c) Several lists in curly brackets, with or without repetition, can be concatenated with colons to create a valid array expression (e.g. $\{a, b\}^3 : \{1, 2, 3\} : \{i-1\}^2$ is equivalent to $\{a, b, a, b, a, b, 1, 2, 3, i-1, i-1\}$)

The type of the array expression is a array of the size given by the number of specified elements, containing the type of the specified elements. If the elements are of mixed types, the type of the whole array expression will be **error**. If the elements are mixed numeric types, the type of the array elements will be float. (e.g. The type of the expression $\{1, 2\}^2 : \{4, 5, 6\}$ will be **array [7] of int**. The type of $\{a\}^5$ will be **array [5] of char**. And the type of $\{1, 2.5\}^2 : \{6\}$ will be **array [5] of float**).

So we could write for example:

```
1 func main()
2   var b : array [10] of int
3   var a : array [6] of float
4   var i, j: int
5   var c : array [10] of char
6
7   b = {5}^10; // fill b with 10 fives
8   b = {i+1}^10; // fill b with 10 times the value of i+1
9
10  // fill a with 6 given values
11  a = {1, j/2.0, 4.1, 2.1*i/j, 44, i-2};
12
13  // fill c with {'a', 'a', 'a', 'b', 'e', 'd', 'x', 'z', 'x', 'z'}
14  c = {'a'}^3: {'b', 'e', 'd'}: {'x', 'z'}^2;
15
16  // fill b with other values
17  b = {j+2*i}^5: {i-3*j}^5;
18 endfunc
```

Test code 5 (0.5 points). The first step is to modify the grammar in order to add the required tokens and the rules to recognize array expressions, consisting of a list of expressions in curly brackets, separated by commas, e.g. {x+1, 2, z}.

Without any additional type checks yet, we can go through the first test code.

This test code:

```
1 func main()
2   var b : array [10] of int
3   var a : array [6] of float
4   var i,j: int
5   var x : float
6   var c : array [9] of char
7
8   z = x+1;
9   x[k] = 3;
10
11   b = {5,5,5,5,5,5,5,5,5,5};
12   b = {i+1,i+2,i+3,i+4,i+5,
13        i+6,i+7,i+8,i+9,i+10};
14
15   a = {1.0, j/2.0, 4.1,
16        2.1*i/j, x, x-2};
17   i = 0;
18   while i<10 do
19     write i; write " ";
20     write a[i]; write "\n";
21     j = i-2*a[j];
22   endwhile
23
24   c = {'a','a','a',
25        'b','e','d','b','e','d'};
26
27   if i*j then
28     x = x+y;
29     a[x] = 1;
30   endif
31
32   b = {j+2*i,j+2*i,j+2*i,j+2*i,
33        i-3*j,i-3*j,i-3*j,i-3*j,
34        0,0};
35 endfunc
```

generates the following errors:

```
Line 8:2 error: Identifier 'z' is undeclared.
Line 9:2 error: Array access to a non array operand.
Line 9:4 error: Identifier 'k' is undeclared.
Line 21:7 error: Assignment with incompatible types.
Line 27:2 error: Instruction 'if' requires a boolean condition.
Line 28:11 error: Identifier 'y' is undeclared.
Line 29:7 error: Array access with non integer index.
```


Test code 6 (0.5 points). Next, we will start with the type checks. First we will address the simplest case, a list of expressions.

The type of the resulting expression must be an array with size the number of elements in the list, and with element type the type of the expressions inside (assume you can use the type of the first element).

If some expression in the list is not of a primitive type, an error is emitted, and the resulting type is **error**. If the expressions in the list are not all of the same type, an error is emitted, and the resulting type is **error**.

Since the array assignment is already in the basic compiler, errors related to the assignment and not to the array expression itself should work out of the box.

In the modified `common/TypesMgr.*` for the exam, you will find new methods `allPrimitiveType` and `allSameType` that will check a vector of `TypeIds`. Error messages for new errors can also be found in module `common/SemErrors.*`.

We pass this test code:

```

1 func main()
2   var b : array [10] of int
3   var a : array [6] of float
4   var i,j: int
5   var x : float
6   var c : array [9] of char
7
8   z = x+1;
9   x[k] = 3;
10
11   b = {5,5,5,5,5,5,5,5,5,5};
12   b = {i+1,i+2,i+3,i+4,i+5,
13       i+6,i+7,i+8,i+9,i+10};
14   b = {1,2,3,4,
15       'a','b','c','d',
16       j>0,x+1};
17
18   a = {j/2.2,j/3.3,j/4.4,
19       j/5.5,j/6.6,j/7.7};
20   a = {j+1,j+2,j+3,j+4,j+5,j+6};
21   a = {'h','h','h'};
22
23   i = 0;
24   while i<10 do
25     write i; write " ";
26     write a[i]; write "\n";
27     j = i-2*a[j];
28   endwhile
29
30   c = {'a','b','c','d','e',
31       'f','g','h','i'};
32
33   if i*j then
34     x = x+y;
35     a[x] = 1;
36   endif
37
38   b = {a,a};
39   b = {c,c,c,c,c,c,c,c,c,c};
40   b = {b};
41 endfunc

```

generating errors:

```

Line 8:2 error: Identifier 'z' is undeclared.
Line 9:2 error: Array access to a non array operand.
Line 9:4 error: Identifier 'k' is undeclared.
Line 14:6 error: Compatible types required in 'array expression'.
Line 20:4 error: Assignment with incompatible types.
Line 21:4 error: Assignment with incompatible types.
Line 27:7 error: Assignment with incompatible types.
Line 33:2 error: Instruction 'if' requires a boolean condition.
Line 34:11 error: Identifier 'y' is undeclared.
Line 35:7 error: Array access with non integer index.
Line 38:6 error: Basic types required in 'array expression'.
Line 39:6 error: Basic types required in 'array expression'.
Line 40:6 error: Basic types required in 'array expression'.

```

Test code 7 (1 point). Next, we will deal with repetitions, where a list can be folded several times (e.g. $\{x+1, 2, z\}^3$ or $\{2\}^4$).

You need to extend the grammar to cover this new format of the array expressions, and adapt the type check to produce the right result type. The size of the array type for the expression will obviously be the number of elements of the list times the number of repetitions (e.g. expression $\{x+1, 2, z\}^3$ results on a list of length 9).

We pass this test code:

```

1 func main()
2   var b : array [10] of int
3   var a : array [6] of float
4   var i,j: int
5   var x : float
6   var c : array [9] of char
7
8   z = x+1;
9   x[k] = 3;
10
11  b = {1,2,3,4,5,6,7,8,9,10};
12  b = {i+1}^5;
13  b = {1,2,3,4,
14      'a','b','c','d',
15      j>0,x+1};
16
17  a = {1.1, 2.2, 3.3,
18      4.4, 5.5, 6.6};
19  a = {1.1, 2.2, 3.3}^2;
20  a = {true, x>0, i<10}^4;
21  a = {j+1}^6;
22  a = {'h'}^3;
23  a = {'h','i','j','k'};
24  a = {'h','i','j','k','l','m'};
25
26  i = 0;
27  while i<10 do
28    write i; write " ";
29    write a[i]; write "\n";
30    j = i-2*a[j];
31  endwhile
32
33  c = {'a'}^9;
34  c = {'a','b','c'}^3;
35  c = {'a','b',x}^3;
36  c = {'a','b','c'}^4;
37
38  if i*j then
39    x = x+y;
40    a[x] = 1;
41  endif
42
43  b = {a}^2;
44  b = {c}^10;
45 endfunc

```

generating errors:

```

Line 8:2 error: Identifier 'z' is undeclared.
Line 9:2 error: Array access to a non array operand.
Line 9:4 error: Identifier 'k' is undeclared.
Line 12:4 error: Assignment with incompatible types.
Line 13:6 error: Compatible types required in 'array expression'.
Line 20:4 error: Assignment with incompatible types.
Line 21:4 error: Assignment with incompatible types.
Line 22:4 error: Assignment with incompatible types.
Line 23:4 error: Assignment with incompatible types.
Line 24:4 error: Assignment with incompatible types.
Line 30:7 error: Assignment with incompatible types.
Line 35:6 error: Compatible types required in 'array expression'.
Line 36:4 error: Assignment with incompatible types.
Line 38:2 error: Instruction 'if' requires a boolean condition.
Line 39:11 error: Identifier 'y' is undeclared.
Line 40:7 error: Array access with non integer index.
Line 43:6 error: Basic types required in 'array expression'.
Line 44:6 error: Basic types required in 'array expression'.

```

Test code 8 (1 point). In this test, we will introduce the concatenation of expressions lists with the colon (e.g. $\{x+1, 2, z\}:\{3,4\}^4$).

Extend the grammar to cover this expressions, and adapt the type checking to properly check that the types of the elements in the lists are primitive types and that all elements are of the same type. If the lists have non primitive types or contain mixed types, the appropriate error is produced, as in previous steps.

The size of the resulting array type is the sum of the sizes of each fragment (taking into account the repetitions, if any). The type of the array elements is the type of the elements of the list (assume you can use the type of the first element).

With this, the following program:

```

1 func main()
2   var b : array [10] of int
3   var a : array [6] of float
4   var i,j: int
5   var x : float
6   var c : array [9] of char
7
8   z = x+1;
9   x[k] = 3;
10
11   b = {1,2,3,4}:{5,6,7}:{8,9,10};
12   b = {i+1}^2:{i-1}^3;
13   b = {1}^3:{'a'}:{j>0,x+1}^3;
14
15   a = {1.1, 2.2}:{3.3,4.4}^2;
16   a = {1.1, 2.2, 3.3}^2;
17   a = {true, x>0}:{i<10}^4;
18   a = {j+1}^6;
19   a = {'h'}^3;
20   a = {'h','i'}^1:{'j','k'};
21   a = {'h','i'}:{'j','k'}^2;
22
23   i = 0;
24   while i<10 do
25     write i; write " ";
26     write a[i]; write "\n";
27     j = i-2*a[j];
28   endwhile
29
30   c = {'a'}:{'b'}:{'c'}:
31     {'d'}:{'e'}:{'f'}:
32     {'g'}:{'h'}:{'i'};
33   c = {'a'}:{'b'}:{'c'}:{'d'}^3:
34     {'e'}:{'f'}:{'g'};
35   c = {'a'}:{'b'}:{'c'}:
36     {'d'}:{'e'}:{'f'}^2:
37     {'g'}:{'h'}:{'i'};
38   c = {'a','b','c'}^3;
39   c = {'a','b'}^2:{x+1};
40   c = {'a','b','c'}^4;
41
42   if i*j then
43     x = x+y;
44     a[x] = 1;
45   endif
46
47   b = {a}^2;
48   b = {c}^10;
49 endfunc

```

must output:

```

Line 8:2 error: Identifier 'z' is undeclared.
Line 9:2 error: Array access to a non array operand.
Line 9:4 error: Identifier 'k' is undeclared.
Line 12:4 error: Assignment with incompatible types.
Line 13:6 error: Compatible types required in 'array expression'.
Line 17:4 error: Assignment with incompatible types.
Line 18:4 error: Assignment with incompatible types.
Line 19:4 error: Assignment with incompatible types.
Line 20:4 error: Assignment with incompatible types.
Line 21:4 error: Assignment with incompatible types.
Line 27:7 error: Assignment with incompatible types.
Line 35:4 error: Assignment with incompatible types.
Line 39:6 error: Compatible types required in 'array expression'.
Line 40:4 error: Assignment with incompatible types.
Line 42:2 error: Instruction 'if' requires a boolean condition.
Line 43:11 error: Identifier 'y' is undeclared.
Line 44:7 error: Array access with non integer index.
Line 47:6 error: Basic types required in 'array expression'.
Line 48:6 error: Basic types required in 'array expression'.

```

Test code 9 (0.5 points). Finally, we will add the check for coercions. If the list combines numerical types (int and float), then it must be accepted, and the type of the array elements must be float.

In the modified `common/TypeMgr.*` contains a method `allNumericType` that will check a vector of `TypeIds`.

With this, the following program:

```

1 func ff(a: array [10] of int,
2       b: array [6] of float): float
3     var i,j : int
4
5     a = {1,2,3,4}:{5,6,7}:{8,9,10};
6     a = {1,2,3}^2:{4,5.1,6,7};
7     a = {1,2,3}^2:{4,5.1};
8
9     b = {i+j}^2:{i*2.1,j/b[1]}^2;
10    b = {i+j}^2:{i,j}^2;
11    b = {i+j}^2:{i,z};
12
13    return b[1]+j<0;
14 endfunc
15
16 func main()
17   var x : array [10] of int
18   var y : array [6] of float
19   var i,j: int
20   var x : float
21   var c : array [9] of char
22
23   y[0] = ff(x,y);
24
25   z = x+1;
26   x[k] = 3;
27
28   i = 0;
29   while i<10 do
30     write i; write " ";
31     write a[i]; write "\n";
32     j = i-2*a[j];
33   endwhile
34
35   if i*j then
36     x = x+y;
37     a[x] = 1;
38   endif
39
40   b = {true,a}^2;
41   b = {c}^10;
42 endfunc

```

must output:

```

Line 6:5 error: Assignment with incompatible types.
Line 7:5 error: Assignment with incompatible types.
Line 10:5 error: Assignment with incompatible types.
Line 11:5 error: Assignment with incompatible types.
Line 11:18 error: Identifier 'z' is undeclared.
Line 13:3 error: Return with incompatible type.
Line 20:6 error: Identifier 'x' already declared.
Line 25:2 error: Identifier 'z' is undeclared.
Line 25:7 error: Operator '+' with incompatible types.
Line 26:4 error: Identifier 'k' is undeclared.
Line 31:11 error: Identifier 'a' is undeclared.
Line 32:13 error: Identifier 'a' is undeclared.
Line 35:2 error: Instruction 'if' requires a boolean condition.
Line 36:7 error: Assignment with incompatible types.
Line 36:10 error: Operator '+' with incompatible types.
Line 37:5 error: Identifier 'a' is undeclared.
Line 37:7 error: Array access with non integer index.
Line 40:2 error: Identifier 'b' is undeclared.
Line 40:12 error: Identifier 'a' is undeclared.
Line 41:2 error: Identifier 'b' is undeclared.
Line 41:6 error: Basic types required in 'array expression'.

```

Important Information

FILES FOR EXAM: In Racó (`examens.fib.upc.edu`) you will find a `examen.tgz` file with the following content:

- `parcial-lab-CL-2024.pdf`: This document, with the statement and the instructions.
- `jps`: Subdirectory with test codes (`jp_chkt_XX.asl`), and its expected output (`jp_chkt_XX.err`).
- `common`: Subdirectory with auxiliary modules `SemErrors` and `TypesMgr` extended with code needed for the exam.
- `avalua.sh`: Script that runs all test codes and says whether or not they are produce the expected result.
- `empaqueta.sh`: Script that creates a `examen-USERNAME.tgz` file with your solution. This is the file to be uploaded to Racó.

STEPS TO FOLLOW:

- Make a copy of the folders `asl` and `common` of your practice in a new directory `examen`.

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

- Switch to the new `examen` directory, and unzip the `examen.tgz` Racó file:

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

This will extract the contents of the package, **adding** to your directory `examen` the files listed above.

IMPORTANT: Do it in the specified order (first a copy of your practice and then decompress `.tgz`). Doing this in reverse order will cause you to lack the required code in `common` and the JPs to be unsuitable.

- Work as usual in folder `examen/asl`.

```
cd asl  
make antlr  
make -j4  
...
```

(If the build is slow due to server overload, you can run the `fast-make.sh` script)

- To see the differences between the output of your `asl` and the expected output in a specific type check test code, you can do:

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

(You can ignore line “There are semantic errors: no code generated” generated by `main.cpp`)

- To run all test codes and see if are passed, run `../avalua.sh`.
- Run `../empaqueta.sh` to create the delivery file `../examen-USERNAME.tgz` which needs to be uploaded to the Racó.
Packages created without using this script will be graded as **NOT PRESENTED**.