# Exercise Compiler Construction (List 9)

Hans de Nivelle

13.01.2016

In the previous List 8, we studied how to typecheck $C$ expressions. Using this knowledge, let us go back to List 1, and see if we can lower the expressions:

1.
```
char tolower( char c )
{
   if( c >= 'A' && c <= 'Z' )      (1)
      c += 'a' - 'A';              (2)
}

void tolower( char* s )
{
   while( *s )                     (3)
   {
      *s = tolower( *s )           (4)
      s ++ ;                       (5)
   }
}
```

You may assume that `tolower` has type `func( void; pntr(char) )`

(a) Type check (using the notation, methods and algorithm from the slides) the statements in the functions above. The result should be an annotated AST with conversions inserted.

(b) Apply the algorithm **translate** in the slides to obtain an intermediate representation.

(c) Inline (at least) the basic functions in the result of the previous exercise. If everything went well, the result should be somewhat similar to the CLANG output.

2.
```
size_t length( char* s )
{
   size_t length = 0;      (1)
   while( * ( s ++ ))      (2)
      ++ length;           (3)
}
```

Do the same in the code above: **(1)** Typecheck the ASTs of the expressions 1,2,3 and insert the conversions. **(2)** Produce an intermediate representation using **translate**. **(3)** Use inlining on the basic functions to get something that looks reasonable.

3. Assume that `m` is declared as `m[ SIZE ][ SIZE ]`. Consider the statement `m[i][j] = 0.0;` As above, **(1)** typecheck the AST and insert the conversions. **(2)** Produce an intermediate representation using **translate**. **(3)** Apply inling in the result.

**This is not a programming exercise!**