**McMaster University**
**Department of Computing and Software**
**Dr. W. Kahl**



# CAS 707 — Formal Specification Techniques

7 March 2018

Individual solutions to the assignment question here are due **electronically** via subversion **before 11:00 a.m. on Monday, March 5**, **respectively (for AQ 6.2) on paper** to the instructor in class.

## Assignment Question 6.1 — Linked Lists

For this question, you may build on your solution to Assignment Question 5.1; minor adaptations to the question to your setting can be appropriate.

For the definitions of Assignment Question 5.1, some of you are apparently seeing different preprocessor behaviour than I am seeing (applying **#define** inside comments...) — a solution that hopefully works in both settings should be to replace the two separate definitions for *Nil* with a single constant definition:

```
const list Nil = NULL;
```

As ACSL pendant of Reynolds' list predicate you may use the following:

```
/*@ inductive isList{L} (list xs, \list <value_type> ys) {
      case isList_Nil{L}:
        isList(Nil, \Nil);
      case isList_cons{L}:
        ∀ list xs, \list <value_type> ys;
          \valid(xs) ⇒ isList(xs→tail, ys) ⇒ isList(xs, \Cons(xs→head, ys));
  }
*/
```

(a) Consider the following specification for the (unchanged) *cons* function (Don't expect WP to like it! Allocation is unimplemented in WP.):

```
/*@ allocates \result;
    assigns *\result;
    ensures \result ≡ NULL ∨
      (\valid(\result) ∧
       ∀ \list <value_type> ys;
                isList(xs, ys) ⇒ isList(\result, \Cons(x, ys)));
*/
nonEmptyList * cons(value_type x, list xs) { // NULL used as error
  nonEmptyList * result = malloc(sizeof(struct _cons));
  if (result)
    { result→head = x;
      result→tail = xs;
    }
  return result;
}
```

Discuss what difference it would make to add the following precondition:

```
  requires ∃ \list <value_type> ys; isList(xs, ys);
```

(b) Discuss how *isList* relates with material included in your Assignment 5 submission (if it does relate to anything).

(c) Specify and implement an *append* function

| **void** *append*(*list* ∗ *p*, *list* *ys*) |
|---|

that appends the second argument list destructively to the end of the first argument list, by only modifying the final *NULL* pointer of the first argument list.

(d) Specify and implement an *concat* function

| *list* *concat*( *list* *xs*, *list* *ys*) |
|---|

that returns the concatenation of *xs* and *ys* without changing those.

Try different implementations (iterative and recursive) and different refinements of the specification (I left part of the behaviour unspecified. . . ); feel free to use (appropriately specified, annotated, and documented) auxiliary functions.

(e) Strive to complete specification and implementation of *testAndInsert* as far as you haven't already done it:

Specify, implement, and annotate the following C function for **insertion into ordered lists**:

| **bool** *testAndInsert*(*list*∗ *p*, **value_type** *n*) |
|---|

Assuming that *p* contains a *reference* to a list with *head* fields in ascending order, the function call "*testAndInsert*(*p*, *n*)" returns a **bool** result indicating whether the list referenced by *p* contained *n* as an *element*, and if it did not, it modifies that list by inserting a new list container with *n* as *element*, such that the resulting list is again in ascending order.

**Assignment Question 6.2 — Reynolds: Chapter 2: Assertions**

(a) Read chapters 1 and 2 of the lecture notes of CS818A3-2011 by John Reynolds.

(b) Solve Exercise 1 of Chapter 2.

(c) Solve at least half of the cases of Exercise 2 of Chapter 2.

(d) Prove soundness for at least half of the rules of each item in Exercise 3 of Chapter 2.

(e) Start reading also chapter 3; in particular strive to get a first understanding of annotated specification (Section 3.3).

**This may be handed in hand-written and on paper!**

---

The code listings above have been produced by including, before `\begin{document}`, the following:

```
\usepackage{listings}
\usepackage{listingsACSL}
\lstset{%
  language=[ACSL]C,
  frame=single,
  identifierstyle=\slshape,
  columns=flexible}
```

Spencer pointed out that `columns=flexible` appears to be key to making `listingsACSL.sty` work without `\ensuremath` problems.