McMaster University
Department of Computing and Software
Musa Al-hassy

COMP SCI 3EA3 — Software Specification and Correctness

January 10, 2017

Exercise 2.1 — Learning about C

Attempt to implement any of the games from yesterday's lecture.
Doing so will teach you the fundamentals of loops and IO, and bring about some comfort with using the language.

Exercise 2.1 — Algebra

Yesterday the word *Monoid* popped-up in discussion. It will reoccur and so we use this oppertunity to *specify* some properties of operators. Using,

```
/*@ axiomatic ThreeUnspecifiedFunctions
{
  logic real f(real x);
  logic real g(real x, real y);
  logic boolean p(real x, real y);
}
*/
```

formalise the following properties; the first is done for you.

(a) f is an involution, or self-inverse:

```
/*@ axiomatic MyProperties
{
    lemma f_involutive: \forall real x; f( f(x) ) == x ;
}
*/
```

(b) g is associative, or we may drop parenthesis when it is written infix.

(c) g has an identity, or it has a "no-op" when construed as program composition.

An associtive operation with an identity is known as a *monoidal operation*!

(d) g is commutative, or the order of arguments is irrelevant.

(e) g has a zero, or it has an "abort" when construed as program composition.

(f) p has a bottom element, or it has a "negative infinity". (Dually, "positive infinity").

(g) p is reflexive, or everything is related to itself.

(h) p is transitive, or p supports catenation of paths when construed as an path-relationship.

(i) p is antisymmetric, or mutually contained items are identical –when p is construed as a containment relation.

A relation satisfying relexitivity, transitivity, and antisymmetry is known as a *partial order*!

Exercise 2.3 — Specify a C function and prove its implementation correct

(a) Specify the function

```
size_type subsetCounterexample( value_type* a, size_type m, value_type* b, size_type n )
```

that tries to find a counterexample that the array `a[0..m-1]` is a *subset* of the array `b[0..n-1]`, where `a` being a subset of `b` means that all elements of `a` also occur somewhere in `b`.

A counterexample for this is the index of an element of `a` that occurs nowhere in `b`.

For example, if `a` contains `[8,2,6,3]`, and `b` contains `[13,2,8,5,1,3]`, then element `6` at index `2` in `a` is the only counterexample, because `a[2]=6`, but `6` does not occur in `b`. So in this case, the function should return `2`.

If no counterexample can be found, the function should return `-1`.

(b) Add loop specifications to aid verification of your implementation.

(c) Write a `main` function as "driver" program to be able to run some tests, and to use value analysis.

Exercise 2.4 — Puzzle

What does the following program do? Add appropriate ACSL specfications.

```
void hehner(int* x, int* y)
{
  if (*x == 0)
  {
      *y = 1; *x = 3;
  }
  else
  {
      *x -= 1; *y = 7;
      hehner(x, y);
      *y *= 2; *x = 5;
  }
}
```

Exercise 2.5 — Assertions

(a) Formalise a partial correctness rule for the for-loop, similar to the while-loop rule.

(b) Render the following rule using ACSL notation, where possible,

$$\boxed{\{Q\}\, S\, \{R\} \;\Rightarrow\; \{Q \wedge P\}\, S\, \{R \wedge P\}}$$

Discuss the validity of this rule.

(c) Discuss How to place asserts about `goto`s.