McMaster University
Department of Computing and Software
M. Al - hassy

COMP SCI 3EA3 — Software Specification and Correctness

January 16, 2016

Name _____ Student Number _____

Special Instructions:

- This examination paper includes 3 pages (including this cover page) and 5 questions.
  You are responsible for ensuring that your copy of the paper is complete. Bring any discrepancy to the attention of your invigilator.
- This is a **closed book** examination.
  No books, notes, texts, calculator or academic aids of any kind are permitted.
- **Read each question completely and carefully** before answering it.
- Answer all questions.
- **In doubt, document!**

All the questions are closely related to or directly from the slides of the second lecture and Sheet 2!

Contents

1 The Main Purpose of This Class — 3 marks —

Define the term *correct-by-construction programming.*

Solution Hints:

Being the main objective of this class, this concept has been mentioned repeatedly and is as follows.

> *A program and its proof should be developed hand-in-hand, with the* proof *usually leading the way.*

Other similar statements, from the slides and lecture include:

- English requirements → Math specifications → Executable Code ;; All along, the proof and the code are developed hand in hand!
- A program together with its specification is a theorem expressing that the program satisfies the specification.
- Programming is a by-product of proving that the formula expressing the specfications (and English requirements) are true.
- A program is a theorem: `{G} S {R}` is precisely the theorem "starting with relation G, we can algorithmically manipulate the states of its variables in finitely many steps to arrive in a state satisfying R" —`S` plays the role of witnessing the truth of the theorem.
- Programming is *calculating* a solution to an inequation: we specify a problem as `{G} X {R}` and we solve for `X`, a desired program.

2 ACSL Formalisation — 2 marks —

Given,

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

Specify the following property in ACSL notation:

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

Solution Hints:

```
/*@ axiomatic MyProperties
{
  lemma p_antisymmetric: \forall real x, y; p(x , y) && p(y , x)  ==>  x == y ;
}
*/
```

3 Mental Execution — 1 marks —

What does the following program do?

```
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;
  }
}
```

That is, what is the precise relationship between x and y when the program is invoked and when it terminates.

Solution Hints:

- If the initial value of x is below zero, the program does not terminate.
- If the initial value of x is zero, the program terminates with the new values of x, y being $3, 1$ respectively.
- If the initial value of x is greater than zero, the program terminates with x set to 5 and y being 2 to the power of the initial value of x.

The last clause hints at overflow issues...

4 Membership — 3 marks —

The following membership algorithm returns true if an element `e` belongs to an array `a`. We find an index `i` witnessing this membership by looking at each index incrementally until it has been found.

Provide appropriate specifications for the `ensures` and `loop invariant` clauses.

Solution Hints:

Notice that the first two lines of the program body are irrelevant; indeed, they are useful artifacts due to a similarity with the invariant! Moreover, the invariant is hinted at above in English prose.

```
/*@
@ requires 0 < len;
@ requires \valid(a+(0..len-1));
@ assigns \nothing;
@ ensures \result <==> \exists int i; 0 <= i < len && a[i] == e;
*/
bool elem(int* a, int len, int e)
{
  int i = 0;
  //@ assert 0 <= i < len;

  /*@
  @ loop invariant (\forall int j; 0 <= j < i ==> a[j] != e);
  @ loop invariant 0 <= i <= len;
  @ loop assigns i;
  @ loop variant len  - i;
  */
  for(int i = 0; i < len ; i++)
    if (a[i] == e) return true;

  return false;
}
```

5 Sorting — 1 marks —

Using the guarded command notation we have learned so far —do-od, if-fi, and simultaneous assignment statements— write a program-fragment that sorts 4 integer variables: w, x, y, z.

Solution Hints:

Someone brought up the problem of sorting three variables in-class and it became a running example, so it is not at all more difficult to step up to four variables. The following program makes appropriate swaps as long as there are items out of order.

$\{true\}$
do
$\square\ x > w \to w, x := x, w$
$\square\ y > x \to x, y := y, x$
$\square\ z > y \to y, z := z, y$
od
$\{w \leq x \leq y \leq z\}$

The End