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 4 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!**

Contents

1 The Main Purpose of This Class — 3 marks — **1**

2 ACSL Formalisation — 2 marks — **2**

3 Mental Execution — 1 marks — **2**

4 Membership — 3 marks — **3**

5 Sorting — 1 marks — **4**

1 The Main Purpose of This Class — 3 marks —

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

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.

```
/*@ axiomatic MyProperties
{
    lemma p_antisymmetric:                                                    ; // FILL IN THE BLANK
}
*/
```

Here is some extra space for your random thoughts:

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.

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.

```
/*@
@ requires 0 < len;
@ requires \valid(a+(0..len-1));
@ assigns \nothing;
@ ensures                                                                    ; // FILL IN THE BLANK
*/
bool elem(int* a, int len, int e)
{
  int i = 0;
  //@ assert 0 <= i < len;

  /*@
  @ loop invariant                                                           ; // FILL IN THE BLANK
  @ loop invariant                                                           ; // FILL IN THE BLANK
  @ loop assigns i;
  @ loop variant len  - i;
  */
  for(int i = 0; i < len ; i++)
    if (a[i] == e) return true;

  return false;
}
```

Here is some extra space for your random thoughts:

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.

The End