McMaster University
Department of Computing and Software
Musa Al-hassy

COMP SCI 3EA3
Final Exam
Solution Hints

COMP SCI 3EA3 — Software Specification and Correctness

April 25, 2017

______________________________ ______________

Name Student Number

This examination paper includes 8 pages (including this cover sheet); it consists of 7 questions (on the first 8 pages), plus a Theorem List (on pages **??**–**??**). You are responsible for ensuring that your copy of the paper is complete. Bring any discrepancy to the attention of your invigilator.

SPECIAL INSTRUCTIONS:

- Make sure your name and student number are on all sheets.
- Do not separate the first 8 pages.
  You are allowed to separate the Theorem List pages ??–??.
- This is a **closed book** examination. No books, notes, texts, calculator or academic aids of any kind are permitted.
- Answer the questions in the space provided.
- **Read each question completely and carefully** before answering it.
- Answer all questions.
- You are **always allowed** to **introduce auxiliary definitions** and **prove auxiliary theorems**.
- **In doubt, document!**
- The marks add up to 100; the bonus question 7 is worth another three marks.

Contents

The following laws may be particularly useful in this examination,

"Abbreviation"	$a \leq x < b \quad\equiv\quad a \leq x \,\wedge\, x < b$	
"Linear Order Negation"	$x < b \quad\equiv\quad \neg(b \leq x)$	
"Interval Split"	$a \leq x < c \quad\equiv\quad a \leq x < b \,\vee\, b \leq x < c$	provided $a \leq b \leq c$
"if-fi Context"	$\textbf{if } x = y \textbf{ then } T\,x \textbf{ else } F\,y \textbf{ fi} \quad=\quad \textbf{if } x = y \textbf{ then } T\,y \textbf{ else } F\,y \textbf{ fi}$	
"if-fi Idempotency"	$\textbf{if } b \textbf{ then } s \textbf{ else } s \textbf{ fi} \quad=\quad s$	

1 Substitute It! — 15 marks —

Give the grammar for terms in the Backus-Naur Form presented in this class, then give the definition of substitution on terms, $\_[\_ := \_] : \mathsf{Term} \to \mathsf{Variable} \to \mathsf{Term} \to \mathsf{Term}$, by pattern matching on the first term.
Afterwards prove $t[x := x] \;=\; t$.

Solution Hints:
Recall that a term is a constant c, a variable named x, or a function symbol f of arity n applied to other terms.

$$t ::= c \mid x \mid f(t_1, \ldots, t_n)$$

With this in-hand, we define substitution by

$$\begin{aligned}
c[x := E] &= c && \text{for constant } c \\
y[x := E] &= \textbf{if } x \text{ and } y \text{ are the same variable } \textbf{then } E \textbf{ else } y \textbf{ fi} \\
f(t_1, \ldots, t_n)[x := E] &= f(t_1[x := E], \ldots, t_n[x := E])
\end{aligned}$$

Just as we defined the operation by induction, we prove the required property by induction: Two base cases and a inductive step.

Constant case $t = c$:

$$\begin{aligned}
& t[x := x] \\
= \;& \{ \text{ case assumption } \} \\
& c[x := x] \\
= \;& \{ \text{ definition of substitution on constants } \} \\
& c \\
= \;& \{ \text{ case assumption } \} \\
& t
\end{aligned}$$

Variable case $t = y$:

$$\begin{aligned}
& t[x := x] \\
= \;& \{ \text{ case assumption } \} \\
& y[x := x] \\
= \;& \{ \text{ definition of substitution on variables } \} \\
& \textbf{if } x \text{ and } y \text{ are the same variable } \textbf{then } x \textbf{ else } y \textbf{ fi} \\
= \;& \{ \text{ context: the then-branch has } x \text{ and } y \text{ indistinguishable, so we may replace } x \text{ with } y \text{ there } \} \\
& \textbf{if } x \text{ and } y \text{ are the same variable } \textbf{then } y \textbf{ else } y \textbf{ fi} \\
= \;& \{ \text{ conditional is idempotent: } \textbf{if } b \textbf{ then } s \textbf{ else } s \textbf{ fi} = s \} \\
& y \\
= \;& \{ \text{ case assumption } \} \\
& t
\end{aligned}$$

Function Application case $t = f(t_1, \ldots, t_n)$:
Assume the claim is true for terms t_i; that is

$$\text{Induction hypothesis: } \forall i : 1..n \bullet t_i[x := x] = t$$

then we calculate,

$$\begin{aligned}
& t[x := x] \\
= \;& \{ \text{ case assumption } \} \\
& f(t_1, \ldots, t_n)[x := x] \\
= \;& \{ \text{ definition of substitution on function application } \} \\
& f(t_1[x := x], \ldots, t_n[x := x]) \\
= \;& \{ \text{ induction hypothesis } \} \\
& f(t_1, \ldots, t_n) \\
= \;& \{ \text{ case assumption } \} \\
& t
\end{aligned}$$

2 The Other Half Of The Quadrivium — 10 marks —

1. Prove that any symmetric, associative, and idempotent operator '$\oplus$' is necessarily self-distributive:

$$x \oplus (y \oplus z) = (x \oplus y) \oplus (x \oplus z)$$

Be **explicit** about any properties of '$\oplus$' in your calculational proof.

Solution Hints:

As usual we may begin with the complicated side and simplify. However, we can "discover" this formula by starting at the simpler side and introducing a copy of one of the variables via idempotency:

$$\begin{array}{ll}
& x \oplus (y \oplus z) \\
= & \quad \{ \text{ Idempotency } \} \\
& (x \oplus x) \oplus (y \oplus z) \\
= & \quad \{ \text{ Associativity, twice } \} \\
& x \oplus \big((x \oplus y) \oplus z\big) \\
= & \quad \{ \text{ Symmetry and Associativity } \} \\
& (x \oplus z) \oplus (x \oplus y) \\
= & \quad \{ \text{ Symmetry } \} \\
& (x \oplus y) \oplus (x \oplus z)
\end{array}$$

2. Prove that provided x does not occur in P and R is non-empty, we have

"Superfluous Quantification for Idempotent $\oplus$": $(\oplus x \mid R \bullet P) = P$

As usual, a quantification operation is necessarily associative and symmetric; moreover in this context it is assumed to have an identity and to be idempotent.

Solution Hints:

$$\begin{array}{ll}
& \big(\oplus x \mid R \bullet P\big) \\
= & \quad \{ \text{ Identity of } \oplus \ \} \\
& \big(\oplus x \mid R \bullet P \oplus e\big) \\
= & \quad \{ \text{ Distributivity of } \oplus \text{ over } \oplus \text{ whose provisos are given} \\
& \quad\ \ \text{along with the result of the previous problem } \} \\
& P \oplus \big(\oplus x \mid R \bullet e\big) \\
= & \quad \{ \text{ Unit Body } \} \\
& P \oplus e \\
= & \quad \{ \text{ Identity of } \oplus \ \} \\
& P
\end{array}$$

3 An Algorithm of the Old Sages — 15 marks —

Produce an algorithm —necessarily with proof— that satisfies the following informal specification:

> "Assign `present` to be *true* iff a given element E is in the ordered (monotone) array $f[0..N-1]$."

Solution Hints:
It is implicit in the *informal* spec that the array-length is a natural number, and we're told the array is ordered,

$$G \quad : \quad N \geq 0 \quad \wedge \quad f \text{ monotone}$$

and the goal of the problem can be formalised as

$$R \quad : \quad present \equiv \big(\exists i : 0..N-1 \bullet E = f[i]\big)$$

We can walk-along f and if we reach the end before witnessing E, then we can set *present* to be *false*. Such a Linear Search is not a completely valid solution since the extra information about the array being ordered is not used and hints that we ought to look for a more efficient solution.

As before, our solution has the form

$$\text{"find } x \text{ with } E = f[x], \underline{\text{if}} \text{ possible"}; \text{"assign a value to } present\text{"}$$

Where the first piece can be refined as follows:

$$
\begin{array}{ll}
& \text{"find } x \text{ with } E = f\ x, \underline{\text{if}} \text{ possible"} \\
\Leftarrow & \{ \text{ pick a value for } x \text{ if } E \text{ is not in the image of } f \ \} \\
& \text{"find } x \text{ with } E = f\ x, \text{ if possible; otherwise } x = -1\text{"} \\
\Leftarrow & \{ \text{ let us look for the largest index at which } E \text{ occurs} \\
& \quad \text{and fictitiously } \textbf{pretend} \text{ that } f(-1) = E. \ \} \\
& x = (\uparrow\ i : -1..N-1 \mid f\ i \leq E) \\
\equiv & \{ \text{ Local Characterisation of Integer Extrema} \\
& \quad \text{with antitonicity proviso: For any } i, j, \\
& \quad \left[\begin{array}{ll} & f\ j \leq E \quad \Rightarrow \quad f\ i \leq E \\ \Leftarrow & \{ \text{ Transitivity } \} \\ & f\ i \leq f\ j \\ \Leftarrow & \{ f \text{ is monotone } \} \\ & i \leq j \end{array}\right. \\
& \quad \text{The non-empty proviso holds since } f(-1) = E \leq E. \\
& \quad \text{The finiteness proviso is clear since we're in the interval } -1..N-1. \\
& \quad \} \\
& -1 \leq x < N \quad \wedge \quad f\ x \leq E \quad \wedge \quad \neg\big(-1 \leq x+1 \leq N \wedge f\ (x+1) \leq E\big) \\
\Leftarrow & \{ \text{ Weakening and contraposition; and linear order negation } \} \\
& -1 \leq x < N \quad \wedge \quad f\ x \leq E \quad \wedge \quad E < f\ (x+1)
\end{array}
$$

This is the post-condition of binary search! Hence, we have

$$\text{"Predicate Binary Search"}[b\ m := f\ m \leq E]\ ;\ present := \textbf{if } 0 \leq x \textbf{ then } (f\ x = E) \textbf{ else } \mathit{false} \textbf{ fi}$$

4 Assign Me To The Moon — 15 marks —

Using the heuristic of "programming is a goal-orented activity", step-by-step construct an algorithm to quantify over an array. Formally, solve:

$$\{\ 0 \le N\ \}\ ?\ \{\ result\ =\ \big(\oplus\, i\ \mid\ 0 \le i < N\ \bullet\ f\ i\big)\ \}$$

Solution Hints:
Replacing constant N with a new variable and placing bounds on it yields invariant

$$P: \qquad result\ =\ (\oplus i\ \mid\ 0 \le i < n\ \bullet\ f\ i) \quad \wedge \quad 0 \le n \le N$$

which says "`result` holds the quantification *so far*". It is clear that P is initially truthified by the assignment $result, n := e, 0$ where e is the unit of $\oplus$. Next, if we have P and $N = n$ then we have the required goal R, whence we take as loop guard

$$B \quad : \quad N \neq n$$

Within the loop we have $P \wedge B$ which entail $N - n > 0$ and so we take $N - n$ as our bound function bf.

The bound is decreased by increasing n, so we consider incrementing it and *calculate* what must happen to our other variable. That is we solve for assignment E in the maintenance of the invariant: Assuming $P \wedge B$,

$$\begin{array}{ll}
 & P[result, n := E, n+1] \\
= & \\
 & E\ =\ (\oplus i\ \mid\ 0 \le i < n+1\ \bullet\ f\ i) \quad \wedge \quad 0 \le n+1 \le N \\
= & \quad \{\ \text{For the right conjunct,} \\
 & \qquad \left[\begin{array}{ll}
 & 0 \le n+1 \le N \\
 \Leftarrow & \quad \{\ \text{Transitivity}\ \} \\
 & 0 \le n \quad \wedge \quad n+1 \le N \\
 = & \quad \{\ \text{Integers are discrete}\ \} \\
 & 0 \le n \quad \wedge \quad n < N \\
 = & \quad \{\ \text{Strict inclusion and abbreviation}\ \} \\
 & 0 \le n \le N \quad \wedge \quad n \neq N \\
 = & \quad \{\ \text{Assumptions } P \wedge B\ \} \\
 & true
 \end{array}\right. \\
 & \qquad \} \\
 & E\ =\ (\oplus i\ \mid\ 0 \le i < n+1\ \bullet\ f\ i) \\
= & \quad \{\ \text{Split off term}\ \} \\
 & E\ =\ (\oplus i\ \mid\ 0 \le i < n\ \bullet\ f\ i) \oplus f\,(n+1) \\
= & \quad \{\ \text{Assumption } P\ \} \\
 & E\ =\ result \oplus f\,(n+1)
\end{array}$$

Hence, the invariant is maintained precisely when we also assign *result* to be E,
which we have calculated to be $result \oplus f\,(n+1)$. Whence,

$$\begin{array}{l}
result, n := e, 0 \\
;\mathbf{do}\ n \neq N \rightarrow result, n := result\ \oplus\ f\ \ n, n+1\ \mathbf{od}
\end{array}$$

5 Quantify! Justify! Edify! — 30 marks —

The "maximum segment sum" is specified by computing, for integer array A,

$$\uparrow p, q \mid 0 \le p \le q \le \mathsf{length}\, A \bullet (\Sigma x \mid p \le x < q \bullet A\, x)$$

By the quantifier nesting law, we could formalise this in `ACSL` as

```
\max(0, len, \lambda integer p;
               \max(p, len, \lambda integer q;
                              \sum(p, q-1, \lambda integer x; A[x])))
```

An alternative is to name the important quantifications and give explicit definitions for them —we can always do this since a quantification over an interval is just a recursively defined notation! For $n : \mathbb{N}$,

$$\begin{array}{lcl} \mathsf{max2D}\, n &=& (\uparrow p, q \mid 0 \le p \le q \le n \bullet S\, p\, q) \\ S\, p\, q &=& (\Sigma x \mid p \le x < q \bullet A\, x) \\ \mathsf{max1D}\, n &=& (\uparrow p \mid 0 \le p \le n \bullet S\, p\, n) \end{array}$$

Now it suffices to compute $\mathsf{max2D}\, n$ where $n = \mathsf{length}\, A$. As such, let us find a recursive definition of $\mathsf{max2D}$.

Taking $n = 0$ and simplifying we find $\mathsf{max2D}\, 0 = 0$, then using split-off term we obtain that $\mathsf{max2D}\,(n+1) = \mathsf{max2D}\, n \uparrow \mathsf{max1D}\,(n+1)$. Consequently, it seems we need to find a recursive definition of $\mathsf{max1D}$. The case $n = 0$ again simplifies to 0, whereas the inductive case is more involved.

— 15 marks — For $0 \le n$, prove $\mathsf{max1D}\,(n+1) = (\mathsf{max1D}\, n + A\, n) \uparrow 0$

Solution Hints:

$$\begin{array}{ll}
& \mathsf{max1D}\,(n+1) \\
= & \{ \text{ Definition and Split off term with proviso } 0 < n+1 \text{ following from } 0 \le n \ \} \\
& (\uparrow p \mid 0 \le p \le n \bullet S\ p\ (n+1)) \uparrow S\ (n+1)\ (n+1) \\
= & \{ \text{ For the right-most term, for any } m\text{:} \\
& \quad \begin{array}{ll}
& S\ m\ m \\
= & \{ \text{ Definition } \} \\
& \sum x \mid m \le x < m \bullet A\, x \\
= & \{ \text{ Linear Order Negation and Contradiction } \} \\
& \sum x \mid \mathit{false} \bullet A\, x \\
= & \{ \text{ Empty Range } \} \\
& 0
\end{array} \\
& \quad \text{Now take } m = n+1. \ \} \\
& (\uparrow p \mid 0 \le p \le n \bullet S\ p\ (n+1)) \uparrow 0 \\
= & \{ \text{ For the quantification body,} \\
& \quad \begin{array}{ll}
& S\ p\ (n+1) \\
= & \{ \text{ Definition } \} \\
& \sum x \mid p \le x < n+1 \bullet A\, x \\
= & \{ \text{ Split off term with } \underline{\text{proviso}}\ p < n+1 \text{ given by the context } \} \\
& (\sum x \mid p \le x < n \bullet A\, x) + A\, n \\
= & \{ \text{ Definition } \} \\
& S\, p\, n + A\, n
\end{array} \\
& \ \} \\
& (\uparrow p \mid 0 \le p \le n \bullet S\, p\, n + A\, n) \uparrow 0 \\
= & \{ \text{ Distributivity of `+' over } \uparrow \text{ with non-empty proviso holding since } 0 \le n \ \} \\
& \left((\uparrow p \mid 0 \le p \le n \bullet S\, p\, n) + A\, n\right) \uparrow 0 \\
= & \{ \text{ Definition of } \mathsf{max1D} \ \} \\
& (\mathsf{max1D}\, n + A\, n) \uparrow 0
\end{array}$$

— 3 marks — Use the previous discussion to fill in the following axiomatisation —the last one is done for you.

Solution Hints:

```
#define max(a,b) ((a) > (b) ? (a) : (b))

/*@ axiomatic MyMaxOps {
  @ logic integer max1D(int* A, integer len)                          ;
  @ logic integer max2D(int* A, integer len)                          ;
  @
  @ axiom base1: \forall int* A                                       ;
  @       max1D(A, 0) == 0                                            ;
  @ axiom splitOff1 : \forall int* A; \forall integer n               ;
  @       max1D(A, n + 1) == max( max1D(A, n) + A[n] , 0 )            ;
  @
  @ axiom base2 : \forall int* A                                      ;
  @       max2D(A, 0) == 0                                            ;
  @ axiom splitOff2 : \forall int* A; \forall integer n               ;
  @       max2D(A , n + 1) == max( max2D(A, n) , max1D(A, n + 1) );
  @ }
  @*/
```

— 12 marks — Provide appropriate specifications for the the following maximum segment sum program:

Solution Hints:

```
/*@ requires 0 <= len                ;
  @ requires \valid(A+(0..len-1))    ;
  @ assigns \nothing                 ;
  @ ensures \result == max2D(A, len);
  */
int kaldewaij(int* A, int len)
{
  int n , r , s;
  n = r = s = 0;

  /*
  @ loop invariant 0 <= n <= len   ;
  @ loop invariant r == max2D(A, n);
  @ loop invariant s == max1D(A, n);
  @ loop assigns n                 ;
  @ loop variant len  - n          ;
  */
  while( n != len )
  {
    s = max(s + A[n] , 0) ;
    r = max(r, s)         ;
    n = n + 1             ;
  }

  return r;
}
```

6 Knowing The Source Code — 15 marks —

One ought to be comfortable using a variety of notational languages —e.g., different programming languages!

As an analogue to the integral $\int_a^b f(x)\,dx$ of a traditional first-year education in *continuous* mathematics, let us introduce the "sum" operation as a *discrete* counterpart,

$$\sum_a^b f(x)\,\delta x \;=\; (+x : \mathbb{Z} \mid a \le x < b \bullet f\,x)$$

A host of familar laws from the continous setting also hold in the discrete setting. In-particular, prove

Solution Hints:

"Additivity": For $a \le b \le c$,

$$\sum_a^b f(x)\,\delta x \;+\; \sum_b^c f(x)\,\delta x \;=\; \sum_a^c f(x)\,\delta x$$

We begin at the complicated side and aim to simplify,

$\sum_a^b f(x)\,\delta x \;+\; \sum_b^c f(x)\,\delta x$

= { definition of "sum" operator, twice }

$\big(+x : \mathbb{Z} \mid a \le x < b \bullet f\,x\big)$
$+ \big(+x : \mathbb{Z} \mid b \le x < c \bullet f\,x\big)$

= { Range Split with provisio:

  $a \le x < b \;\wedge\; b \le x < c$
  = { abbreviation }
  $a \le x \;\wedge\; x < b \wedge b \le x \;\wedge\; x < c$
  = { Linear order negation }
  $a \le x \;\wedge\; \neg(b \le x) \wedge b \le x \;\wedge\; x < c$
  = { contradiction }
  $a \le x \;\wedge\; \mathit{false} \;\wedge\; x < c$
  = { zero of $\wedge$ }
  false

}

$\big(+x : \mathbb{Z} \mid a \le x < b \;\vee\; b \le x < c \bullet f\,x\big)$

= { interval split with $a \le b \le c$ }

$\big(+x : \mathbb{Z} \mid a \le x < c \bullet f\,x\big)$

= { definition }

$\sum_a^c f(x)\,\delta x$

"Fubini's Theorem": For $a \le b$ and $c \le d$,

$$\sum_a^b \left(\sum_c^d f(x,y)\,\delta x\right)\delta\,y \;=\; \sum_c^d \left(\sum_a^b f(x,y)\,\delta y\right)\delta\,x$$

We begin at the complicated side and simplify,

$\sum_a^b \big(\sum_c^d f(x,y)\,\delta x\big)\delta\,y$

= { definition of "sum" notation, twice }

$\big(+y : \mathbb{Z} \mid a \le y < b \bullet (+x : \mathbb{Z} \mid c \le x < d \bullet f\,x\,y)\big)$

= { Nesting }

$\big(+y, x : \mathbb{Z} \mid a \le y < b \;\wedge\; c \le x < d \bullet f\,x\,y\big)$

= { Dummy List Permutation and symmetry of $\wedge$ }

$\big(+x, y : \mathbb{Z} \mid c \le x < d \;\wedge\; a \le y < b \bullet f\,x\,y\big)$

= { Nesting }

$\big(+x : \mathbb{Z} \mid c \le x < d \bullet (+y : \mathbb{Z} \mid a \le y < b \bullet f\,x\,y)\big)$

= { definition of "sum" notation, twice }

$\sum_c^d \big(\sum_a^b f(x,y)\,\delta y\big)\,\delta x$

7 (Bonus) Why Are We Here? — 3 marks —

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

Solution Hints:

It's what we've been doing the whole term: Calculating programs from their specifications.

Have A Great Summer!