TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Student ID: .....

EXAMINATIONS — 2012

TEST

SWEN 224
Formal Foundations of Programming
MODEL ANSWERS AND COMMENTS

Time Allowed: 50 minutes

Instructions:
- There are 50 possible marks.
- Answer all questions in the boxes provided.
- If additional space is required you may use one of the spare pages, and indicate where your answer is in the box for that question.
- Write clearly and cross out rough working and anything else you don't want marked.
- Non-electronic foreign language dictionaries are allowed.
- Calculators ARE NOT ALLOWED (and not required).
- No other reference material is allowed.

Question	Topic	Marks	Achieved
1.	Understanding Assertions	12	
2.	Writing Assertions	12	
3.	Verification	12	
4.	Loops	14	
	Total	50	

Student ID: .....

Question 1. Understanding Assertions [12 marks]

Consider the following assertion:

$$\exists i : 1..|A| - 1, j : 1..|A| \bullet A[j] = A[i] + A[i+1]$$

where A is assumed to be an array of numbers, with indexes starting from 1.

(a) [8 marks] For each of the following arrays, say whether the assertion is true or false for that array:

(i) $A = (1, 2, 3)$

true

We can show that this assertion is true by taking $i = 1$ and $j = 3$, which gives $A[j] = A[3] = 3$, and $A[i] + A[i+1] = A[1] + A[2] = 1 + 2 = 3$.

(ii) $A = (5, 3, 15, 6)$

false

We can see that this assertion is false by observing that the possible values of i (i.e. 1, 2 and 3) give 8, 18 and 21 as the possible values of $A[i] + A[i+1]$, and since none of these is in A, there can be no value if j that satisfies the assertion.

(iii) $A = (0, 0)$

true

We can show that this assertion is true by taking $i = 1$ and $j = 1$, which gives $A[j] = A[1] = 0$, and $A[i] + A[i+1] = A[1] + A2] = 0 + 0 = 0$. Taking $i = 1$ and $j = 2$ would also work.

(iv) $A = (0)$

false

We can see that this assertion is false because there is no value of i between 1 and $|A| - 1$, since $|A| - 1 = 0$, so there can be no value satisfying the required condition.

This question was mostly done well, though a few students clearly still don't understand existential quantification. I didn't ask for explanations, but some of the things students wrote beside their answers suggested that although they got the right answer, they didn't really understand why their answer was correct.

(b) [4 marks] State in words what the assertion means.

Some element in the array is equal to the sum of some pair of consecutive elements.

The text at the start of Question 1 said "Consider the following assertion", and parts (a) and (b) both refer to that assertion. So part (b) is asking what *the given assertion* means, not what *an assertion* is. If you wrote something coherent about what an assertion is, you got half marks.

In explaining what the assertion means, you should aim to write something that is as clear as possible and as simple as possible in plain English, expressing the intuitive meaning of the assertion, not just paraphrase the mathematical notation (e.g. saying there is an i between 1 and the size of A minus 1 and a j between 1 and the size of A such that the element at j is the sum of the element at i and the element at $i + 1$). In this case, there is no need to talk explicitly about indexes or positions in the array — but I didn't penalise you for doing so.

Question 2. Writing Assertions [12 marks]

-9mm

Write an assertion (a mathematical/logical expression) to formalise each of the following statements.

(a) [3 marks]

The values of x and y are both greater than w and both less than or equal to z.

$$w < x \leq z \wedge w < y \leq z$$

A few people added quantifiers for w, x, y and z. The question talks about "the values of x, y and z", so it is about these values, and no quantifiers are needed.

(b) [3 marks] The values in array A are in strictly descending order.

$$\forall i : 1..|A| - 1 \bullet A[i] > A[i+1]$$
or
$$\forall i, j : 1..|A| \bullet i < j \Rightarrow A[i] > A[j]$$

Strictly descending means that every element (apart from the first) is less than the one before it. Apart from getting that right, the main thing is to get the range(s) of the quantified variable(s) right — and get the right quantifier.

(c) [3 marks] Some value in array A occurs only once.

$$\exists i : 1..|A| \bullet \forall j : 1..|B| \bullet i \neq i \Rightarrow A[i] \neq A[j]$$

This requires two quantifiers — it can't be done with only one, and using a third is redundant. You need to think very carefully about what you are saying when you have two quantifiers — it is very easy to end up not saying what you intended.

The question doesn't mention an x!!!

(d) [3 marks] Array B is a *rotation* of array A, where rotating an array means taking some number, say k between 0 and $|A| - 1$, of elements from the end of the array, moving the rest of the array up k places, and placing the removed elements at the front of the array. For example, $(1,2,3,4)$, $(4,1,2,3)$, $(3,4,1,2)$, $(2,3,4,1)$ are all rotations of array $(1,2,3,4)$.

$\exists k : 0..|A| \bullet \forall i : 1..|A| \bullet B[i] = A[i + k \bmod |A|]$
or
$\exists k : 0..|A| \bullet B[1..k] = A[|A| - k..|A|] \wedge$
$B[k+1..|B|] = A[1..|A| - k - 1]$
or
$\exists C, D \bullet A = C \frown D \wedge B = D \frown C$

This was meant to be a more challenging question! You need to existentially quantify k, because it can be any value between 0 and $|A| - 1$; then it's just a matter of saying that the first k elements of B are the same as the last k elements of A, and the last $|A| - k$ elements of B are the same as the first $|A| - k$ elements of A. This can be expressed in several different ways, some of which are shown above. Note how the use of mod allows the two parts to be combined, and how quantifying on strings allows us to avoid talking about k at all.

In tackling a problem like this, it is often helpful to use some kind of informal notation or diagram to describe the general case before attempting to formalise it. For example, you might think of the question as saying that is we have $A = a_1 \cdots a_k \, a_{k+1} \cdots a_n$, then we will have $B = a_{k+1} \cdots a_n \, a_1 \cdots a_k$. Alternatively, we can visualise the relationship between A and B like this:

$A =$ 

$B =$ 

Question 3. Verifying Simple Programs [12 marks]

For each of the following correctness assertions, give the verification condition required to demonstrate that the correctness assertion is valid.

You are not required to prove these verification conditions.

(a) [3 marks] $\{x \neq 0 \wedge y \neq 0\}\, x := x + y\, \{x \neq 0\}$

$x \neq 0 \wedge y \neq 0 \Rightarrow x + y \neq 0$

This condition is obtained by pulling the postcondition ($x \neq 0$) back through the assignment, replacing x by $x + y$, and requiring that the resulting condition ($x + y \neq 0$) is implied by the precondition ($x \neq 0 \wedge y \neq 0$).

An alternative, and equivalent, condition can be obtained using forward substitution, but you need to distinguish between the old and new values of x. The whole point of the backward substitution rule is that it avoids this complication.

(b) [6 marks]

$\{1 \leq k < |A| \wedge s = \sum_{i=1}^{k} A[i]\}$ **if** $A[k+1] \neq 0$ **then** $s := s + A[k+1]$ **fi** $\{s = \sum_{i=1}^{k+1} A[i]\}$

$1 \leq k < |A| \wedge s = \sum_{i=1}^{k} A[i] \wedge A[k+1] \neq 0 \Rightarrow s + A[k+1] = \sum_{i=1}^{k+1} A[i]$

$1 \leq k < |A| \wedge s = \sum_{i=1}^{k} A[i] \wedge \neg\,(A[k+1] \neq 0) \Rightarrow s = \sum_{i=1}^{k+1} A[i]$

In this case, we get two verification conditions, corresponding to the two possible outcomes of the **if** test. When the **if** test is true, the condition obtained by pulling the postcondition back through the assignment $s := s + A[k]$ must be implied by the conjunction of the precondition and the **if** condition. When the **if** test is false, since there is no **else** part, the postcondition must be implied by the conjunction of the precondition and the negation of

the **if** condition (which is equivalent to saying that the **else** part is a **skip** statement).

(c) [3 marks] $\{A[1] = 5 \wedge i = 1\}\, A[i] := A[i] + A[i]\, \{A[1] = 10\}$

$A[1] = 5 \wedge i = 1 \Rightarrow A[i := A[i] + A[i]][1] = 10$

In this case, we have to treat the assignment $A[i] := A[i] + A[i]$ as an assignment to A, i.e. $A := A[i : +A[i] + A[i]]$, so we replace A in the postcondition by $A[i := A[i] + A[i]]$ to get the condition that must be implied by the given precondition.

Since we have $i = 1$ in the precondition, this can be simplified to $A[1] = 5 \Rightarrow A[1 := A[1] + A[1]][1] = 10$, which can be further simplified to $A[1] = 5 \Rightarrow A[1] + A[1] = 10$, which in turn reduces to $5 + 5 = 10$, which is clearly true.

This question was done reasonably well, though a disappointing number of students gave no answer to one or more parts. Some of you clearly need to read the lecture slides to remind yourself what a verification condition is!

You were told you were not required to prove the verification conditions, so you shouldn't waste valuable time doing so.

Question 4. Loops [14 marks]

(a) [6 marks] The following loop sums the elements of an array A (indexed from 1), along with pre and postconditions shown as assertions:

assert $k = 0 \wedge s = 0$;
while $k < |A|$ **do**
  $k := k + 1$;
  $s := s + A[k]$
od;
assert $s = \sum_{i=1}^{|A|} A[i]$

The loop invariant is $0 \leq k \leq |A| \wedge s = \sum_{i=0}^{k} A[i]$.

State the verification conditions that must be proved in order to verify the loop.

You are not required to prove these verification conditions.

The invariant holds on entry.

$k = 0 \wedge s = 0 \Rightarrow 0 \leq k \leq |A| \wedge s = \sum_{i=0}^{k} A[i]$

The invariant is preserved by the loop body.

$0 \leq k \leq |A| \wedge s = \sum_{i=0}^{k} A[i] \wedge k < |A| \Rightarrow 0 \leq k \leq |A| \wedge s = \sum_{i=0}^{k} A[i]$

The postcondition holds on exit.

$0 \leq k \leq |A| \wedge s = \sum_{i=0}^{k} A[i] \wedge \neg (k < |A|) \Rightarrow s = \sum_{i=1}^{|A|} A[i]$

The first condition says that the precondition implies the loop invariant.

The second condition is obtained by pulling the loop invariant back though the loop body (first through $s := s + a[k]$, and then through $l := k + 1$) to get a condition that must be implied by the conjunction of the loop invariant and the loop test.

The third condition says that the conjunction of the loop invariant the the negation of the loop test implies the postcondition.

Note that you were asked to give verification conditions for a specific loop, so you need to show the actual conditions obtained by doing these substitutions, etc., not just describe their general form.

I did not expect conditions for proving termination to be given, but gave credit for them if they were given.

Termination is shown by giving a bound function, in this case $|A| - k$, which we can show is always non-negative and is decreased each time the loop body is executed.

(b) The following are three alternative ways of writing the algorithm given above.

In each case, give a loop invariant that could be used to verify the loop, and state any additional preconditions that may be required.

(i) [2 marks]

assert $k = |A| \wedge s = 0$;
while $k > 0$ **do**
  $s := s + A[k]$;
  $k := k - 1$
od;
assert $s = \sum_{i=1}^{|A|} A[i]$

$0 \leq k \leq |A| \wedge s = \sum_{i=k+1}^{|A|} A[i]$

(ii) [3 marks]

assert $k = 2 \wedge s = A[1]$;
while $k \leq |A|$ **do**
  $s := s + A[k]$;
  $k := k + 1$
od;
assert $s = \sum_{i=1}^{|A|} A[i]$

$2 \leq k \leq |A| \wedge s = \sum_{i=1}^{k-1} A[i]$

Need to add precondition $|A| > 0$

(iii) [3 marks]

assert $k = 1 \wedge s = 0$;
while $k \leq |A|$ **do**
  $s := s + A[k] + A[k+1]$;
  $k := k + 2$
od;
assert $s = \sum_{i=1}^{|A|} A[i]$

$1 \leq k \leq |A| + 1 \wedge odd(k) \wedge s = \sum_{i=0}^{k-1} A[i]$

Need to add precondition $even(|A|)$

In order to get these loop invariant, you have to understand what each of the loops is doing and how it works. Since they all sum the elements of an array, and you were given an invariant for a loop that sums the elements of an array in you should expect that the invariants required here would have a similar form to the one given in part (a).

The loop in (i) sums elements of A starting from the last element and working back to the first element, so at any stage s is the sum of the elements from $k+1$ up to the last element. In this case, k indicates the next element to be added, rather than the last one added (as it was in the loop in part (a)).

The loop in (ii) starts by initialising s to $A[1]$, and K records the next element to be added. The additional precondition is required to ensure that $A[1]$ exists.

The loop in (iii) adds elements two at a time, and again uses k to record the next element to be added. The additional precondition is required to ensure that we can sum the elements by adding the two at a time.

I was mostly concerned with whether you got the constraint on s about right, and realised when extra preconditions were needed, and overlooked a lot of minor errors in the ranges of k and i.

Student ID: .....

****