p2q9

~~p11q1~~

2000

LCP

Software Engineering II, Part 1a AND Diploma

Consider this program over integer variables:

```
k := K;
x := X;
z := 1;
while k <> 0 do  {k := k-1;  z := z*x}
```

Given that the loop invariant is $z \times x^k = X^K$, show that executing this proram stores the value of X^K in the variable z. [5 marks]

It is proposed to insert the following code just before the assignment `k:=k-1`:

```
while even(k) do  {k := k/2;  x := x*x}
```

State the loop invariant of this inner loop and show that the modified program still stores the value of X^K in z. [7 marks]

Briefly describe formal specification languages, top-down design and fault avoidance techniques, indicating their respective roles in a software development project. [8 marks]

Solution Notes

We must how that the invariant holds at the start. Here $z = 1$, $x = X$ and $k = K$ so trivially $z \times x^k = 1 \times X^K = X^K$. After executing the loop body, it still holds because $z \times x^k = z \times x \times x^{k-1}$. Upon termination we have $k = 0$, so $z = z \times x^0 = z \times x^k = X^K$. Termination requires $k \geq 0$ initially.

The inner loop has the same invariant as the outer one. It holds at the outset because it is at the start of the outer loop. It is preserved after an execution of the inner loop because $x^k = (x^2)^{(k/2)}$ provided the integer k is even. Therefore we still have $z \times x^k = X^K$ after the inner loop terminates and so the correctness argument for the outer loop is unchanged.

(The third part is just material from the notes, joined in an informed manner. What follows is an outline answer.) A *formal specification* uses a language such as Z, whose semantics is defined mathematically, to give a precise definition of what each basic system operation should do. Such specifications

are not essential, but have been shown to help elicit the more obscure points of the requirements. A thorough understanding of the requirements is a prerequisite of the design phase.

Top-down design consists of developing the program incrementally from dummy routines, or stubs. The program is always executable and this systematic approach should lead to a design that can be understood later.

Fault avoidance is a family of techniques to help prevent run-time errors. It is necessary even if the requirements have been specified formally and the design performed carefully by refinement. Such techniques include switching on compiler warnings, using assert statements, using library routines, avoiding needless code tuning.