1.1 Invariant: $s = \sum \text{pints}(n..N]$ and $0 \leq n \leq N$.

```
MODULE Total2;

   (* Read a finite sequence of no more than |MAX| integers from
      standard input and print their sum to standard output. *)

IMPORT In, Out;

CONST MAX = 10;

VAR n, N, x, s : INTEGER;
    pints : ARRAY MAX OF INTEGER;

BEGIN
    (* read the numbers *)
    N := 0;
    In.Int(x);
    WHILE In.Done DO
        pints[N] := x;
        N := N + 1;
        In.Int(x)
    END;

    (* calculate the total *)
    n := N; s := 0;
    WHILE n # 0 DO
        n := n - 1;
        s := s + pints[n];
    END;

    (* print the total *)
    Out.Int(s, 0); Out.Ln
END Total2.
```

1.2 Invariant: $m = \max \text{pints}[0..n)$ and $1 \leq n \leq N$.

```
MODULE Max;

IMPORT In, Out;

CONST MAX = 10;

VAR n, N, x, m : INTEGER;
    pints : ARRAY MAX OF INTEGER;

BEGIN
    (* read the numbers *)
    N := 0;
    In.Int(x);
```

```
    WHILE In.Done DO
        pints[N] := x;
        N := N + 1;
        In.Int(x)
    END;

    (* calculate the maximum *)
    n := 1; m := pints[0];
    WHILE n # N DO
        IF pints[n] > m THEN
            m := pints[n]
        END;
        n := n + 1
    END;

    (* print the maximum *)
    Out.Int(m, 0); Out.Ln
END Max.
```

1.3

```
MODULE Hit;

IMPORT In, Out;

CONST MAX = 10;

VAR n, N, x, m, h : INTEGER;
    pints : ARRAY MAX OF INTEGER;

BEGIN
    (* read the numbers *)
    N := 0;
    In.Int(x);
    WHILE In.Done DO
        pints[N] := x;
        N := N + 1;
        In.Int(x)
    END;

    (* calculate the maximum *)
    n := 0; h := 0;
    IF N > 0 THEN
    h := 1; m := pints[0]; n := 1
END;
    WHILE n # N DO
        IF pints[n] > m THEN
            m := pints[n]; h := h + 1
        END;
        n := n + 1
    END;
```

```
    Out.Int(h, 0); Out.Ln
END Hit.
```

Yes, as `IF N > 0` would not be needed.

1.4

```
MODULE DivMod;

IMPORT In, Out;

VAR x, y, q : INTEGER;

BEGIN
In.Int(x);
In.Int(y);
q := 0;

WHILE x >= y DO
q := q + 1;
x := x - y;
END;

WHILE x < 0 DO
q := q - 1;
x := x + y;
END;

Out.Int(q, 0); Out.Ln;
Out.Int(x, 0); Out.Ln;
END DivMod.
```

1.5

```
MODULE Sqrt;

IMPORT In, Out;

VAR x, y, t, l : INTEGER;

BEGIN
In.Int(y);

x := 0; t := 0; l := 0;
WHILE t <= y DO
IF l > 0 THEN
t := t + x;
l := l - 1;
ELSE
x := x + 1;
l := x;
t := 0;
END
END;

Out.Int(x - 1, 0); Out.Ln
```

```
END Sqrt.
```

2.1
```
MODULE Comp;

IMPORT In, Out;

VAR s, t : ARRAY 200 OF CHAR;
    equal, less : BOOLEAN;
    i : INTEGER;

BEGIN
In.Line(s);
In.Line(t);

equal := TRUE; less := FALSE;
LOOP
IF (s[i] = 0X) & (t[i] = 0X) THEN EXIT END;
IF s[i] > t[i] THEN
equal := FALSE;
EXIT
ELSIF s[i] < t[i] THEN
equal := FALSE;
less := TRUE;
EXIT
END;
i := i + 1
END;

IF equal THEN Out.String("Equal");
ELSIF less THEN Out.String("Less than");
ELSE Out.String("Greater");
END;
Out.Ln
END Comp.
```

2.2 (a)
```
pow x n m = pow' x n m 0 1
    pow' x n m j y
      | j == n = y
      | otherwise = pow' x n m (j + 1) (y * x `rem` m)
```

(b)
```
pow x n m = pow' x n m n 1
    pow' x n m i y
      | i == 0 = y
      | otherwise = pow' x n m (i - 1) (y * x `rem` m)
```

(c)
```
pow x n m = pow' x n m n 1 x
    pow' x n m i y z
      | i == 0 = y
      | i `rem` 2 == 0 = pow' x n m (i `div` 2) y (z * z `rem` m)
      | otherwise = pow' x n m (i - 1) (y * z `rem` m) z
```

(d)
```
pow x n m = pow' x n m n 1 x
    pow' x n m i y z
```

```
      | i == 0 = y
      | i `rem` 2 == 1 = pow' x n m ((i - 1) `div` 2) (y * z) (z * z `rem` m)
      | otherwise = pow' x n m (i `div` 2) y (z * z `rem` m)
```

(e)
```
-- identical to Power3
pow x n m = pow' x n m n 1 x
pow' x n m i y z
  | i == 0 = y
  | i `rem` 2 == 0 = pow' x n m (i `div` 2) y (z * z `rem` m)
  | otherwise = pow' x n m (i - 1) (y * z `rem` m) z
```

3.1
```
MODULE Gcd;

IMPORT In, Out;

PROCEDURE euclid (a, b : INTEGER; VAR x, y, d : INTEGER);
VAR xx, yy : INTEGER;
BEGIN
IF b = 0 THEN
d := a;
x := 1;
y := 0;
ELSE
euclid (b, a MOD b, xx, yy, d);
x := yy;
y := xx - (a DIV b)*yy;
END
END euclid;

VAR x, y, m, n, a : INTEGER;

BEGIN
In.Int(x);
In.Int(y);

euclid(x, y, m, n, a);

Out.Int(a, 0); Out.Ln;
Out.Int(m, 0); Out.Ln;
Out.Int(n, 0); Out.Ln;
END Gcd.
```

3.2
```
MODULE PrintLR2;

IMPORT Args, Conv, Out;

CONST Enough = 20;
     (* enough room for 2^63 as a decimal numeral *)

VAR
  t: INTEGER;
  buf: ARRAY 128 OF CHAR;
```

```
  n, i, u, x: INTEGER;
  d: ARRAY Enough OF INTEGER;

BEGIN
  Args.GetArg(1, buf);
  t := Conv.IntVal(buf);

  (* Pre: |t > 0| *)

  (* Method LR:  compute the digits from left to right *)

  (* Invariant: |n > 0  &  x = 10^n  &  x <= t| *)

  n := 1; x := 10;
  WHILE (t DIV 10) >= x DO
    n := n+1;
    x := 10*x
  END;

  (* |n > 0  &  x = 10^n  &  x <= t < 10*x| *)

  (* Invariant: |0 <= i <= n  &  0 <= u < x = 10^i  &
                t = u + 10 * x * val(d[i..n))| *)

  i := n; u := t;
  WHILE i # 0 DO
    i := i-1;
d[i] := u DIV x;
    u := u MOD x;
    x := x DIV 10;

    Out.Char(CHR(d[i] + ORD('0')))
  END;
  Out.Ln
END PrintLR2.
```

4.1

$$\frac{\{P \wedge B\}T\{Q\}\{P \wedge \neg B\}\text{SKIP}\{Q\}}{\{P\}\texttt{IF B THEN T END}\{Q\}}$$

4.2 $s[0..N-n) = s[n..N) \iff s[i] = s[i+n] \forall 0 \geq i < n \iff s$ recurs with period n.

```
MODULE Recur;

IMPORT In, Out, Strings;

VAR s : ARRAY 200 OF CHAR;
    i, pos, n : INTEGER;
    P : ARRAY 200 OF INTEGER;
```

```
BEGIN
In.Line(s);
n := Strings.Length(s);
pos := 0;
i := 1;
WHILE i < n DO
WHILE (pos > 0) & (s[pos] # s[i]) DO pos := P[pos]; END;
IF s[pos] = s[i] THEN pos := pos + 1; END;
P[i + 1] := pos;
i := i + 1;
END;

Out.Int(n-P[n], 0); Out.Ln;
END Recur.
```