Exercise 1: Values of variables and expressions in a data environment

1. Let the data environment d = [(x, **R**, 9.3), (y, **R**, 2), (z, **Z**, 3), (z, **Z**, 4)]. Determine the values of the following variables and expressions in d:

1.1: z
1.2: y
1.3: x
1.4: w
1.5: 6.6
1.6: x + 2*y
1.7: y↑z - x
1.8: y↑x - r
1.9: x>4*y
1.10: x>4*y **and** 2*y<z
1.11: 2*z+z/y

2. Let the data environment d = [(k, **Z**, 2), (y(0), **R**, 2), (y(1), **R**, 3.2), (j, **Z**, 1), (y(2), **R**, 5.1), (j, **Z**, 3), (y(3), **R**, 7.5), (y(2), **Z**, 9)]. Determine the values of the following variables and expressions in d:

2.1: y(k)
2.2: y(j+k)
2.3: y(k-j) + (j-k)
2.4: y(j-k)
2.5: y(k-j+x)
2.6: y(3*j - k) + y(2*j)
2.7: y(k/2)
2.8: y((j+k)/2)
2.9: y((j+k+1)/2)
2.10: y(k) < y(j+k)
2.11: y(k) ≥ y(j-k)
2.12: j ≤ k

3. A variable name (e.g. x) can be viewed as a function on 𝔻. What is the domain of this function? Write a Boolean expression (condition) which specifies the domain.

4. The expression

x/(y-3) > z **and** nm = "AP136"

is a combination of functions and can itself be viewed as a function on 𝔻. Draw a hierarchical (tree) diagram illustrating the way the various functions (/, -, >, **and** and =) are composed to form the function represented by the entire expression. Write a Boolean condition which specifies the domain of this function. What is the range of this function?

(end)

Exercise 2: Program statements as functions on ID

1. Let the data environment d = [(x, **R**, 9.3), (y, **R**, 2), (z, **Z**, 3), (z, **Z**, 4), (w, **Z**, 6)]. Determine the data environment A.d for each of the following assignment statements A:

1.1: x := 1.4*y + z
1.2: z := z + y
1.3: z := x + y
1.4: (z, w) := (y + z + w, 2*y + 3*z + 4*w)
1.5: (z, z) := (y + z + w, 2*y + 3*z + 4*w)
1.6: (z, z) := (y + z + w, y*w - 1)
1.7: (y, w) := (w, y)
1.8: y :=: w

2. Let the data environment d = [(k, **Z**, 2), (y(0), **R**, 2), (y(1), **R**, 3.2), (j, **Z**, 1), (y(2), **R**, 5.1), (j, **Z**, 3), (y(3), **R**, 7.5), (y(2), **Z**, 9)]. Determine the data environment A.d for each of the following assignment statements A:

2.1: y(j) := y(k) + 3
2.2: (j, y(j)) := (j + 1, y(k) + 3)

3. Let the data environment d = [(x, **R**, 9.3), (y, **R**, 2), (z, **Z**, 3), (w, **Z**, 6)]. Determine the data environment S.d for each of the following if statements S:

3.1: if x<0 then y := -x else y := x endif
3.2: if x>0 then y := -x else y := x endif
3.3: if r>0 then y := -x else y := x endif
3.4: if x>0 then r := -x else y := x endif
3.5: if x<0 then r := -x else y := x endif

4. Let the data environment d be as given in problem 3 above. Determine the data environment S.d for each of the following sequences S of statements.

4.1:
```
z := y + z + w
w := 2*y + 3*z + 4*w
```

4.2:
```
x := y*z
y := w
```

4.3:
```
y := w
x := y*z
```

4.4:
```
x := x + w*z
y := w
```

4.5:
```
y := w
x := x + w*z
```

4.6:
```
x := y*z
y := w
z := 2*x - w
```

5. Let the data environment d = [(x, **R**, 9.3), (y, **R**, 2)]. Determine the data environment D.d for each of the following declare statements D:

5.1: declare (z, **R**, z + y)

5.2: declare (z, **R**, x + y)

6. Let the data environment d be as given in problem 5 above. Determine the data environment S.d for each of the sequences S of statements below:

6.1:
```
declare (z, R, x + y)
release z
```

6.2:
```
declare (z, R, x + 2*y)
release y
```

6.3:
```
declare (z, R, x)
release x
```

6.4:
```
declare (z, R, x)
x:=y
y:=z
release z
```

7. Let the data environment d = [(i, **Z**, 0), (s, **R**, 0), (n, **Z**, 2), (x(1), **Z**, 1), (x(2), **Z**, 2)]. Let the while loop W be:

```
while i<n do
    i := i + 1
    s := s + x(i)
endwhile
```

Determine the data environment W.d by applying the formal definition of the while loop or the while lemmata directly.

(end)

Exercise 3: Preconditions and postconditions

1. Determine the complete preconditions:

1.1: {?} z := z + x {z ≤ max}
1.2: {?} x := z - y {x - y > 0}
1.3: {?} x := z - y {y - x > 0}
1.4: {?} x := 5 - z {w*y - 2*w↑2 < z}
1.5: {?} sum := sum + z {sum = x + y + z}
1.6: {?} y(m) := z {y(m) = y(n)}
1.7: {?} d(j) := a(k) {and$_{i=1}^{j-1}$ d(i) ≤ d(i+1)}
1.8: {?} if x < 0 then y := -x else y := x endif {y > 0}
1.9: {?} if x < 0 then y := -x else y := x endif {y ≥ 0}
1.10: {?} if x < 0 then y := -x else y := x endif {y < 0}
1.11: {?} if x < 0 then y := -x else y := x endif {y ≤ 0}
1.12: {?} if x < 0 then y := -x else y := x endif {2 ≤ y ≤ 4}
1.13: {?} if x < 0 then y := x else y := x - 2 endif {-1 ≤ y ≤ 4}

1.14: {?}
i := i + 1
sum := sum + x(i)
{i∈ **Z** and n∈ **Z** and i≤n and sum=$\sum_{j=1}^{i}$ x(j)}

1.15: {?}
x(gl) :=: x(gr)
gr := gr - 1
gl := gl - 1
{gl<gr≤ig and$_{i=gl+1}^{gr}$ x(i)=x(gr) and$_{i=gr+1}^{ig}$ x(gr)<x(i)}

2. Prove the following propositions, using the relevant proof rules.

2.1: {V1 and B} S1 {P} and {not B} S2 {P}
⇒ {V1} if B then S1 else S2 endif {P}

2.2: {B} S1 {P} and {V2 and not B} S2 {P}
⇒ {V2} if B then S1 else S2 endif {P}

2.3: {V1} S1 {P} and {V2} S2 {P}
⇒ {V1 and V2} if B then S1 else S2 endif {P}

3. Prove the following strict and complete versions of proof rule DC1, using the lemmata for the several types of preconditions:

3.1: {V1}S{P1} strictly and {V2}S{P2} strictly
⇒ {V1 and V2}S{P1 and P2} strictly

3.2: {V1}S{P1} completely and {V2}S{P2} completely
$\Rightarrow$ {V1 and V2}S{P1 and P2} completely

(end)

Exercise 4: While loops and loop invariants

1. Verify (i.e. prove) the following correctness proposition by applying proof rule W2 and other proof rules as required:

{n∈ **Z** and 0≤n}
  declare (sum, **R**, 0)
  declare (i, **Z**, 0)
  while i<n do
    i := i + 1
    sum := sum + x(i)
  endwhile
  release i
{sum = $\sum_{k=1}^{n} x(k)$}

for which the developer gave the following loop invariant:

n∈ **Z** and i∈ **Z** and 0≤i≤n and sum = $\sum_{k=1}^{i} x(k)$

Organize your proof clearly and systematically. Show each step in the decomposition process clearly and distinctly.

2. Complete the program segment below by deducing the missing parts denoted by a question mark (?) by suitable algebraic manipulation of the relevant Boolean expressions which would arise in the correctness proof (not by guessing or by trial and error). Use the loop invariant given below. Use the hypotheses of proof rule W2 as appropriate as the basis for your derivation of the missing parts. Show precisely how you derived each missing component of the program segment.

{n∈ **Z** and 0≤n}
  declare (sum, **R**, ?)
  declare (i, **Z**, ?)
  while ? do
    sum := ?
    i := i - 1
  endwhile
  release i
{sum = $\sum_{k=1}^{n} x(k)$}

loop invariant: n∈ **Z** and i∈ **Z** and 0≤i≤n and sum = $\sum_{k=i+1}^{n} x(k)$

3. Prove the correctness of your completed version of the program segment in problem 2 above for the given loop invariant. Again, organize your proof clearly and systematically and show each step in the decomposition process.

(end)

Exercise 5 and Design Assignment: Design of a while loop

Introduction

You are encouraged to consult with your colleagues on possible approaches to this design problem, but the final written solution which you hand in must be your own individual work.

The design problem

You are to design a subprogram which performs the function described below.

The one dimensional array X is given, with index values ranging from 1 to n inclusive. The set associated with the elements of array X is unspecified. A function prop(.) is available for use in expressions in the subprogram to be designed. This function maps the value of an element of array X to either true or false, depending upon whether or not that value satisfies a certain (but here unspecified) property.

The subprogram to be designed is to permute the values of the array elements X such that

- the values in the lower region of X (i.e. starting at X(1)) satisfy the property (i.e. the value of the function prop is true) and
- the values in the upper region of X (i.e. ending at X(n)) do not satisfy the property (i.e. the value of the function prop is false).

In addition, the subprogram to be designed must determine the value of a variable which identifies the boundary between the two regions of the array X.

Questions and tasks to do

1. What ambiguity or ambiguities in the above preliminary specification remain to be clarified in order to specify adequately the interface between the subprogram to be designed and a program calling it? How should such ambiguities be clarified in a real life situation? What does “adequately” mean in this context? For whom is the interface specification written?

2. Specify a postcondition for the subprogram outlined above. Illustrate your postcondition with a diagram showing the range of index values of the array X and the two regions of the array X.

3. Specify a precondition for the subprogram to be designed. The precondition may be an ordinary one (i.e. it is not necessary that it be strict). Illustrate your precondition with a diagram of the same type as that used to illustrate your postcondition.

4. Give the remaining information required in a specification of your subprogram. (Cf. the lecture notes, section 6.2 and the examples in section 5.3.1 and elsewhere.)

5. Identify more than one possible loop invariant. Illustrate each with a diagram of the same type as that used to illustrate your postcondition. Select one suitable loop invariant. Why did you select the one you did? What criteria did you use for making your choice? Hint: There are at least three feasible loop invariants, one or more of which are better in some sense than others.

6. Design a subprogram satisfying the precondition and postcondition and for your selected loop invariant. Base all design decisions on proof rules, relevant conditions (e.g. the loop invariant)

and other correctness considerations, **not** on intuitive or subjective considerations, e.g. about how the program operates. In your several design steps use the most efficient (least time consuming) combination of formal, semi-formal and informal approaches which you are confident will lead to a provably correct program. How did you determine (a) the loop condition, (b) the initialization and (c) the body of the loop?

7. Prove mathematically rigorously the correctness of your design. Organize your proof of correctness in such a way that it can be checked easily and straightforwardly by an independent reviewer whose time is being charged for. (I.e. the longer the time required by the reviewer, the lower your mark will be.)

(end)

Exercise 6: General review

1. Let the data environment d=[(x(1), **R**, 3.3), (x(2), **R**, 2), (x(3), **R**, 1), (k, **Z**, 5)]. Determine the following:

1.1: (x(x(k-2))).d
1.2: (release x(k-1)).d
1.3: (x(x(3)):=x(1)+k).d

2. Solve the following proof tasks. If a correctness proposition which is to be verified is not, in fact, true, deduce a counterexample (i.e. a test case which will illustrate the presence of the error) from the discrepancy in your attempted proof.

2.1: {V?} y(n):=y(2)+k {y(j)=w}

2.2: {ia≤na and ib≤nb+1 or ia≤na+1 and ib≤nb}
if ib>nb or ia≤na and A(ia)≤B(ib) then ia:=ia+1 else ib:=ib+1 endif
{ia≤na+1 and ib≤nb+1} ?

2.3: {ia≤na and ib≤nb+1 or ia≤na+1 and ib≤nb}
if ib>nb or A(ia)≤B(ib) then ia:=ia+1 else ib:=ib+1 endif
{ia≤na+1 and ib≤nb+1} ?

3. Using the lemma for a complete precondition, prove the complete version of proof rule IF1, i.e. the following

Theorem: If

{V and B} S1 {P} completely and
{V and not B} S2 {P} completely

then

{V} if B then S1 else S2 endif {P} completely

4. Design a program segment PS such that

1. {V} PS {P} and
2. (call PS).d = [(r, **Z**, .)] & d for every data environment d in the domain of PS

where V and P are given as follows. Justify your choice of each part of PS by a relevant proof rule and indicate which proof rule you applied.

V: n∈ **Z** and z∈ **Z** and 1≤z≤n+1

P: n∈ **Z** and z∈ **Z** and r∈ **Z** and 1≤z≤r≤n+1 and $\mathrm{and}_{i=z}^{r-1}$ x(i)=sp and (r=n+1 or x(r)≠sp)

(end)

Exercise 7: Monitoring a nuclear reactor

Introduction

You are encouraged to consult with your colleagues on possible approaches to this design problem, but the final written solution which you hand in must be your own individual work.

System environment: The safety monitoring and control system

During the operation of the nuclear reactor various variables (e.g. temperature, pressure, coolant flow rates, neutron flux, etc.) are measured at various locations in the reactor installation and are transmitted as electrical signals to the monitoring and control computer. These input variables are processed by the computer system. The results of these calculations control certain safety related functions as well as the display panel in the operational control room.

The safety system is subdivided into three subsystems: monitoring, control and display. Sensory and control data flow between the reactor and the control subsystem in both directions. For each measured variable a binary signal ("trip signal") from the reactor to the monitoring subsystem indicates whether the measured variable is within the normal range or not. Also for each measured variable a "veto" signal from the control subsystem to the monitoring subsystem indicates whether the corresponding trip signal should be ignored (e.g. because it has already been acted upon appropriately) or not. The monitoring subsystem processes the trip and veto signals and calculates the monitor mode and the state (off or on) of each display lamp. The monitor mode signal (one signal for the entire system) is required by the control subsystem. The display lamp signals (one for each measured variable) are required by the display subsystem.

The subprogram to be designed: the monitoring subsystem

The subject of this exercise is the specification and design of the subprogram for the monitoring subsystem (see section above). This subprogram calculates the monitor mode and the states of the display lamps. The input data to this subprogram consists of the following globally declared program variables: the trip signals, the veto signals, the display lamp signals and the monitor mode. The monitoring subprogram is called by the main program repeatedly and frequently in order to update the output signals.

The monitor mode is either normal or abnormal ("ok" or "trip"). The monitor mode should be set to "trip" whenever any trip signal is active while the corresponding veto signal is inactive (off). A display lamp should be switched on whenever the corresponding trip signal is active. Every output signal (output program variable) of this subprogram should latch, i.e. remain on once switched on. These variables will be reset if and as required by the control subsystem, not by the subprogram to be designed in this exercise.

Your task

Develop an unambiguous, mathematically precise specification of the externally observable behaviour of the subprogram to be designed (i.e. from the standpoint of the calling program). Include in your specification a *strict* precondition and a postcondition. Design a subprogram which implements that specification and prove its correctness formally.

The agreement of a number of safety engineers with your specification, in particular with your postcondition, must be obtained. Your specification should, therefore, be formulated and explained in a way appropriate to their backgrounds and experience. Select your notation, diagrams, etc. accordingly. These engineers are mainly nuclear and electrical engineers with significant experience with electrical and electronic instrumentation for nuclear reactors.

Note that the design of the entire system will be subjected to a thorough examination, analysis and verification procedure before the safety of the system is certified. Your design and documentation (including your proof of correctness) should, therefore, be presented in a form which will enable such an examination to be conducted as effectively and efficiently as possible, i.e. in minimum time and at minimum cost.

Questions:

1. How, in logical terms, is latching (leaving a signal on if already on) achieved in a digital electronic circuit in which the input is a trigger (turn on) signal and the output the latched signal? How can the same effect be realized most simply with a computer program variable?

2. Specify the interface between the subprogram to be designed and the calling program precisely and as completely as appropriate and necessary (e.g. in the form of correctness propositions). Specify a strict precondition and a postcondition for your subprogram. What type(s) of diagram(s) and/or other notation are most appropriate for presenting the meaning of your specification to the safety engineers and for obtaining their agreement?

3. Which variables can be modified by the execution of the subprogram to be designed? Which variables does the subprogram reference? (Give complete lists.)

4. If your subprogram contains a loop, state the loop invariant and the loop variant.

5. State your criteria and reasons for each design decision. What alternatives were possible for each design decision? Why did you reject them?

6. Design the complete subprogram.

7. How have you taken into consideration the requirement that your design, the presentation of it and your correctness proof be effectively and most efficiently verifiable?

References: (Answer all questions above first and only then refer to the following references.)

Bloomfield, Robin E. and Froome, Peter K.D., “The Application of Formal Methods to the Assessment of High Integrity Software”, *IEEE Transactions on Software Engineering*, 1986 September, Vol. SE-12, No. 9, pp. 988-993.

Bloomfield, R.E. and Ehrenberger, W.D., *Licensing issues associated with the use of computers in the nuclear industry*, EUR11147en, Commission of the European Communities, Directorate-General Telecommunications, Information Industries and Innovation, Luxembourg, 1987. See especially pages 183 and 200.

Fields, Bob and Elvang-Gøransson, Morten, “A VDM Case Study in mural”, *IEEE Transactions on Software Engineering*, 1992 April, Vol. 18, No. 4, pp. 279-295.

(end)

Exercise 8: Compressing a flight departure display

Introduction

You are encouraged to consult with your colleagues on possible approaches to this design problem, but the final written solution which you hand in must be your own individual work.

System environment

Flight departures are displayed in airport lounges and check-in areas on large specially designed boards with one line per flight. When a flight departs, its line is blanked out. From time to time, these blank lines are eliminated and succeeding non-blank lines moved up, freeing space at the bottom of the display boards. The space thus freed can then be used for data on later flights.

In the computer system which drives the display boards the flight data to be displayed are stored in the array D, with index values ranging from 1 to n inclusive, where n is the number of lines on the display boards. The data for each flight are stored in one array element. A second array A corresponds to the array D. The value of each array element A(i) indicates whether the data in D(i) is active (A(i)=true) or is inactive (A(i)=false), i.e. is to be considered logically blanked out because the flight has already departed.

The subprogram to be designed: compressing the flight data arrays

The subprogram *compress* to be designed should compress the data in the arrays D and A, i.e. move active lines to earlier elements in the arrays so that all inactive data lines are in consecutive elements at the ends of the arrays. In the process of compressing the arrays, the relative order (the sequence) of the active data lines should remain unchanged. The subprogram compress should return a value nact which indicates the number of active data lines in the final arrays.

Your task

Develop an unambiguous, mathematically precise specification of the externally observable behaviour of the subprogram to be designed (i.e. from the standpoint of the calling program). Include in your specification a strict precondition and a postcondition. Design a subprogram which implements that specification and prove its correctness formally.

The agreement of the airport authority's staff with your specification must be obtained before proceeding with the development of the subprogram. Your specification should, therefore, be formulated and explained in a way appropriate to their backgrounds and experience. Select your notation, diagrams, etc. accordingly. These staff members have some experience in specifying and developing application software. Some represent user departments; others, the data processing department. Some, but not all, have specialized software development knowledge.

Questions:

1. How, in the postcondition, can one express the requirement that the order of active data lines be the same before and after execution of the subprogram?

2. Specify the interface between the subprogram to be designed and the calling program precisely and as completely as appropriate and necessary (e.g. in the form of correctness propositions).

Specify a strict precondition and a postcondition for your subprogram. What type(s) of diagram(s) and/or other notation are most appropriate for presenting the meaning of your specification to the customer's staff in order to obtain their agreement?

3. Which variables can be modified by the execution of the subprogram compress? Which variables does the subprogram reference? (Give complete lists.)

4. If your subprogram contains a loop, state the loop invariant.

5. State your criteria and reasons for each design decision. What alternatives were possible for each design decision? Why did you reject them?

6. How have you taken into consideration the requirement that your specification and the presentation of it be understandable to the people for whom it is written?

(end)

Exercise 9: Generalizing a given subprogram

Introduction

You are encouraged to consult with your colleagues on possible approaches to this design problem, but the final written solution which you hand in must be your own individual work.

Background

An existing subprogram *mid* returns as its result a substring of the input string. Input variables are the string variable p1 and the integer variables p2 and p3. The result is returned as the value of the string variable rmid. The value of rmid is the substring of p1 which begins in position p2 and is p3 elements long. The variable rmid is not declared by mid. The positions in strings are numbered beginning with 1 (not 0).

The subprogram mid is very restrictive. Neither p1 nor rmid may be the null string (be of zero length). The input variable p2 must refer to a position within p1. The variables p2 and p3 together must refer to a substring which ends within p1, i.e. does not "go over the end" of p1. If any of these restrictions are violated, executing mid leads to a run-time error and abnormal termination.

Also available is a subprogram *len*. The value of its input variable p1 is a string. The value of its output variable rlen is an integer indicating the length of p1. The existing documentation does not indicate whether or not the subprogram len will handle an empty input string or not.

The subprogram to be designed: a generalized version of mid

The new subprogram midgen to be designed should have as weak a precondition as feasible. In particular, it should allow either p1 or rmid (or both) to be of zero length. Any integer values (positive, zero or negative) of p2 and p3 should be permitted. If the input variables have "erroneous" or "illogical" values the subprogram midgen should return the longest substring of p1 which would seem to make sense.

The new subprogram midgen is to replace mid. The input and output variables of both subprograms should, therefore, have the same names. The new subprogram midgen may use, i.e. call, mid. If it does, it must, of course, ensure that the restrictive strict precondition of mid is satisfied before such a call is executed.

Your task

1. Develop an unambiguous, mathematically precise specification of the externally observable behaviour of the **given** subprogram **mid** (i.e. from the standpoint of the calling program). Include in your specification a strict precondition and a postcondition.

2. Develop an unambiguous, mathematically precise specification of the desired behaviour of the **new** subprogram **midgen** to be designed. Include in your specification a strict precondition and a postcondition.

3. Show that the specification of midgen is more general than that of mid, i.e. that any (correct) call to mid may be replaced by a call to midgen. In other words, show that midgen satisfies mid's specification.

4. Finally, design midgen and prove that it satisfies its specification.

(end)

Exercise 10: Finding the zero of a function

Introduction

You are encouraged to consult with your colleagues on possible approaches to this design problem, but the final written solution which you hand in must be your own individual work.

Summary of your task

An existing subprogram *gcalc* calculates the value of a function g. Design the subprogram *zero* which determines the zero crossing of g using the subprogram gcalc.

The function g

The function g is a real function of several real parameters, only one of which, f, is of consequence in this exercise. The function g of f is continuous but not everywhere differentiable. It is not necessarily monotonic. It is known, however, that for all permissible values of the other parameters (1) g has exactly one zero (i.e. g.f=0 for one and only one value of f), (2) for sufficiently large positive values of f the value of g.f is positive and (2) for sufficiently large negative values of f the value of g.f is negative.

Typically the zero of g lies between f=-3 and f=+3. You may base design decisions which affect the efficiency of the subprogram zero on this observation, but the correctness of zero may **not** depend upon it.

The given subprogram gcalc

The result produced by the given subprogram gcalc approximates g.f. The inaccuracy of this result can be neglected for the purposes of this exercise, i.e. one can consider here that gcalc calculates the function g exactly.

The program variable pf is the input variable and the program variable resg is the result variable (output). Both are floating point variables. The output variable resg is not declared by gcalc. The values of the other input parameters are passed to gcalc via other program variables which must be declared and assigned values by the caller of zero or some superior subprogram.

The subprogram zero to be designed

The new subprogram zero to be designed should determine an approximation to the zero of g. The required accuracy of the result is given by the input variable d ($0<d<<1$). The result of the subprogram zero should be returned to the calling program as the value of the variable resf0.

When designing the subprogram zero strive for high executional speed (low time complexity).

Your task

1. Develop an unambiguous, mathematically precise specification of the externally observable behaviour of the given subprogram gcalc (i.e. from the standpoint of the calling program). Include in your specification a strict precondition and a postcondition.

2. Develop an unambiguous, mathematically precise specification of the desired behaviour of the

new subprogram zero to be designed. Include in your specification a strict precondition and a postcondition.

3. Finally, design zero and prove that it satisfies its specification. Prove that your iterative method converges and that any loop in the subprogram zero will always terminate.

Questions:

4. Have you interpreted the description of the variable d so that it represents absolute or relative accuracy? Why?

5. May the value of d be arbitrarily small? If yes, why? If no, what considerations limit its lower bound?

6. How can one ensure that an iterative process for locating a point with a unique property in an interval terminates (converges)?

(end)

Exercise 11: Semantics of a Pascal and a Basic program

1. Translate the following program written in Pascal. In your new formulation of this program use only program statements and compositions thereof that have been defined in the lectures and lecture notes (i.e. assignment statement, declaration, release, null statement, if statement, sequence of statements, while loop and subprogram call without formal parameter passing).

```
program printsquares;
procedure square(par1: integer; var par2: integer);
begin
   par2 := par1 * par1;
   par1 := -10
end;

var n, f: integer;
begin
   writeln(n, ' ', f, 'First line');
   n := 1;
   while n ≤ 10 do
     begin
       square(n, f);
       writeln(n, ' ', f);
       n := n + 1
     end
end.
```

2. If the second line above were changed to

```
procedure square(var par1: integer; var par2: integer);
```
[alternative A]

or to

```
procedure square(par1: integer; par2: integer);
```
[alternative B]

or to

```
procedure square(var par1: integer; par2: integer);
```
[alternative C]

how would your translated program be different?

3. Translate in the same way the following program written in Basic.

```
1000 REM Game of 13 matches
1010 N=13
1020 S=1
1030 W=1
2000 IF N≤0 THEN GOTO 4000
2010 GOSUB 5000
3000 N=N-W
3010 S=3-S
```

```
3090 GOTO 2000
4000 PRINT "End of game. Player";S;"won."
4010 END

5000 ...
...
5080 W=...
5090 RETURN
```

(end)