APPENDIX

Solutions to selected exercises

2.3 Exercises

1. Draw a JSP structure diagram of a student record, using only those elements specified in the following description: A student record consists of the name of the student, the student registration number, date of birth, course attended, year of course.

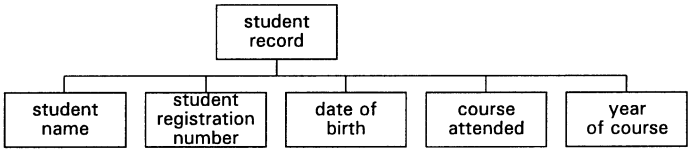

Figure A.1 Solution to Exercise 1.

2. Draw simple JSP structure diagrams for the following elements of the student record specified in Exercise 1:

(a) each student name is represented as last name followed by first name;
(b) student registration number is year of entry combined with a unique application number;
(c) date of birth is held as year, month and day (to facilitate sorting into age order).

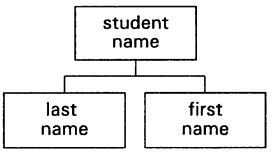

Figure A.2 Solution to Exercise 2(a).

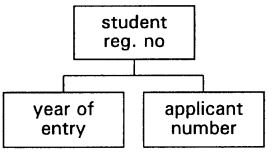

Figure A.3 Solution to Exercise 2(b).

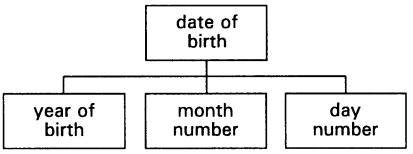

Figure A.4 Solution to Exercise 2(c).

3. Draw the full JSP structure diagram for the student record described in Exercise 1, incorporating the elements from Exercise 2.

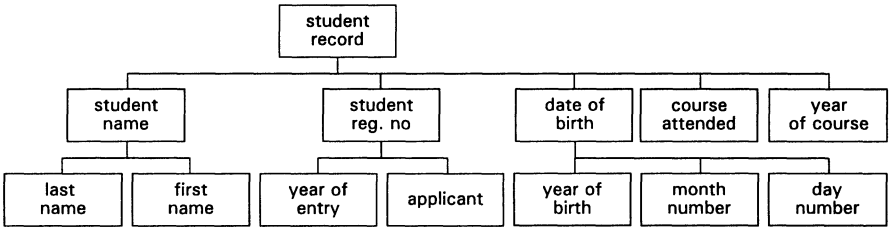

Figure A.5 Solution to Exercise 3.

4. Draw the JSP structure diagram for which the description is as follows. The academic year is a sequence of three terms: term 1, term 2 and term 3. Term 3 is a sequence of revision followed by an examination.

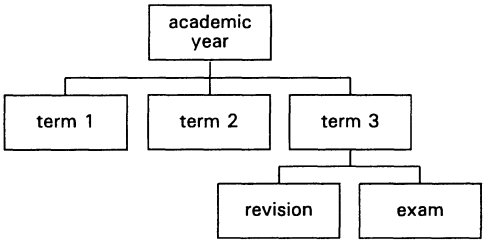

Figure A.6 Solution to Exercise 4.

5. Describe in words what the JSP structure diagram shown in Figure 2.10 (repeated as Figure A.7) represents.

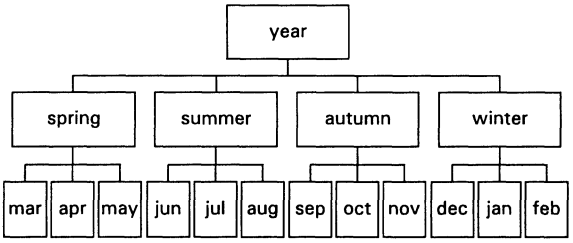

Figure A.7 Structure diagram for Exercise 5 (repeat of Fig.2.10).

A year is a sequence of the seasons Spring, Summer, Autumn and Winter. Spring is a sequence of the months March, April and May; Summer is a sequence of June, July and August. The season of Autumn is a sequence of September, October and November and Winter is a sequence of December, January and February.

The boxes Spring, Summer, Autumn and Winter are sequence components of the sequence structure year; each season is also a sequence structure with sequence components comprising the months of the season.

2.5 Exercises

1. A company employee may be either a technical specialist or an administrator. Express this as a JSP structure diagram.

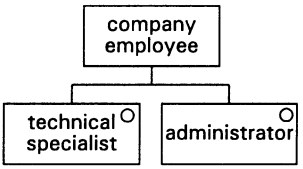

Figure A.8 Solution to Exercise 1.

2. A technical specialist within an organization may be a lawyer, an accountant, a systems analyst, an architect or a chemist. A company employee who is a chemist may work in the field of organic or inorganic chemistry. Express this as a JSP structure diagram.

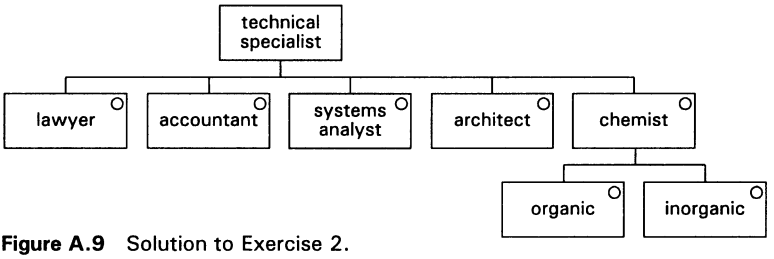

Figure A.9 Solution to Exercise 2.

Figure A.10 also represents the situation described above. Note that as the structure is a selection, the order of the selection components is not important.

See Figure A.10 (p. 274).

3. A company administrator could work in one of the following departments: personnel, payroll and pensions, registry and records, sales, marketing, or purchasing. Use a JSP structure diagram to describe this.

See Figure A.11 (p. 274).

4. Combine the details described in Exercises 1 to 3 to produce a single JSP diagram.

See Figure A.12 (p. 274).

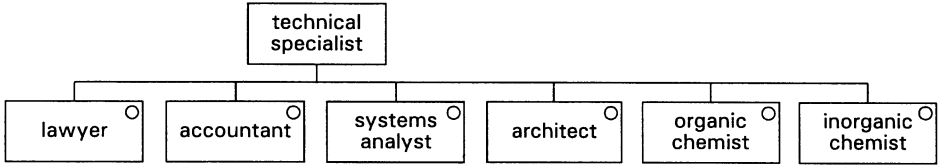

Figure A.10 Alternative solution to Exercise 2.

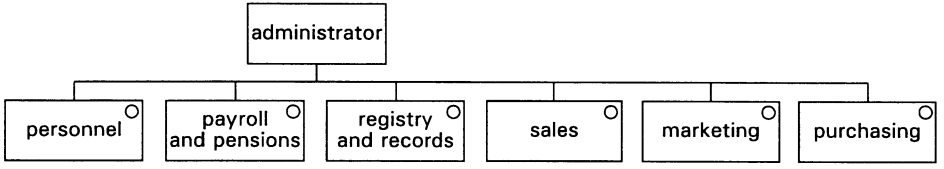

Figure A.11 Solution to Exercise 3.

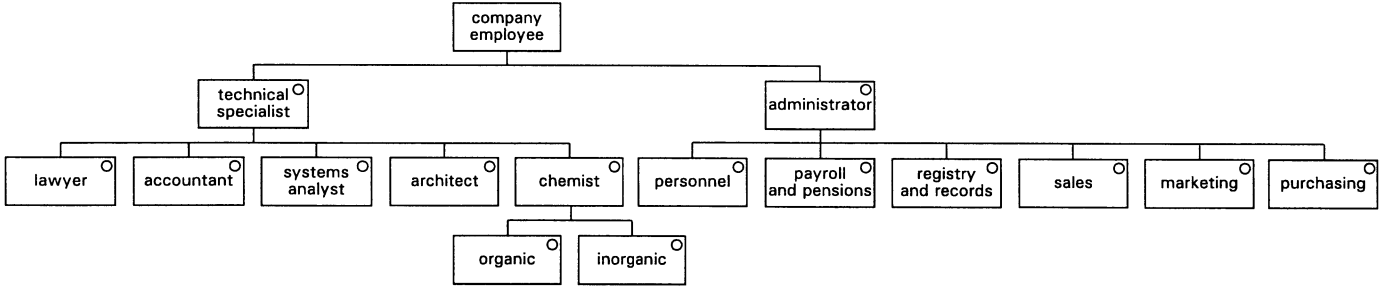

Figure A.12 Solution to Exercise 4.

5. An athlete may be tall (5 feet 10 inches or above), or below this height; each individual will be male or female. Show how this can be expressed in a JSP structure diagram in different ways.

The first possible solution (Fig. A.13) distinguishes between those who are tall and those who are not, and then differentiates between male and female in each case.

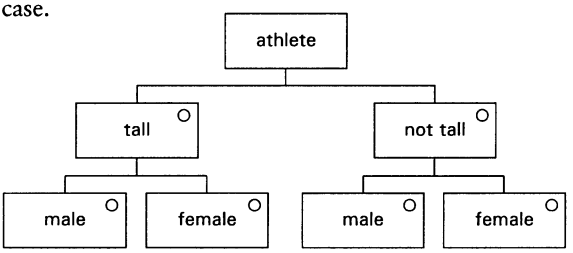

Figure A.13 First possible solution to Exercise 5.

The second possible solution (Figure A.14) differentiates between male and female athletes, and then distinguishes between those who are tall and those who are not in each case.

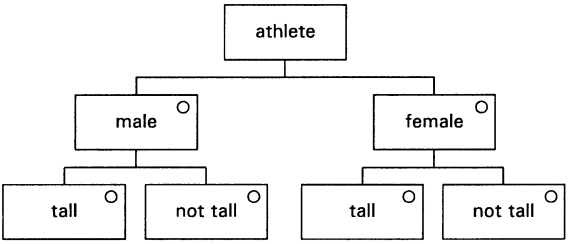

Figure A.14 Second possible solution to Exercise 5.

A third possible solution (Figure A.15) differentiates between each classification of male, female, tall and not tall at the same level.

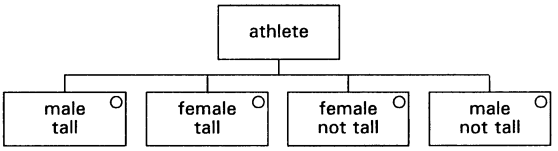

Figure A.15 Third possible solution to Exercise 5.

If we were only interested in the tall female athletes, the simple diagram in Figure A.16 would be sufficient for our purposes.

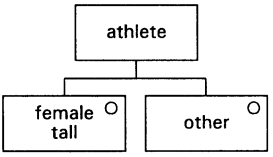

Figure A.16 Solution to Exercise 5 for tall female athletes only.

2.7 Exercises

1. A century consists of 100 years. Each year lasts 365 days (ignoring leap years for this exercise). A century could also be considered to consist of 10 decades. Show different ways in which a century can be represented using JSP structure diagrams.

   A century is an iteration of decade (10 decades to a century) (Fig. A.17).
   A century is an iteration of decade (10 decades to a century). A decade is an iteration of year (10 years to each decade) (Fig. A.18).
   A century is an iteration of year (100 years to each century) (Fig. A.19).
   A century is an iteration of day (36500 days to each century) (Fig. A.20).
   A century is an iteration of decade (10 decades to a century). A decade is an iteration of year (10 years to each decade). A year is an iteration of day (365 days to each year) (Fig. A.21).

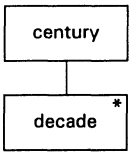

Figure A.17 Century is an iteration of decade.

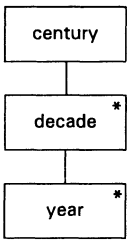

Figure A.18 Century is an iterations of decade; decade is an iteration of year.

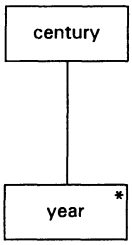

Figure A.19 Century is an iteration of year.

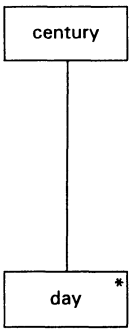

Figure A.20 Century is an iteration of day.

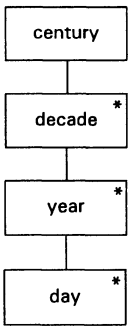

Figure A.21 Century is an iteration of decade; decade is an iteration of year; year is an iteration of day.

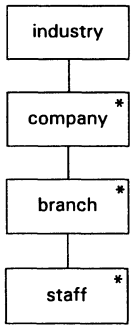

Figure A.22 Solution to Exercise 2.

2. A particular industry, e.g. travel, consists of a number of individual companies (such as travel agents). An individual company may have several branches in different locations. Each branch employs a number of staff. Illustrate the structure of the industry using a JSP diagram.

   A solution is given in Figure A.22 (above).

2.11 Exercises

1. A vending machine dispenses a bar of chocolate when the correct money is inserted (there is only one type of chocolate bar available in the machine). Use a structure diagram to illustrate the activity of inserting money and taking a bar of chocolate. *Hint: sequence* – the order of activities is important.

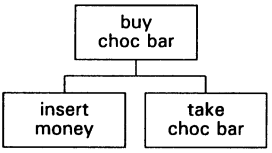

Figure A.23 Solution to Exercise 1.

2. A more sophisticated vending machine dispenses a variety of bars of chocolate: plain chocolate, milk chocolate, fruit and nut, white chocolate (all at the same price). Draw a structure diagram to illustrate the action of choosing which bar of chocolate is to be dispensed (ignoring all other activities, such as inserting money, for the present). *Hint: selection* – a bar of chocolate can only be **one** of those available, so the order is not important.

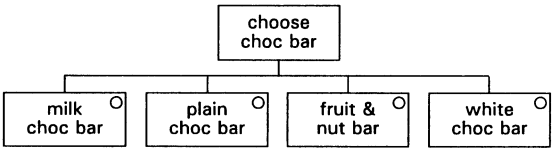

Figure A.24 Solution to Exercise 2.

3. Consider the purchase of several chocolate bars from the machine described in Exercise 2. Draw the structure diagram. *Hints: selection* – a bar of chocolate can only be **one** of those available, so the order is unimportant; *iteration* – there may be many purchases.

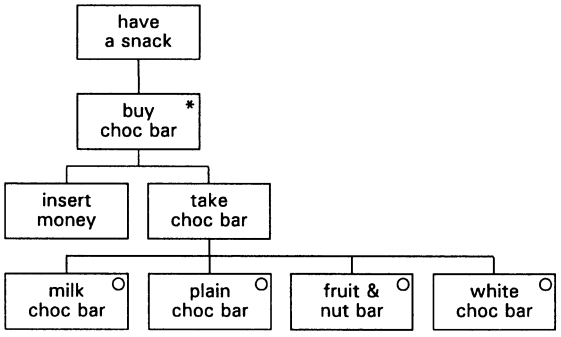

Figure A.25 Solution to Exercise 3.

4. A book consists of a number of pages. On each page there are many lines, and on each line there are many words. Draw the structure diagram. *Hint*: *iteration* – a book has many pages, etc.

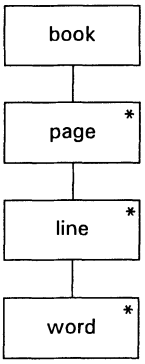

Figure A.26 Solution to Exercise 4.

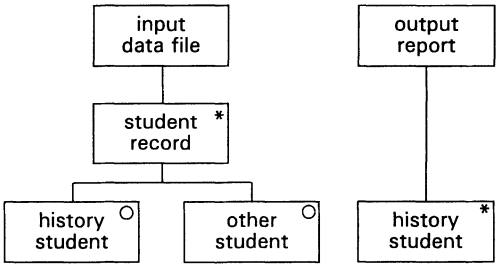

Figure A.27 Solution to Exercise 1.

3.3 Exercises

Draw the input and output data file structures for the following:

1. From a file of student records, you wish to list all students who are taking a course in history.

   See Figure A.27 (above).

2. From the same student file you wish to list all the female students who are taking a mathematics module.

   Three approaches are shown as a solution to this problem: (a) Figure A.28; (b) Figure A.29; (c) Figure A.30.

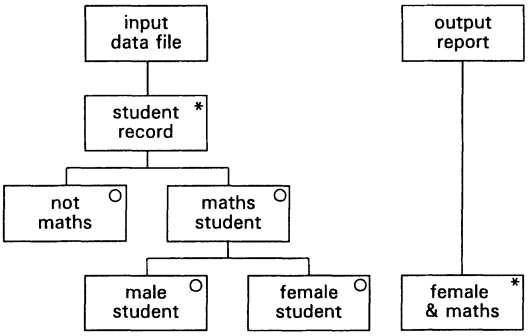

Figure A.28 One possible solution to Exercise 2.

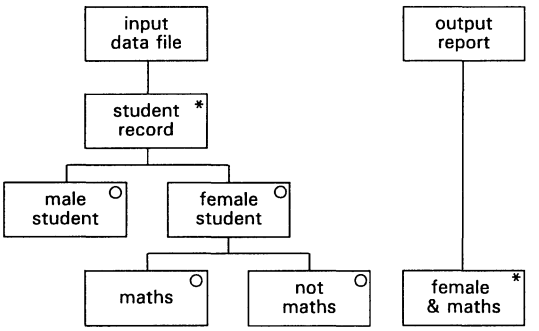

Figure A.29 A second possible solution to Exercise 2.

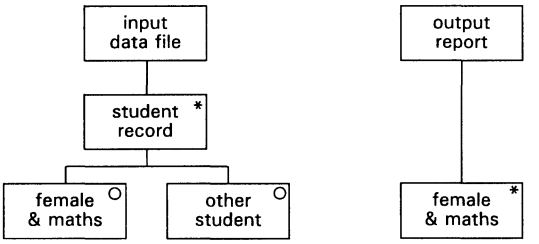

Figure A.30 A third possible solution to Exercise 2.

4. From the same student file you wish to list all the environmental science students who are female, and print the total number of these students and the grand total of all students on the file.

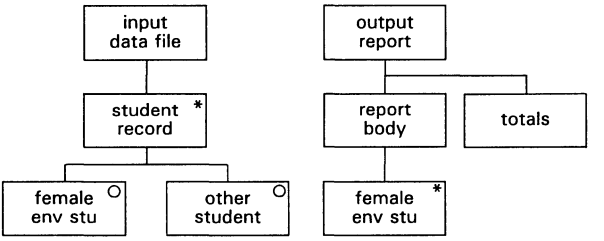

Figure A.31 Solution to Exercise 4.

5. From the same student file you wish to find the total number of all philosophy students who are over 25 years old.

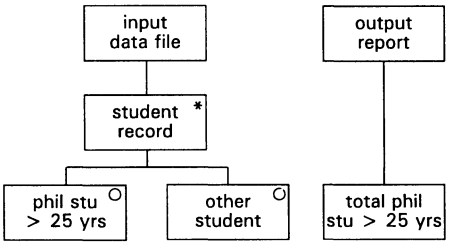

Figure A.32 Solution to Exercise 5.

6. From an employee file you wish to list all the employees earning over £30,000 who speak Italian.

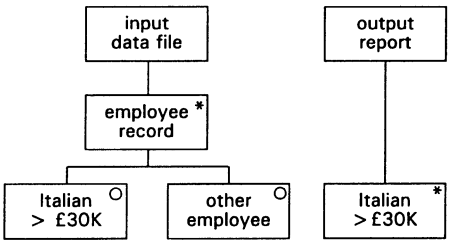

Figure A.33 Solution to Exercise 6.

7. From the same employee file you wish to print out details of all employees who are not qualified in accountancy.

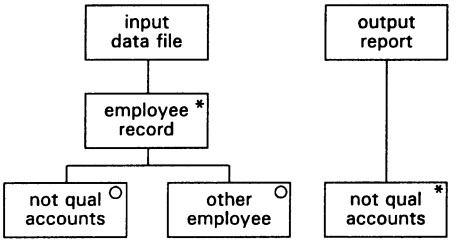

Figure A.34 Solution to Exercise 7.

12. The data structure shown in Figure 3.5 (repeated as Figure A.35) represents a file that may contain records A, B, C, D and E. These are the leaves of the tree. Is the file structure compatible with each of the following record sequences:

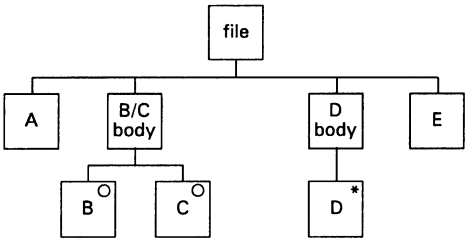

Figure A.35 Data structure for Exercise 12 (repeat of Fig. 3.5).

(a) A B C D E	No, cannot have both B and C
(b) A C E	Yes
(c) A D E	No, must have B or C (but not both)
(d) A B C E	No, cannot have both B and C
(e) A B E E	Yes
(f) A B E	Yes

(g) B D E	No, A is absent
(h) A B C	No, must have E present
(i) A B D D D E	Yes
(j) A C D D E	Yes
(k) A C B E	No, cannot have both B and C
(l) A C C D E	Yes

4.7 Exercises

Consider the following problems, draw the input and output data file structures, identify the points of correspondence and merge to form the basic process structure in each case.

1. From a file of student records, you wish to list all students who are taking a course in history.

   The input and ouput structures, together with their points of correspondence are shown in Figure A.36. Input and output data structures merged on the points of correspondence to form basic process structure are shown in Figure A.37.

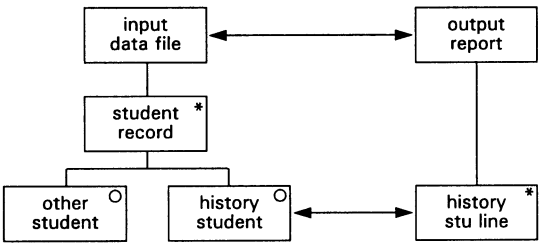

Figure A.36 Points of correspondence for Exercise 1.

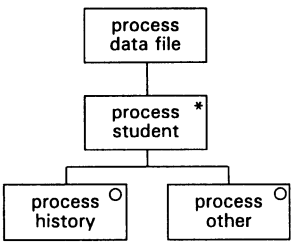

Figure A.37 The basic process structure for Exercise 1.

2. From the same student file you wish to list all the female students who are taking a mathematics module.

   Taking the three approaches shown earlier (Exercises 3.3) as a solution to this problem: (a) Figure A.38; (b) Figure A.39; (c) Figure A.40.

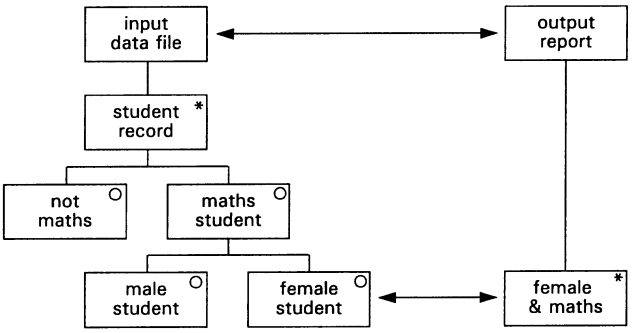

Figure A.38 Points of correspondences for first possible solution to Exercise 2.

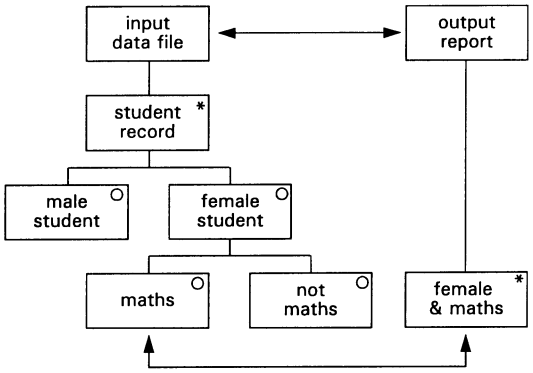

Figure A.39 Points of correspondence for second possible solution to Exercise 2.

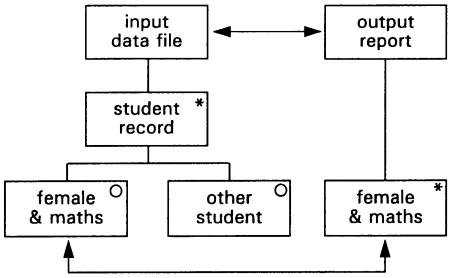

Figure A.40 Points of correspondence for third possible solution to Exercise 2.

5.4 Exercises

From the input and output file structures for the following exercises, identify the "housekeeping" activities that need to be performed and add process boxes to the process structure diagram to allow these actions to be made.

1. From a file of student records you wish to list all students who are taking a course in history.

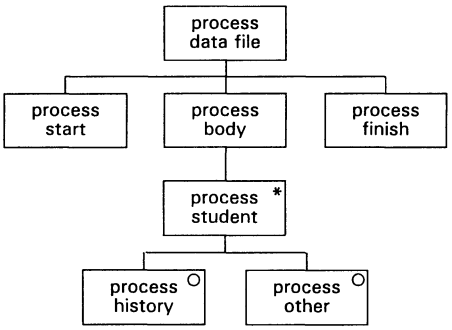

Figure A.41 Solution to Exercise 1.

6.6 Exercises

From the input and output file structures for the following exercises, introduced in the previous exercises, establish the points of correspondence, merge the data structures to form a process structure, specify the conditions and operations and allocate these to the process structure.

1. From a file of student records you wish to list all students who are taking a course in history.

The conditions are:

```
c1  not end of file
c2  if history student
c3  else
```

The operations list is:

1. `Read student record from input file`
2. `Open input file for input`
3. `Open report file for output`
4. `Write student record to output report file`
5. `Stop processing`
6. `Close input file`
7. `Close output file`

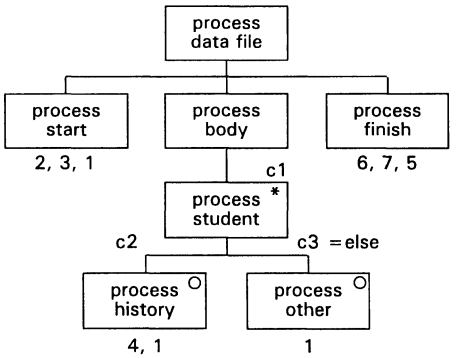

Figure A.42 The process structure for Exercise 1.

7.5 Exercises

1. Write the schematic logic for the structures shown in (a) Figure 7.13 (repeated as Figure A.43), (b) Figure 7.14 (repeated as Figure A.44), (c) Figure 7.15 (repeated as Figure A.45) and (d) Figure 7.16 (repeated as Figure A.46).

The logic for (a) is:

```
A seq
    B seq
        B operations
    B end
    C itr while c1
        D seq
            D operations
        D end
    C end
A end
```

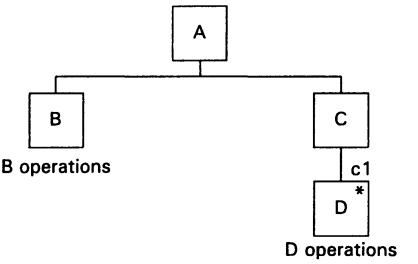

Figure A.43 The process structure for Exercise 1(a).

The logic for (b) is:

```
A sel if c1
    B seq
        B operations
    B end
A or if c2
    C seq
        E seq
            E operations
        E end
        F seq
            F operations
        F end
    C end
A or if c3 (else)
    D seq
        D operations
    D end
A end
```

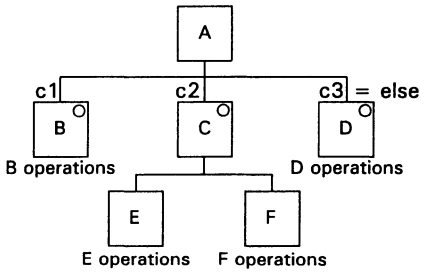

Figure A.44 The process structure for Exercise 1(b).

The logic for (c) is:

```
A sel if c1
    B seq
        B operations
    B end
A or c2 (else)
    C itr while c3
        D seq
            D operations
        D end
    C end
A end
```

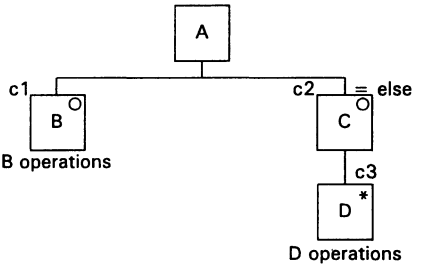

Figure A.45 The process structure for Exercise 1(c).

The logic for (d) is:

```
A seq
    B seq
        B operations
    B end
    C sel if c1
        E seq
            E operations
        E end
    C or if c2 (else)
        F seq
            F operations
        F end
    C end
    D seq
        D operations
    D end
A end
```

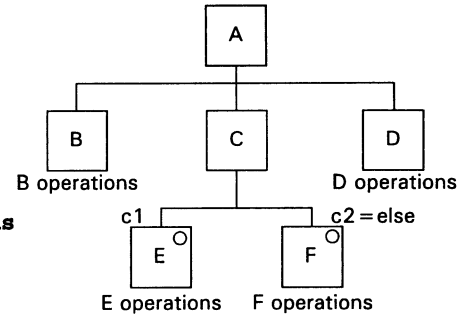

Figure A.46 The process structure for Exercise 1(d).

2. Reconstruct the process structures from the schematic logic:

(a)

```
A seq
    B seq
        C seq
            do C operations
        C end
        D seq
            do D operations
        D end
    B end
    E sel if c1
        F seq
            do F operations
        F end
    E or c2 (else)
        G seq
            do G operations
        G end
    E end
    H itr while c3
        J seq
            do J operations
        J end
    H end
A end
```

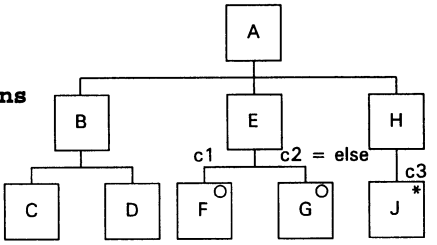

Figure A.47 The process structure for Exercise 2(a).

(b)
```
A itr while c1
    B sel if c2
        C itr while c5
            D seq
                do D operations
            D end
        C end
    B or if c3
        L seq
            E seq
                do E operations
            E end
            F sel c6
                P seq
                    G seq
                        do G operations
                    G end
                    H seq
                        do H operations
                    H end
                P end
            F or c7 (else)
                J seq
                    do J operations
                J end
            F end
        L end
    B or c4 (else)
        K seq
            do K operations
        K end
    B end
A end
```

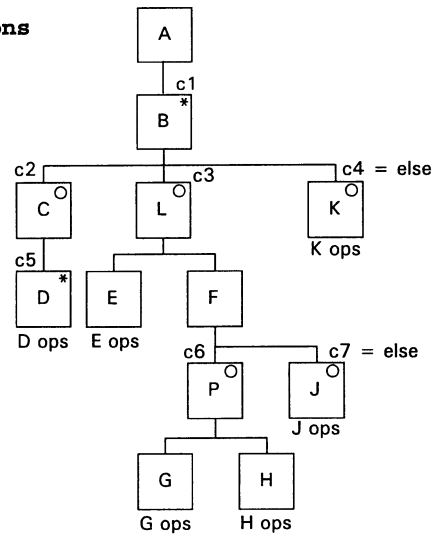

Figure A.48 The process structure for Exercise 2(b).

Consider the following problems. Generate the schematic logic for each process structure.

i. From a file of student records, you wish to list all students who are taking a course in history.

The schematic logic is as follows:

```
process data file seq
  process start seq
    2.open input file for input
    3.open report file for output
    1.read student record from input file
  process start end
  process body itr while not end of file
    process student sel if history student
      4.write student record to output report file
      1.read student record from input file
    process student or (else)
      1.read student record from input file
    process student end
  process body end
  process finish seq
    6.close input file
    7.close output file
    5.stop processing
  process finish end
process data file end
```

8.5 Exercises

1. Convert the following schematic logic into the programming language of your choice.

(a) In the following translation of the schematic logic into Pascal the schematic logic is given as comments, e.g. `(* process total staff seq *)`:

```
begin
(* process total staff seq *)
  (* process start seq *)
    assign(staffile, 'staffile');
    reset(staffile);  (* open staff file for input *)
    assign(report, 'report');
    rewrite(report);  (* open report file for output *)
    staff_tot := 0;   (* set staff total to zero *)
    readln(staffile, name, job_title);
             (* read staff record from
               input staff file *)
  (* process start end *)
```

```
(* process staff body itr while not end of staff file *)
while not eof(staffile) do
begin
   (* process record seq *)
     staff_tot := staff_tot + 1;
            (* add 1 to staff total *)
     readln(staffile, name, job_title);
            (* read staff record from
              input staff file *)
   (* process record end *)
end;
(* process staff body end *)
(* process total seq *)
  writeln(report, 'Starlight Staff');
     (* write company title to output report file *)
  writeln(report, 'staff total: ', staff_tot:8)
     (* write staff total to output report file *)
  writeln(report, 'October 1996');
     (* write date to output report file *)
process total end
(* process finish seq *)
  close(staffile);          (* close staff file *)
  close(report);            (* close report file *)
  end.                      (* stop processing *)
(* process finish end *)
(*process total staff end*)
```

Consider the following problems. Produce the code for each process structure, using the programming language of your choice.

3. From a file of student records, you wish to list all students who are taking a course in history.
   In the following translation of the schematic logic into Pascal the schematic logic is given as comments, e.g. `(* process data file seq *)`:

```
begin
(* process data file seq *)
  (* process start seq *)
    assign(stufile, 'stufile');
    reset(stufile);
    assign(report, 'report');
    rewrite(report);
    readln(stufile, name, course);
  (* process start end *)
  (* process body itr while not end of file *)
  while not eof(stufile) do
    (* process student sel if history student *)
    if course = 'history' then
```

```
      begin
        (writeln(report, name:20, course:10);
        readln(stufile, name, course);
      (* process student or (else) *)
      end
      else
        readln(stufile, name, course);
      (* process student end *)
    (* process body end *)
    (* process finish seq *)
      close(stufile);
      close(report);
      end.
    (* process finish end *)
  (* process data file end *)
```

Note that it will be necessary to add the appropriate data declarations, etc., in order to achieve a program that runs.

9.5 Exercises

The following structures represent files; valid record types are A, B, C, D and E. Records in error are of type **X**, and may be inserted at any point and in any number. The original records retain their order. Elaborate the structures below for errors of

(a) insertion
(b) substitution
(c) omission

and then elaborate for further errors in each case.

1. A simple input file, contains only two records, A followed by B (Fig. 9.13; repeated as Fig. A.49).

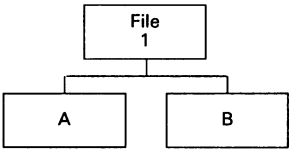

Figure A.49 The input file for Exercise 1 (repeat of Fig. 9.13).

Figures A.50, A.51 and A.52 show the structure elaborated for errors of (a) insertion; (b) substitution and (c) omission respectively.

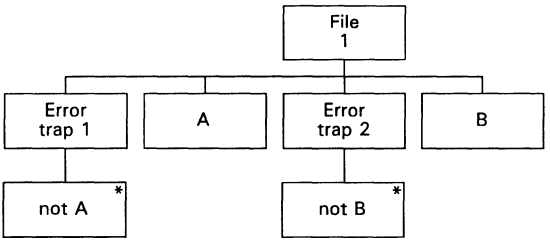

Figure A.50 Elaboration for errors of insertion: exercise 1.

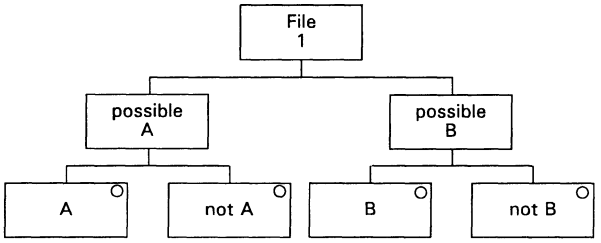

Figure A.51 Elaboration for errors of substitution: exercise 1.

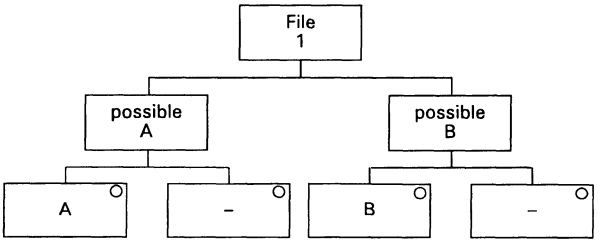

Figure A.52 Elaboration for errors of omission: exercise 1.

2. An input file contains an initial record A, followed by several B records (Fig. 9.14, repeated as Fig. A.53).

   Figures A.54, A.55 and A.56 show the structure elaborated for errors of (a) insertion; (b) substitution and (c) omission respectively.

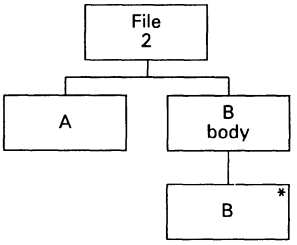

Figure A.53 The input file for Exercise 2 (repeat of Fig. 9.14).

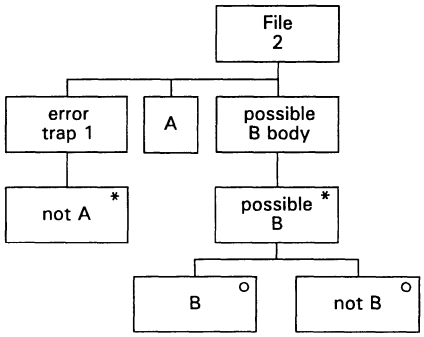

Figure A.54 Elaboration for errors of insertion: exercise 2.

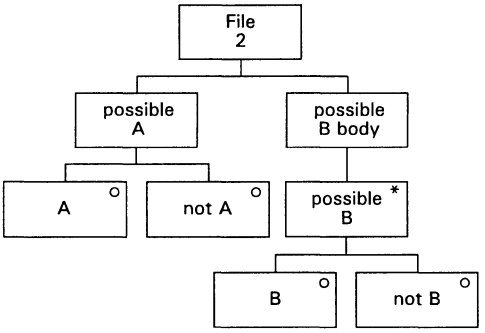

Figure A.55 Elaboration for errors of susbstitution: exercise 2.

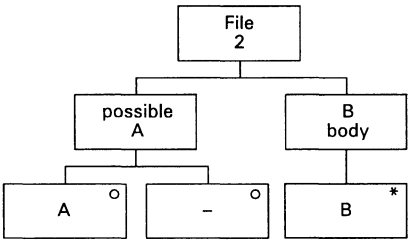

Figure A.56 Elaboration for errors of omission: exercise 2.

3. An input file contains an initial A record, followed by several B records, and then a single terminating record that may be of type C, D or E (Fig. 9.15, repeated as Fig. A.57).

   Figures A.58, A.59 and A.60 show the structure elaborated for errors of (a) insertion; (b) substitution and (c) omission respectively.

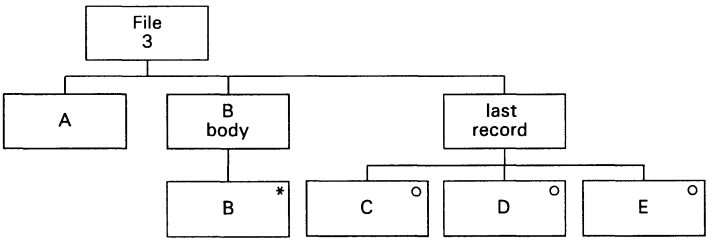

Figure A.57 The input file for Exercise 3 (repeat of Fig. 9.15).

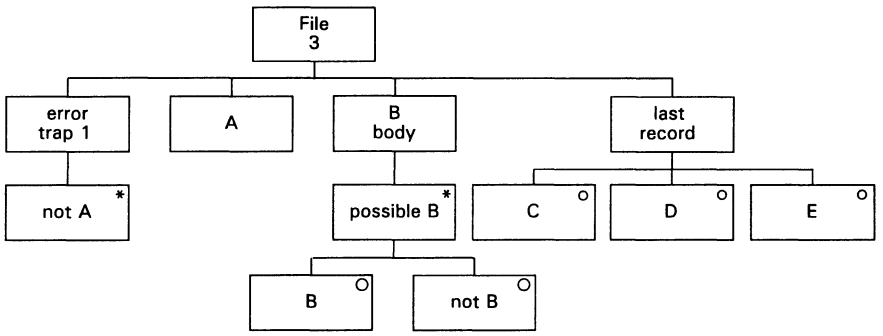

Figure A.58 Elaboration for errors of insertion: exercise 3.

10.7 Exercises

1. How could the process structure diagram for the case study be developed to allow for additional functions? Such functions could be, for example, to inspect a customer record without changing it, or to print a customer record.

   The process structure diagram is designed in order to allow such flexibility of approach. It would be possible to add another function as another selection part within the function iterated component. It would also be possible to remove a function, for example, to prevent records from being deleted from the customer database. The process structure diagram would have the appearance of Figure A.61 if inspect and print options were to be added. (The cancel option has been omitted for clarity).

   Note that the new functions inspect and print have been added including the ability to deal with errors of insertion.

2. Do you consider that elaborating for errors of insertion would be the best choice if the customer database system were to run in batch rather than interactive mode? Explain your reasoning.

   Elaborating for errors of insertion is a suitable choice for dealing with data entered interactively by a user; if the user makes an error, it is likely that further attempt(s) will be made until a correct entry is made (or the user terminates the session).

   In the context of a batch system, it will not be possible for the user to respond to errors in the same way as with an interactive system; it is therefore necessary to consider which type of error is most likely to be made if the updating of the customer database is performed by reference to an input file. If errors arise in the records stored for processing against the main database, this could be regarded as an error of substitution because each erroneous record can be regarded as occurring in substitution for a correct record. It would be difficult to determine if errors of omission had occurred, as, if there is no update record for a particular record on the database, this may simply mean that the record concerned does not need to be updated on this occasion.

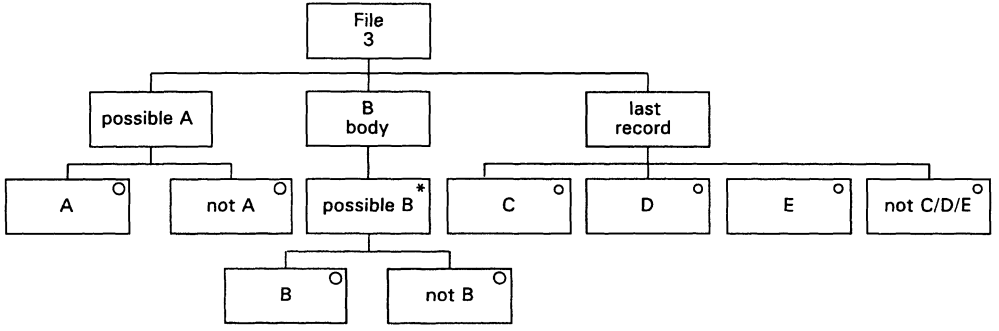

Figure A.59 Elaboration for errors of substitution: exercise 3.

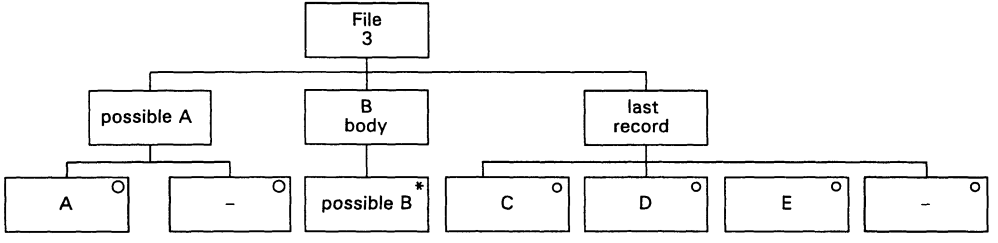

Figure A.60 Elaboration for errors of omission: exercise 3.

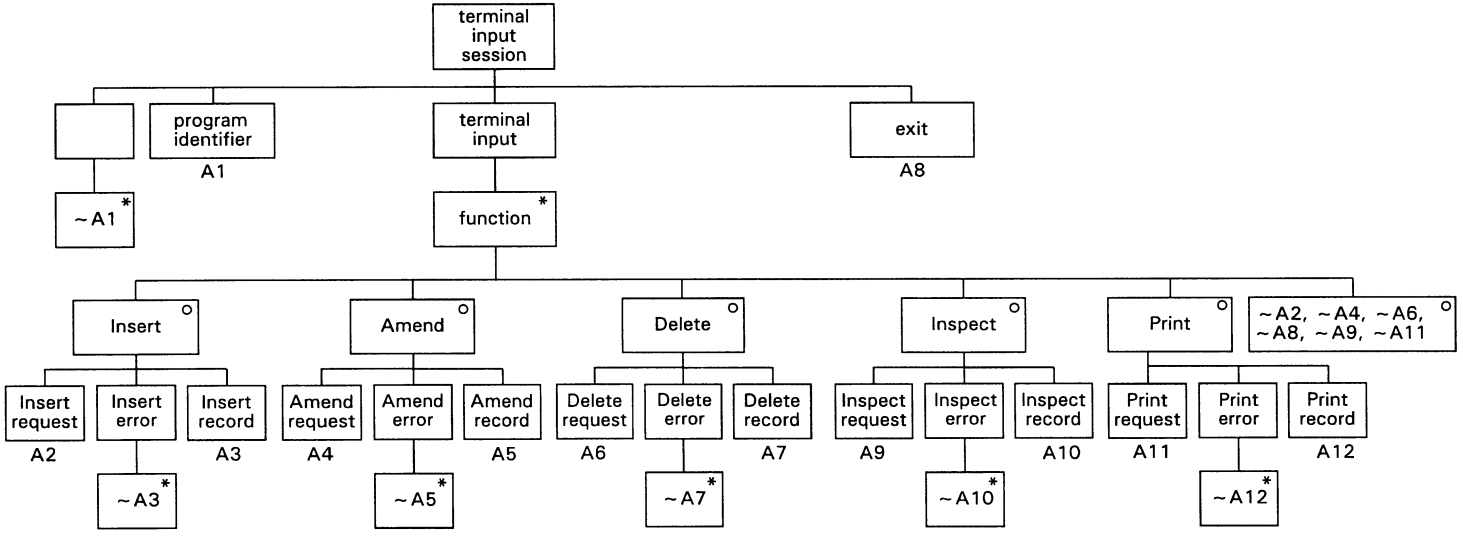

Figure A.61 Customer database system with inspect and print functions added.

11.5 Exercises

1. Describe the role of **posit**, **admit** and **quit** in backtracking.

   Posit and **admit** are used to replace **if . . . then . . . else . . .** when the information needed to test the condition on a selection is not available, and therefore it is not possible to decide with absolute certainty which selection branch should be taken. **Posit** replaces the **if . . . then . . .** part of the selection; the result of this is that one branch of the selection is entered without first testing the condition. It is possible, using this approach, that the wrong branch is taken. When sufficient information is available to establish whether or not the right path has been chosen, a test can be carried out; if the chosen path is the correct one, processing can continue as if the test had been performed at the usual point at the start of the selection. However, if it is found that the wrong branch has been selected, control must be transferred to the other branch, labelled by **admit** (replacing the **else** branch). The transfer of control to the **admit** part is executed by means of a **quit**, which is normally implemented as a **go to** statement. It is important that any intolerable side effects are avoided or reversed if control needs to be transferred from the **posit** branch to the **admit** branch.

12.4 Exercises

1. Explain the meaning of the following terms, and how they are related
   correspondence
   lack of correspondence
   structure clash.

   A *correspondence* is said to occur between an input and an output data structure, where input is processed to generate output, if the elements in each are related by the following rules:
   - the elements are found in the same order
   - the elements appear the same number of times
   - the elements occur in the same context

   If these rules are satisfied at all appropriate points in the two data structures, the structures can be merged to form the basic process structure. If there is a point in the data structures where the rules are broken, e.g. because the elements do not occur in the same order, there is a structure clash; this will prevent the data structures from being merged to form a process structure. If there are boxes in one data structure that do not occur in the other, e.g. a title on an output report, this is a lack of correspondence which does not prevent the structures from being merged. All boxes in either data structure diagram must be represented by a box in the process structure diagram.

   A *lack of correspondence* means that an element on either the input file or the output file does not occur in the other file. Examples include: a total will be calculated during processing, a heading will be at the top of a page, and a page number will be at the bottom of a report page; these will therefore appear on the output file but not on the input file.

A *structure clash* indicates that there is conflict between some part of the input data structure and the output data structure because one or more of the rules for establishing a correspondence has been broken. This means that the input and output data structures cannot be merged and another solution method must be adopted. It may be possible to redesign the data structures in order to avoid the structure clash, or it may be necessary to redesign as if using an intermediate data file by creating two process structures; this latter approach can be converted to program inversion (the intermediate file is not physically created; the two programs communicate record-by-record).

2. What are the three types of structure clash? Give an example of each.

An *ordering clash* occurs when the records do not appear in the same order in the input file and the output file; sorting the input file into the required order will resolve the clash.

A *boundary clash* occurs when the boundaries (e.g. departments) on the input file do not match up with the boundaries (e.g. pages) on the output file. In this case, two programs need to be designed; an intermediate file is used as the output file from the first process and as the input file to the second process. Implementing program inversion does away with the need to create the intermediate file – the two programs communicate by passing each record between the main program and the subprogram as and when required.

An *interleaving clash* can be regarded as a combination of a boundary clash and an ordering clash; the records in the input file relating to a specific entity are interspersed with other records, i.e. they are interleaved. One solution would be to sort the input file (as with an ordering clash). Another option would be to design several different programs (one for each distinct entity) and implement these using multiple program inversion.

3. What is the difference between using an intermediate file and implementing program inversion when dealing with the problem of a structure clash?

There is no difference in the *structure* of the two programs if an intermediate file is created or if program inversion is implemented. The difference that does occur is that the intermediate data file is not physically created if program inversion is implemented, as each record that would have been written to the intermediate data file as output is instead passed to the other program for processing; this removes the need for the second program to read the intermediate data file explicitly because the records are passed across as and when they require processing.

4. Why is it necessary to delete references to an intermediate file in the subprogram when program inversion is being used?

References to the intermediate file are no longer required as the main program will call the subprogram passing the record that would have been read from the intermediate file. The records are passed between the main program and the subprogram in the same order as they would have been written to the intermediate file (and read from the intermediate file). Therefore there is no need for the intermediate file to be created in a physical sense, it is only used as an aid to the design of the two communicating programs.

13.6 Exercises

1. Show how you would dismember the process structure diagram shown in Figure 13.9 (repeated as Fig. A.62) (a) graphically; and (b) in tabular form.

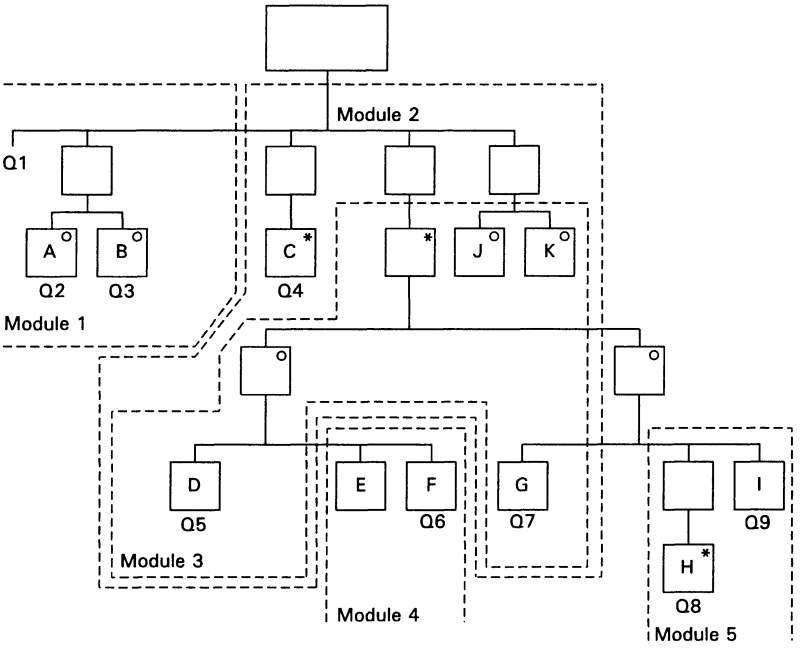

Figure A.62 The process structure diagram for Exercise 13.6.1 (repeat of Fig. 13.9).

Note that module 3 is completely contained within module 2; it would be possible to merge these into a single module if required.

The resume execution table is shown as Table A.1, the module link table as Table A.2 and the module link table produced by merging modules 2 and 3 as Table A.3.

Table A.1 Resume execution table: exercise 1.

Resume points	Elementary components
Q1	A, B
Q2	C, D, G, J, K
Q3	C, D, G, J, K
Q4	C, D, G, J, K
Q5	E, F
Q6	D, G, J, K
Q7	H, I
Q8	H, I
Q9	D, G, J, K

Table A.2 Module link table: exercise 1.

Module name	Elementary components	Suspend points	Module link
Mod 1	A, B	Q2	Mod 2
		Q3	Mod 2
Mod 2	C, D, G, J, K	Q4	Mod 2
		Q5	Mod 4
		Q7	Mod 5
		END	NIL
Mod 3	D, G, J, K	Q5	Mod 4
		Q7	Mod 5
		END	NIL
Mod 4	E, F	Q6	Mod 3
Mod 5	H, I	Q8	Mod 5
		Q9	Mod 3

Table A.3 Module link table: exercise 1, merged modules.

Module name	Elementary components	Suspend points	Module link
Mod 1	A, B	Q2	Mod 2
		Q3	Mod 2
Mod 2	C, D, G, F, J, K	Q4	Mod 2
		Q5	Mod 3
		Q7	Mod 4
		END	NIL
Mod 3	E, F	Q6	Mod 2
Mod 4	H, I	Q8	Mod 4
		Q9	Mod 2

Table A.3 shows that the two modules, where module 3 was contained within module 2, can be merged into a single module. The overhead for doing this is that there will be some cases where the elementary component **C** will be loaded when it is not required, i.e. module 2 will be loaded where the "old" module 3 would have been loaded.

If a large amount of code is involved with elementary component **C** it might be considered wasteful to load the entire module if it is known that **C** will not be required; in these circumstances, the previous dismemberment would be employed.

2. Show how you would dismember the process structure diagram shown in Figure 13.10 (repeated as Fig. A.63) (a) graphically; and (b) in tabular form.

The resume execution table is shown as Table A.4 and the module link table as Table A.5.

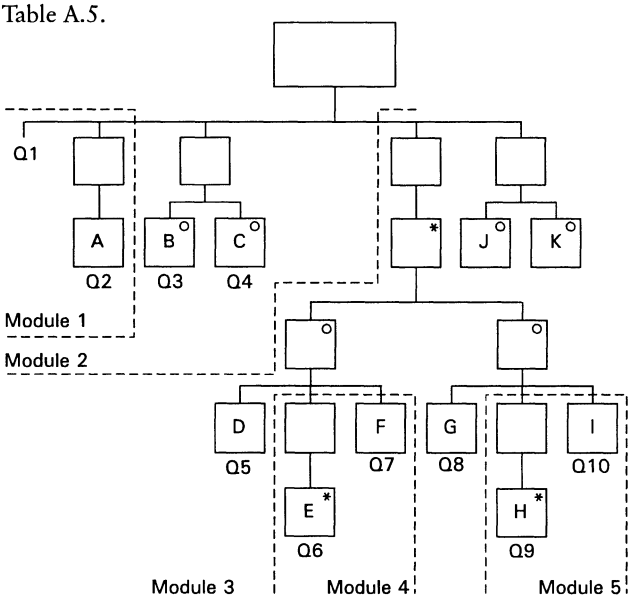

Figure A.63 The process structure diagram for Exercise 2 (repeat of Fig. 13.10) with graphic dismemberment.

Table A.4 Resume execution table: exercise 2.

Resume points	Elementary components
Q1	A
Q2	B, C
Q3	D, G, J, K
Q4	D, G, J, K
Q5	E, F
Q6	E, F
Q7	D, G, J, K
Q8	H, I
Q9	H, I
Q10	D, G, J, K

Table A.5 Module link table: exercise 2.

Module name	Elementary components	Suspend points	Module link
Mod 1	A	Q2	Mod 2
Mod 2	B, C	Q3	Mod 3
		Q4	Mod 3
Mod 3	D, G, J, K	Q5	Mod 4
		Q8	Mod 5
		END	NIL
Mod 4	E, F	Q6	Mod 4
		Q7	Mod 3
Mod 5	H, I	Q9	Mod 5
		Q10	Mod 3

3. Show how you would dismember the process structure diagram shown in Figure 13.11 (repeated as Fig. A.64) (a) graphically; and (b) in tabular form.

The resume execution table is shown as Table A.6 and the module link table as Table A.7.

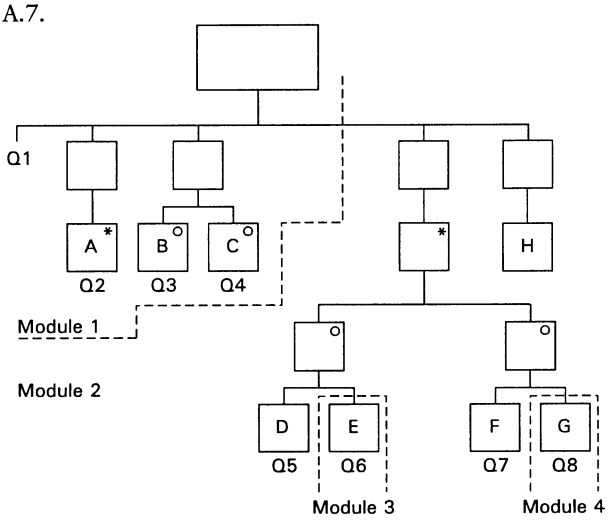

Figure A.64 The process structure diagram for Exercise 3 (repeat of Fig. 13.11) with graphic dismemberment.

Table A.6 Resume execution table: exercise 3.

Resume points	Elementary components
Q1	A, B, C
Q2	A, B, C
Q3	D, F, H
Q4	D, F, H
Q5	E
Q6	D, F, H
Q7	G
Q8	D, F, H

Table A.7 Module linke table: exercise 3

Module name	Elementary components	Suspend points	Module link
Mod 1	A, B, C	Q2	Mod 1
		Q3	Mod 2
		Q4	Mod 2
Mod 2	D, F, H	Q5	Mod 3
		Q7	Mod 4
		END	NIL
Mod 3	E	Q6	Mod 2
Mod 4	G	Q8	Mod 2

4. Show how you would dismember the process structure diagram shown in Figure 13.12 (repeated as Fig. A.65) (a) graphically; and (b) in tabular form.

   The resume execution table is shown as Table A.8 and the module link table as Table A.9.

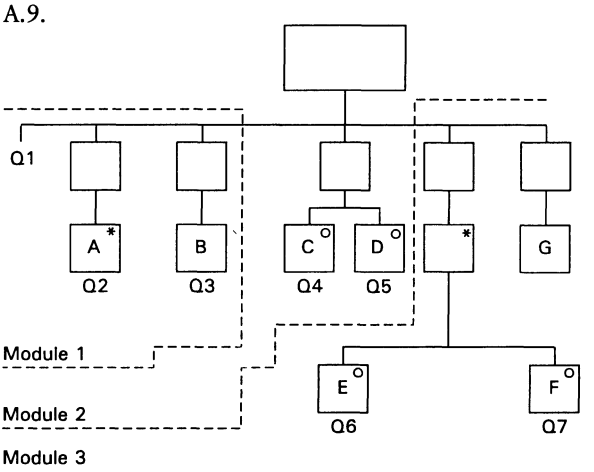

Figure A.65 The process structure diagram for Exercise 4 (repeat of Fig. 13.12) with graphic dismemberment.

Table A.8 Resume execution table: exercise 4.

Resume points	Elementary components
Q1	A, B
Q2	A, B
Q3	C, D
Q4	E, F, G
Q5	E, F, G
Q6	E, F, G
Q7	E, F, G

Table A.9 Module link table: exercise 4.

Module name	Elementary components	Suspend points	Module link
Mod 1	A, B	Q2	Mod 1
		Q3	Mod 2
Mod 2	C, D	Q4	Mod 3
		Q5	Mod 3
Mod 3	E, F, G	Q6	Mod 3
		Q7	Mod 3
		END	NIL

References

Delannoy, C. 1989. *Turbo Pascal programming.* Basingstoke: Macmillan.

Farmer, M. 1989. *The intensive Pascal course.* 2nd edn. Kent: Chartwell-Bratt.

Findlay, W. & D. A. Watt 1990. *Pascal: an introduction to methodical programming.* 3rd edn. London: Pitman.

Gottfried, B. S. 1990. *Theory and problems of programming in C.* Schaum's Outline Series in Computers. New York: McGraw-Hill.

Haiduk, H. P. 1990. *Object oriented Turbo Pascal. Problem solving and programming.* New York: McGraw-Hill.

Holmes, B. J. 1991. *Structured programming in COBOL,* 2nd edn. London: DP Publications.

Jackson, M. A. 1975. *Principles of program design.* New York: Academic Press

King, M. J. & J. P. Pardoe 1992, *Program design using JSP: a practical introduction.* 2nd edn. Basingstoke: Macmillan.

Konvalina, J. & S. Wileman 1987. *Programming with Pascal.* Singapore: McGraw-Hill.

Perry, P. J. 1993. *Crash course in C.* Indianapolis, IN: Que Corp.

Robson, D. J. 1990. *Programming for change with Pascal.* Kent: Chartwell-Bratt.

Storer, R. 1991. *Practical program development using JSP.* Oxford: Blackwell Scientific Publications.

Thompson, J. B. 1989. *Structured programming with COBOL and JSP.* Kent: Chartwell-Bratt.

Index