CHAPTER 13

File Structures

(Solutions to Practice Set)

Review Questions

1. The two access methods are sequential and random.
2. The old master file is the file that should be updated; the new master file contains the current data (the data from the old master file including any changes that were made during the update process).
3. The transaction file contains changes that should be made to the old master file.
4. To access a record in a file randomly, we need to know the address of the record.
5. The index is a table that relates the keys of the data items to the addresses in the file where the data are stored.
6. In direct hashing, the key is the address of the record in the file.
7. In modulo division hashing, the key is divided by the file size. The remainder plus 1 is used as the address of the record in the file.
8. In digit extraction hashing, certain digits are removed from the key and used as the address of the record.
9. A collision occurs when two hashed record have the same address. The three collision methods are open addressing, linked list resolution, and bucket hashing. In open addressing, the prime area is searched for an unoccupied address. In linked list resolution, the first record is stored in the home address, but it contains a pointer to the second record. In bucket hashing, a group of records are stored in a buckets which are locations that can accommodate more than one record.
10. A text file is a file of characters while a binary file is a collection of data stored in the internal format of the computer.

Multiple-Choice Questions

11. d	12. a	13. a	14. d	15. c	16. a
17. a	18. c	19. d	20. a	21. d	22. b
23. a	24. c	25. a	26. d	27. d	28. a

Exercises

29. The files are shown in Figure S13.29.

Figure S13.29 *Exercise 29*

New Master File

Key	Name	Pay Rate
14	John Wu	17.00
16	George Brown	18.00
17	Duc Lee	11.00
26	Ted White	23.00
31	Joanne King	28.00
89	Mark Black	19.00
90	Orva Gilbert	20.00
92	Betsy Yellow	14.00

Error File

Action	Key	Name	Pay Rate
A	17	Martha Kent	17.00

30. Both data file and index file are shown in the following tables.

Index	
Key	*Address*
077654	004
093245	003
114237	001
123453	000
156734	002
256743	005
423458	006

Data file			
Address	*Key*	*Name*	*Dept.*
000	123453	John Adam	CIS
001	114237	Ted White	MTH
002	156734	Jimmy Lions	ENG
003	093245	Sophie Grands	BUS
004	077654	Eve Primary	CIS
005	256743	Eva Lindens	ENG
006	423458	Bob Bauer	ECO

31.

a. (14232 **mod** 41) + 1 = 5 + 1 = 6
b. (12560 **mod** 41) + 1 = 14 + 1 = 15
c. (13450 **mod** 41) + 1 = 2 + 1 = 3
d. (15341 **mod** 41) + 1 = 7 + 1 = 8

32.

a. $142^2 = 20164 \rightarrow 16$

b. $125^2 = 15625 \rightarrow 62$

c. $134^2 = 17956 \rightarrow 95$

d. $153^2 = 23408 \rightarrow 40$

33.

a. 14 + 22 = 36

b. 12 + 57 = 69

c. 13 + 49 = 62

d. 15 + 32 = 47

34.

a. 41 + 22 + 43 = 106

b. 21 + 57 + 11 = 89

c. 31 + 49 + 91 = 171

d. 51 + 32 + 31 = 114

35.

a. (10278 **mod** 411) + 1 = 3 + 1 = 4

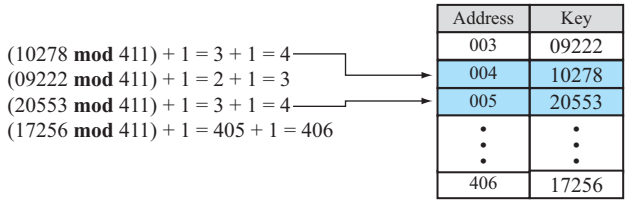
b. (08222 **mod** 411) + 1 = 2 + 1 = 3

c. (20553 **mod** 411) + 1 = 3 + 1 = 4 (collision) → 5

d. (17256 **mod** 411) + 1 = 405 + 1 = 406

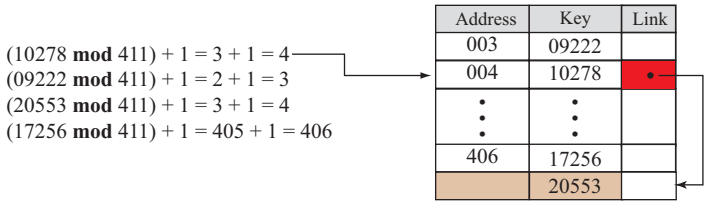
The result of open addressing resolution is shown in Figure S13.35.

Figure S13.35 *Exercise 35*



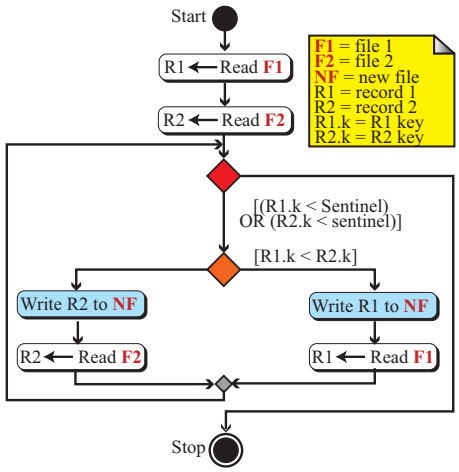
36. The result of linked list resolution is shown in Figure S13.36.

Figure S13.36 *Exercise 36*



37. The UML is shown in Figure S13.37. Note that the routine is works correctly even if one or both input files are empty (having only one dummy record).

Figure S13.37 *Exercise 37*



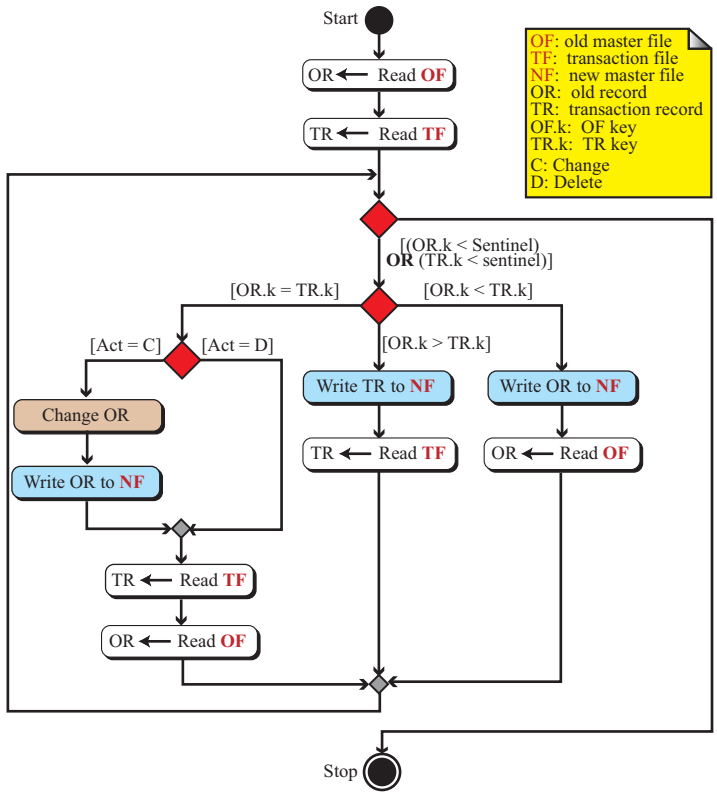
38. Algorithm S13.38 shows the routine in pseudocode.

Algorithm S13.38 *Exercise 38*

```
Algorithm: MergeFile (F1, F2, Sentinel)
Purpose: It merges two sequential files
Pre: Given F1 and F2 and the Sentinel
Post:
Return: A new File (NF)
{
        R1 ← Read F1
        R2 ← Read F2
        while ((R1.k < Sentinel) OR (R2.k < Sentinel))
        {
                if (R1.k < R2.k)
                {
                        Write R1 to NF
                        R1 ← Read F1
                }
                else
                {
                        Write R2 to NF
                        R2 ← Read F2
                }
        }
        return NF
}
```

39. The UML is shown in Figure S13.41.

Figure S13.39 *Exercise 39*



For simplicity, we assume that there is no discrepancy between OF and TF, which means that there is no need to create an error file. The routine follows the logic in the solution to Exercise 37. The loop will terminate when both files have reached their dummy records. The process, however, needs three decision branches. If the there is no transaction for a record (OR.k < TR.k), the record in the old master file is copied to the new transaction file. If there is a new record in the transaction file to be inserted into the new master file (OR.k > TR.k), then the routine simply writes that record. If (OR.k = TR.k), it means that a record needs to be either partially changed or totally deleted. In the case of deletion, no action is needed. In the case of change, the record in the old master file is updated and written to the new master file. Note that the routine works correctly if the transaction file is empty (having only one dummy record). This may happen, if at the end of the updating period, there is no changes to the old master file. The old master file is simply copied to the new master file. The routine also works correctly if the old master file is empty (creation of a new file).

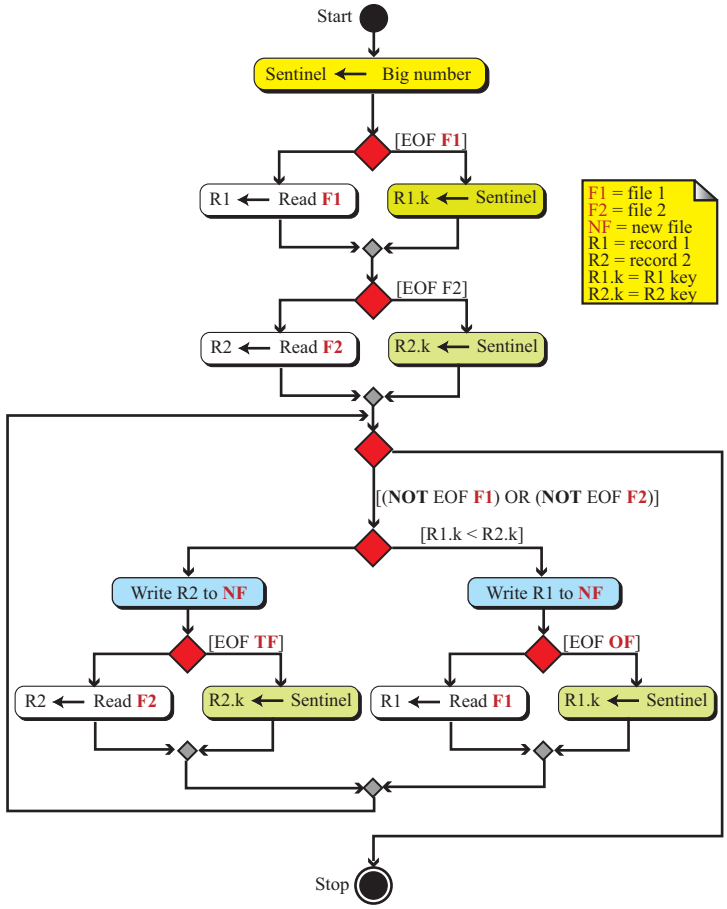
40. Algorithm S13.40 shows the routine in pseudocode. It exactly follows the UML in Exercise 39. We have used an empty block when a record is to be deleted to show all cases (matching the pseudocode with the UML in Exercise 39). It is not normally used in practice.

Algorithm S13.40 *Exercise 40*

```
Algorithm: UpdateFile (OF, TR, Sentinel)
Purpose: It updates a sequential files
Pre: Given old master file (OF) and transaction file (TF) and the Sentinel
Post:
Return: New master file (NF)
{
        OR ← Read OF
        TR ← Read TF
        while ((OR.k < Sentinel) OR (TR.k < Sentinel))
        {
                if (OR.k < TR.k)                    // The record in OF needs to be copied
                {
                        Write OR to NF
                        OR ← Read OF
                }
                if (OR.k > TR.k)                    // A new record needs to be added
                {
                        Write TR to NF
                        TR ← Read TF
                }
                if (OR.k = TR.k)
                {
                        if (ACT = C)                // Partial change in the record
                        {
                                Change OR
                                Write OR to NF
                        }
                        if (ACT = D) {}             // No action (empty block)
                        OR ← Read OF
                        TR ← Read TF
                }
        }
        return NF
}
```

41. The UML is shown in Figure S13.41. To simplify the diagram we assume that there is no error.

Figure S13.41 *Exercise 41*



Note that the above digram is similar to the digram shown in Exercise 37. The only difference is that we need to create a dummy record for each file if the file has reached its end. For example, when F1 reaches its end, we store the sentinel value in the key field of the of R1 (a dummy record). So in the next iteration, this record is not selected for processing. When both files reach their ends, the loop is terminated. Note that we first test the status of the end-of-file before reading a record from the file. This is needed because some system creates an error if we try to read from a file when the file has reached its end. Note that the routine works correctly even if one file or both files are empty. If one of the file is empty, the other file is copied to the new file. If both files are empty, the routine never enters the loop.

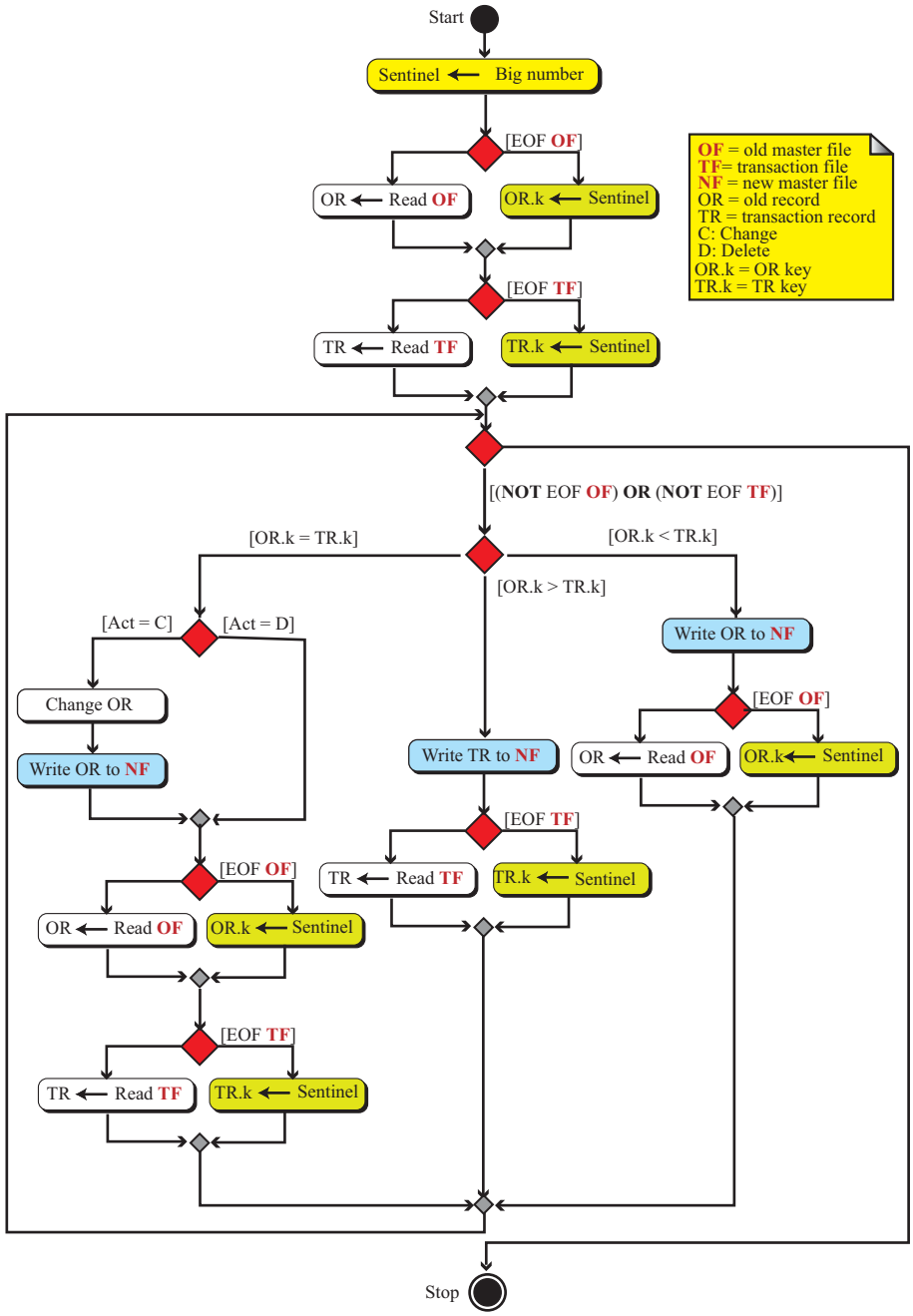
42. Algorithm S13.42 shows the routine in pseudocode. The algorithm exactly follow the UML in Exercise 41. If any of the two input files reach their ends, the routine creates a dummy record with the sentinel as the key value.

Algorithm S13.42 *Exercise 42*

```
Algorithm: MergeFile (F1, F2)
Purpose: It merges two sequential files
Pre: Given file 1 (F1) and file2 (F2)
Post:
Return: New file (NF)
{
        Sentinel ← A big number
        if (EOF F1)
                R1.k ← Sentinel
        else
                R1 ← Read F1
        if (EOF F2)
                R2.k ← Sentinel
        else
                R2 ← Read F2
        while ((NOT EOF F1) OR (NOT EOF F2))
        {
                if (R1.k < R2.k)
                {
                        Write R1 to NF
                        if (EOF F1)
                                R1.k ← Sentinel
                        else
                                R1 ← Read Fl
                }
                else
                {
                        Write R2 to NF
                        if (EOF F2)
                                R2.k ← Sentinel
                        else
                                R2 ← Read F2
                }
        }
        return NF
}
```

43. The UML is shown in Figure S13.43. To simplify the diagram we assume there is no error. The diagram combines the logic in Exercises 37, 39, and 41.

Figure S13.43 *Exercise 43*



44. Algorithm S13.44 shows the routine in pseudocode.

Algorithm S13.44

```
Algorithm: UpdateFile (OF, TR)
Purpose: It updates a sequential files
Pre: Given old master file (OF) and transaction file (TF)
Post:
Return: New master file (NF)
{
        Sentinel ← A big number
        if (EOF OF)                 OR.k ← Sentinel
        else                        OR ← Read OF
        if (EOF TR)                 TR.k ← Sentinel
        else                        TR ← Read TF
        while ((NOT EOF OF) OR (NOT EOF TF))
        {
                if (OR.k < TR.k)
                {
                        Write OR to NF
                        if (EOF OF)             OR.k ← Sentinel
                        else                    OR ← Read OF
                }
                if (OR.k > TR.k)
                {
                        Write TR to NF
                        if (EOF TF)             TR.k ← Sentinel
                        else                    TF ← Read TF
                }
                if (OR.k = TR.k)
                        if (Act = C)
                        {
                                Change OR
                                Write OR to NF
                        }
                if (EOF OF)             OR.k ← Sentinel
                else                    OR ← Read OF
                if (EOF TF)             TR.k ← Sentinel
                else                    TF ← Read TF
        }
        return NF
}
```