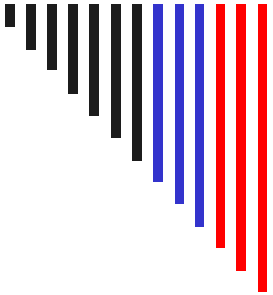


Chapter 13
Binary Input/Output

Objectives

- To understand the differences between text and binary files
- To write programs that read, write, and/or append binary files
- To be able to detect and take appropriate action when file errors occur
- To be able to process files randomly
- To be able to create text files from binary files and vice versa
- To understand and be able to implement file merges
- To understand and be able to implement the classic sequential file update

13-1 Text versus Binary Streams

In this section, we compare and contrast text streams versus binary streams.

Topics discussed in this section:

Text and Binary Files
Differences between Text and Binary Files
State of a File
Opening Binary Files
Closing Binary Files

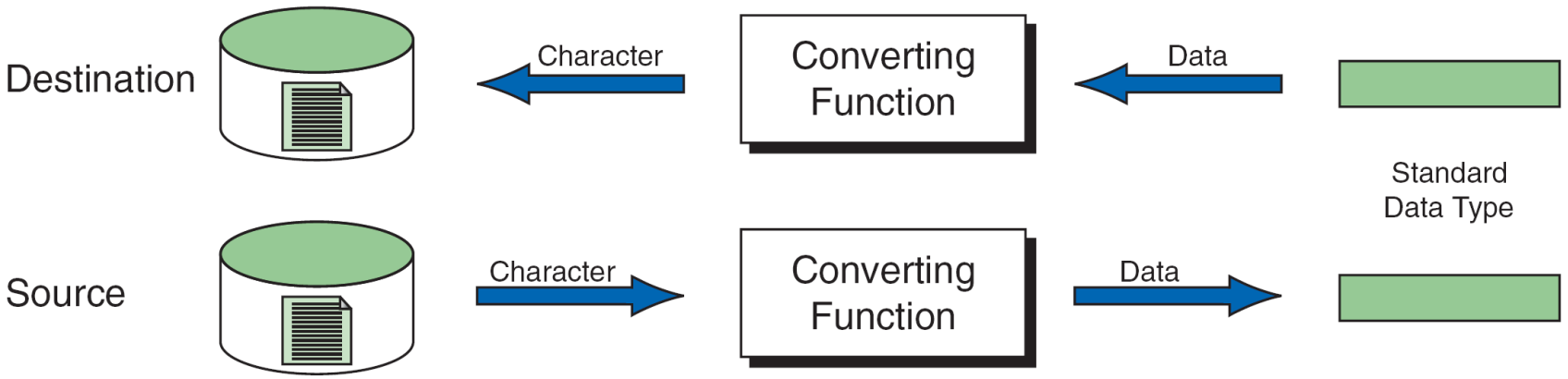


FIGURE 13-1 Reading and Writing Text Files

Note

Formatted input/output, character input/output, and string input/output functions can be used only with text files.

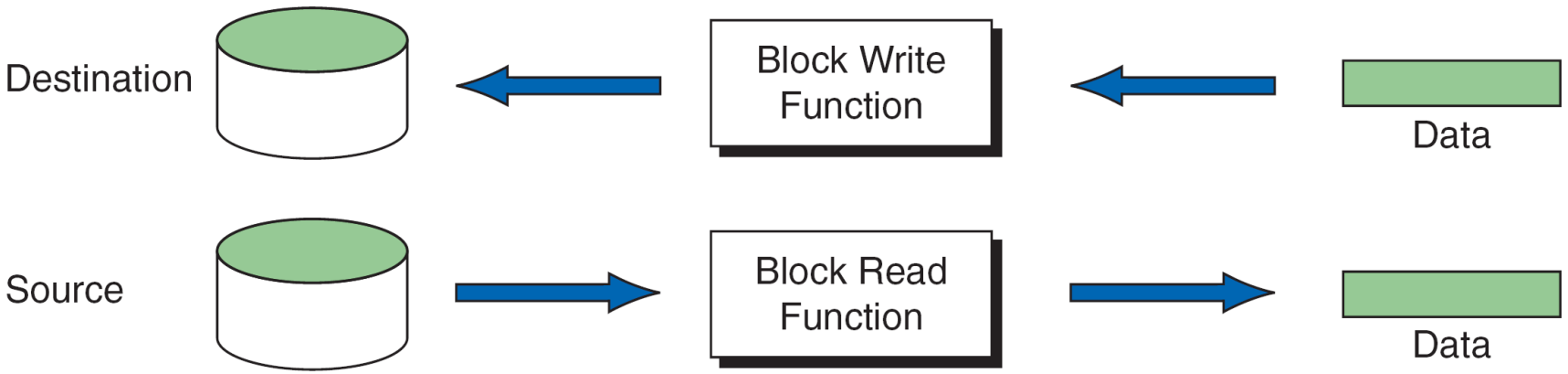


FIGURE 13-2 Block Input and Output

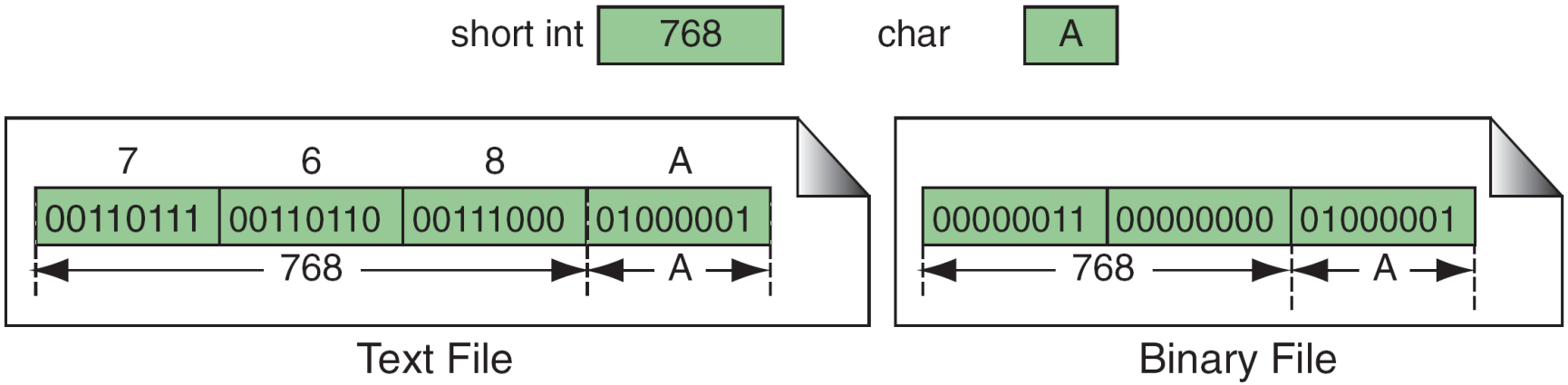


FIGURE 13-3 Binary and Text Files

Note

Text files store data as a sequence of characters; binary files store data as they are stored in primary memory.

Mode	r	w	a	r+	w+	a+
Open State	read	write	write	read	write	write
Read Allowed	yes	no	no	yes	yes	yes
Write Allowed	no	yes	yes	yes	yes	yes
Append Allowed	no	no	yes	no	no	yes
File Must Exist	yes	no	no	yes	no	no
Contents of Existing File Lost	no	yes	no	no	yes	no

Table 13-1 File Modes

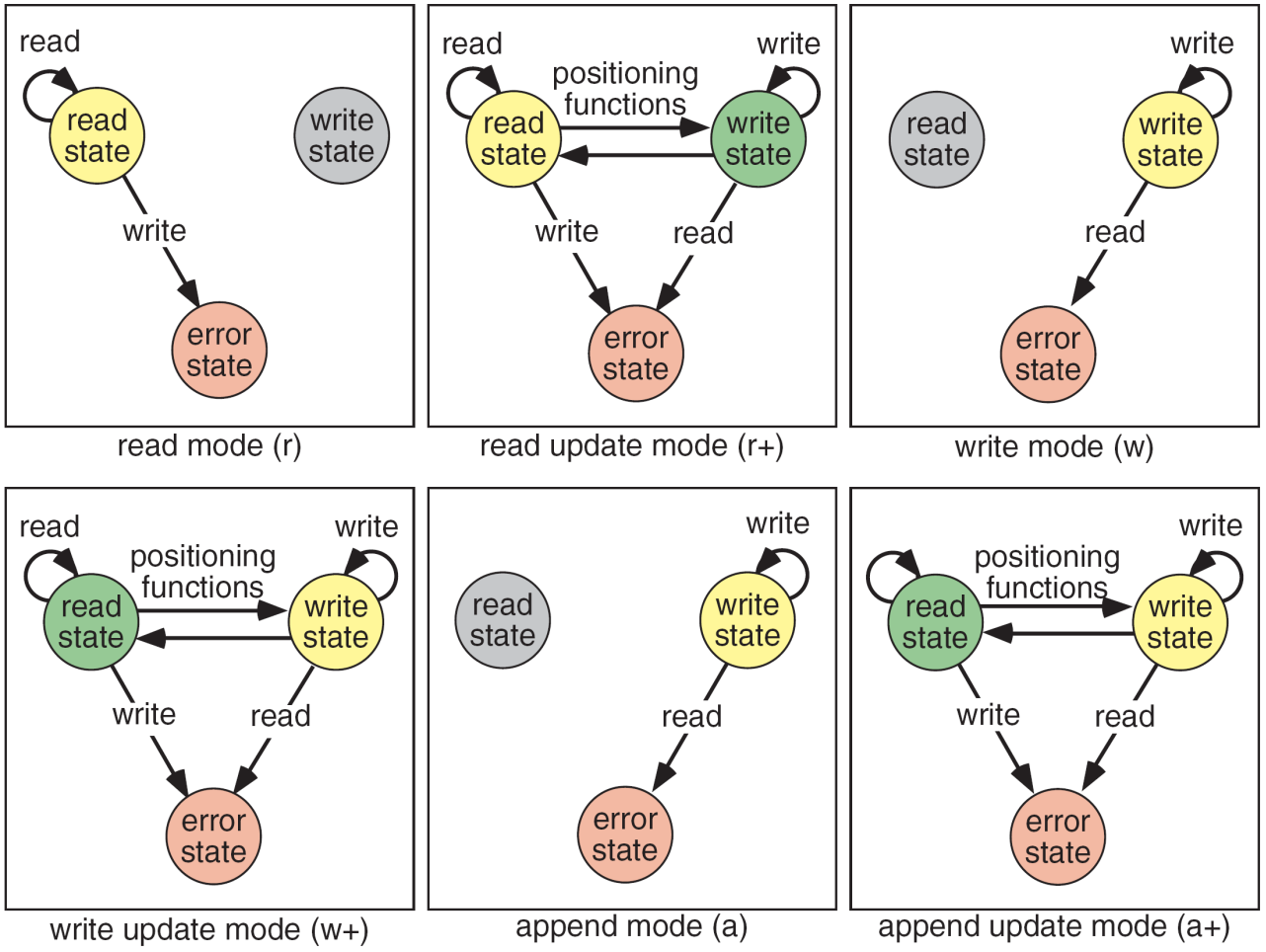


FIGURE 13-4 File States

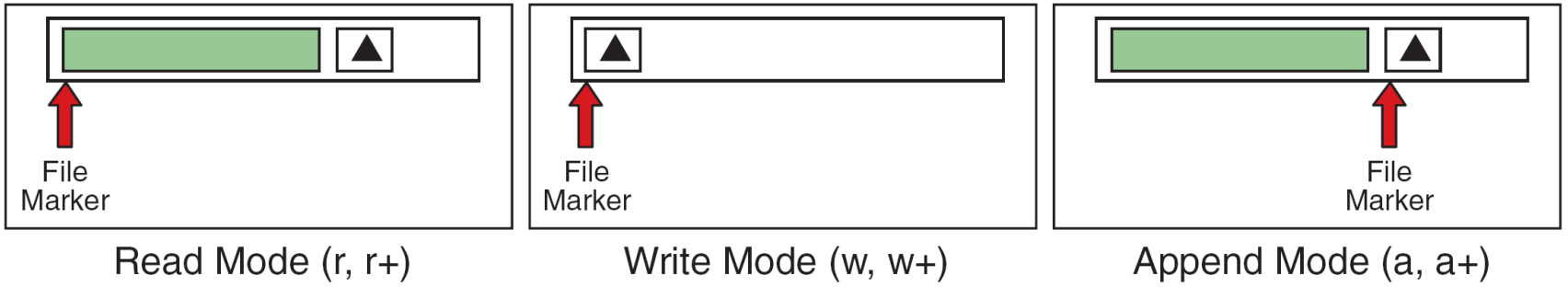


FIGURE 13-5 File-Opening Modes

Opening Binary Files

- spReadBin = fopen (“myFile.bin”, “rb”);
- spWriteBin = fopen (“myFile.bin”, “w+b”);
- spApndBin = fopen (“myFile.bin”, “ab”);

- fclose (spReadBin);

13-2 Standard Library Functions for Files

C has eight categories of standard file library functions. We have already discussed the first four in Chapter 7 and Chapter 11. We discuss the other four categories, which are more related to binary files, in this section.

Topics discussed in this section:

Block Input/Output Functions
File Status FunctionsComments
Positioning Functions
System File Operations

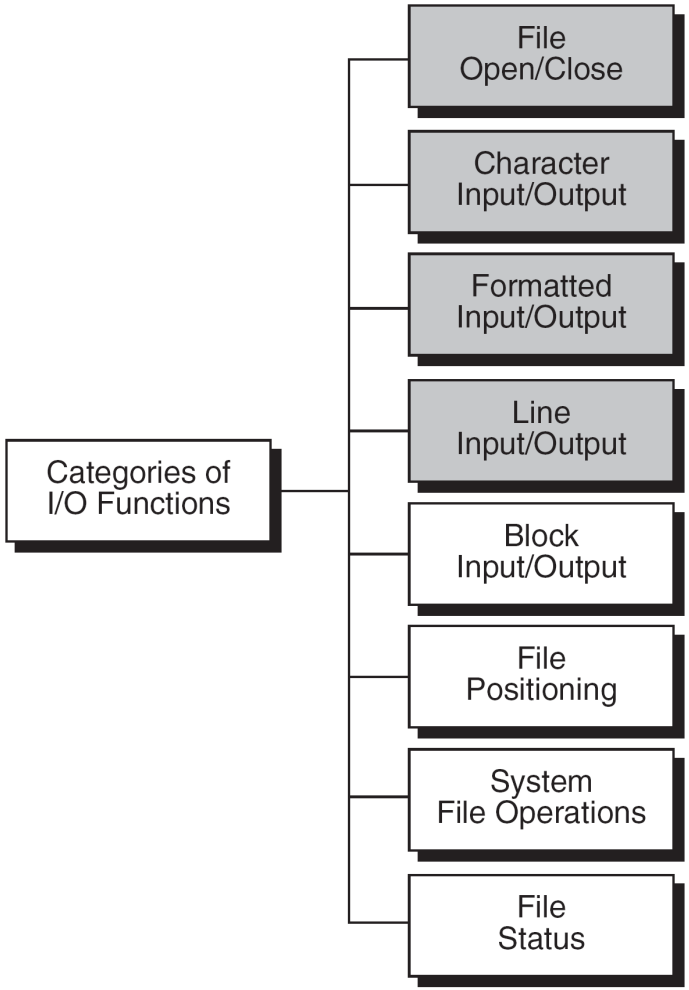
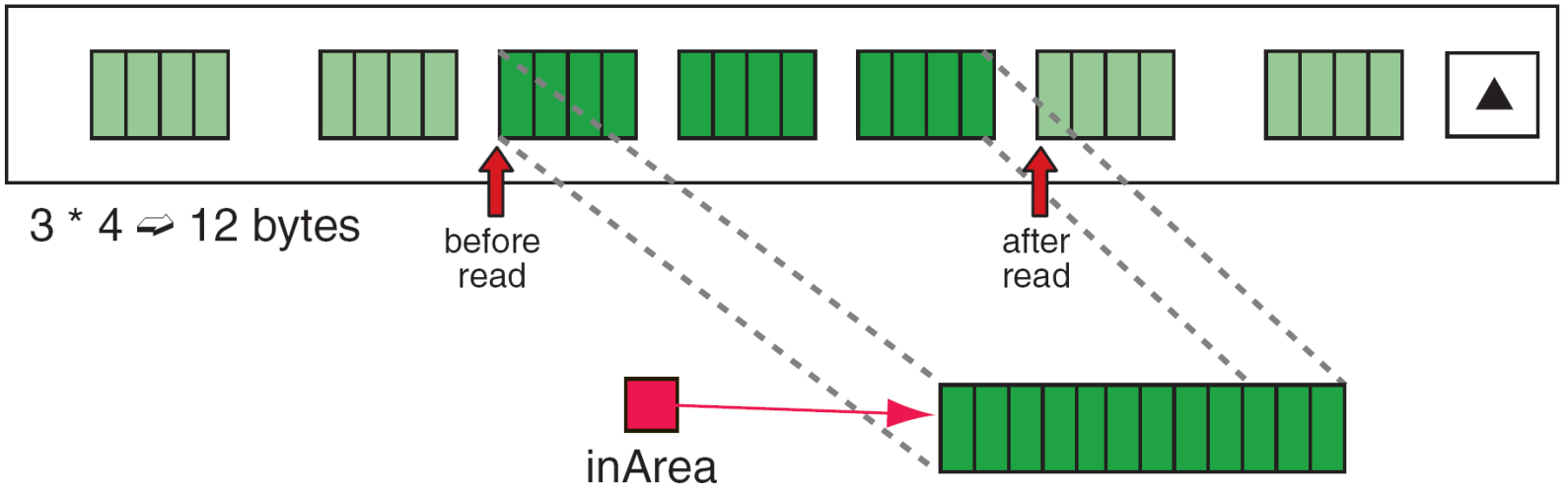


FIGURE 13-6 Types of Standard Input/Output Functions




```
fread (inArea, sizeof (int), 3, spData);
```

FIGURE 13-7 File Read Operation

PROGRAM 13-1 Read File of Integers

```
1  // Read a file of integers, three integers at a time.
2  {
3     …
4  // Local Declarations
5     FILE* spIntFile;
6     int   itemsRead;
7     int   intAry[3];
8
9  // Statements
10    spIntFile = fopen("int_file.dat", "rb");
11     …
12    while ((itemsRead = fread(intAry,
13              sizeof(int), 3, spIntFile)) !=  0)
14       {
15        // process array
16              …
17       } // while
18    …
19 }  // block
```

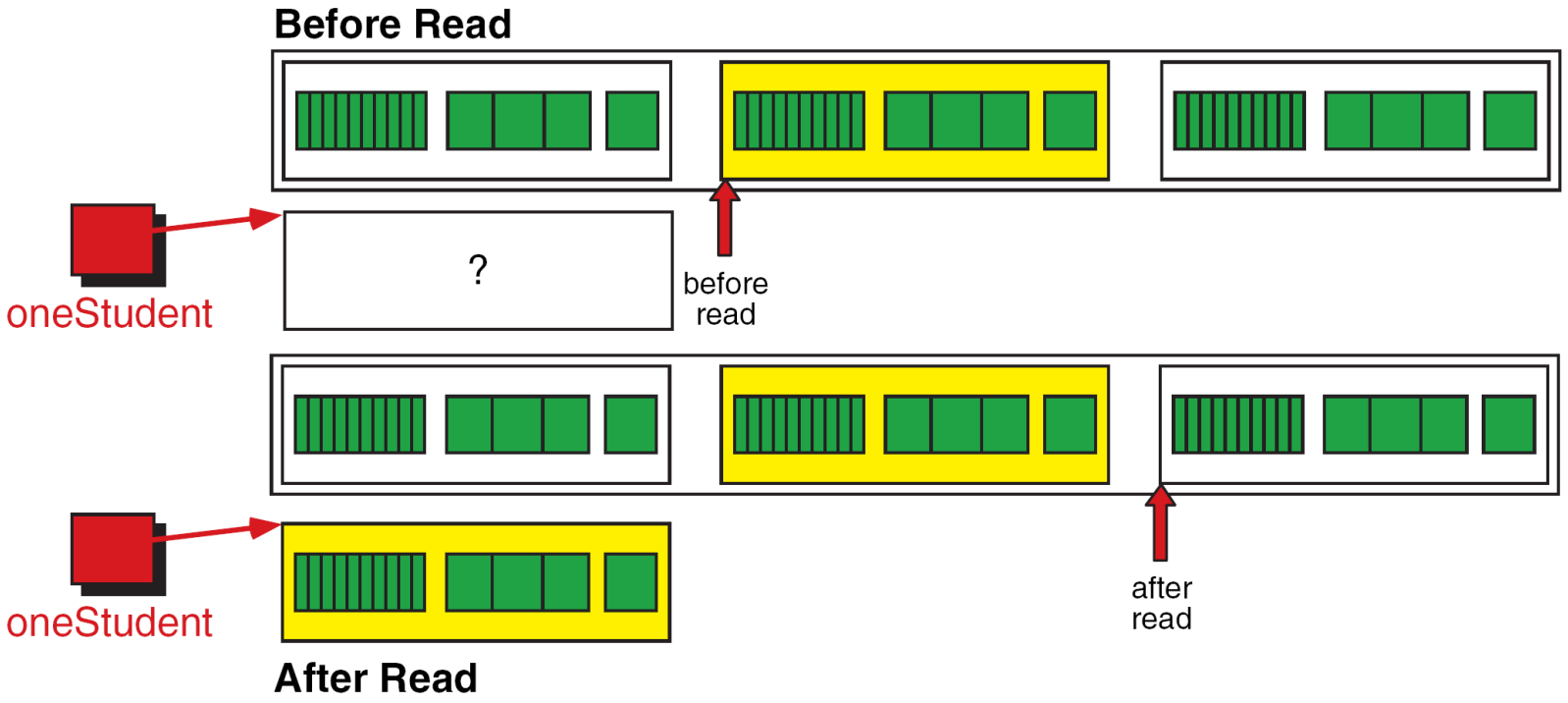


FIGURE 13-8 Reading a Structure

PROGRAM 13-2 Read Student File

```
1  /* Reads one student's data from a file
2        Pre   spStuFile is opened for reading
3        Post  stu data structure filled
4              ioResults returned
5  */
6  int readStudent (STU* oneStudent, FILE* spStuFile)
7  {
8  // Local Declarations
9     int ioResults;
10
11 // Statements
12    ioResults = fread(oneStudent,
13                      sizeof(STU), 1, spStuFile);
14    return ioResults;
15 } // readStudent
```

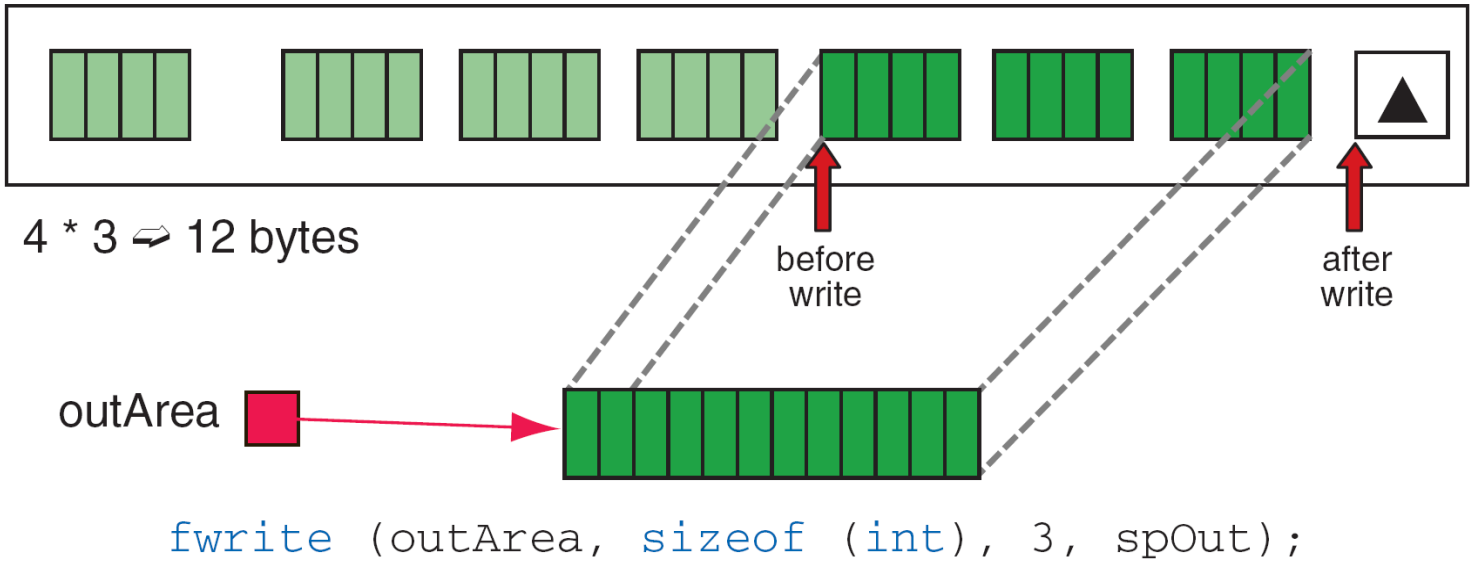


FIGURE 13-9 File Write Operation

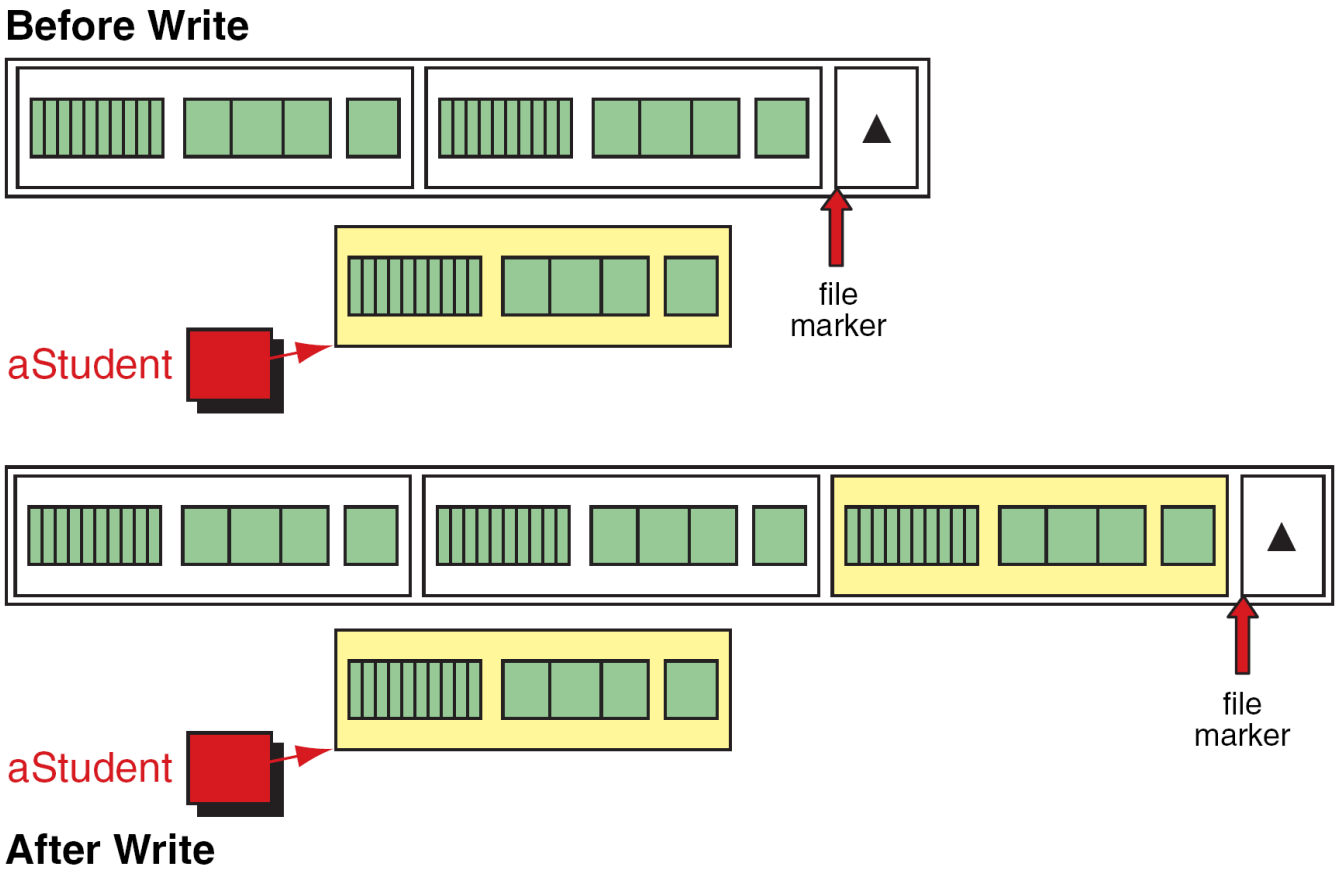


FIGURE 13-10 Writing a Structure

PROGRAM 13-3 Write Structured Data

```
 1  /* Writes one student's record to a binary file.
 2        Pre  aStudent has been filled
 3             spOut is open for writing
 4        Post aStudent written to spOut
 5  */
 6  void writeStudent (STU* aStudent, FILE* spOut)
 7
 8  {
 9  // Local Declarations
10     int ioResult;
11
12  // Statements
13     ioResult = fwrite(aStudent,
14                       sizeof(STU), 1, spOut);
15     if (ioResult != 1)
16        {
17         printf("\a Error writing student file \a\n");
18         exit (100);
19        } // if
20     return;
21  }  // writeStudent
```

File Status Functions

- feof
- ferror
- clearerr

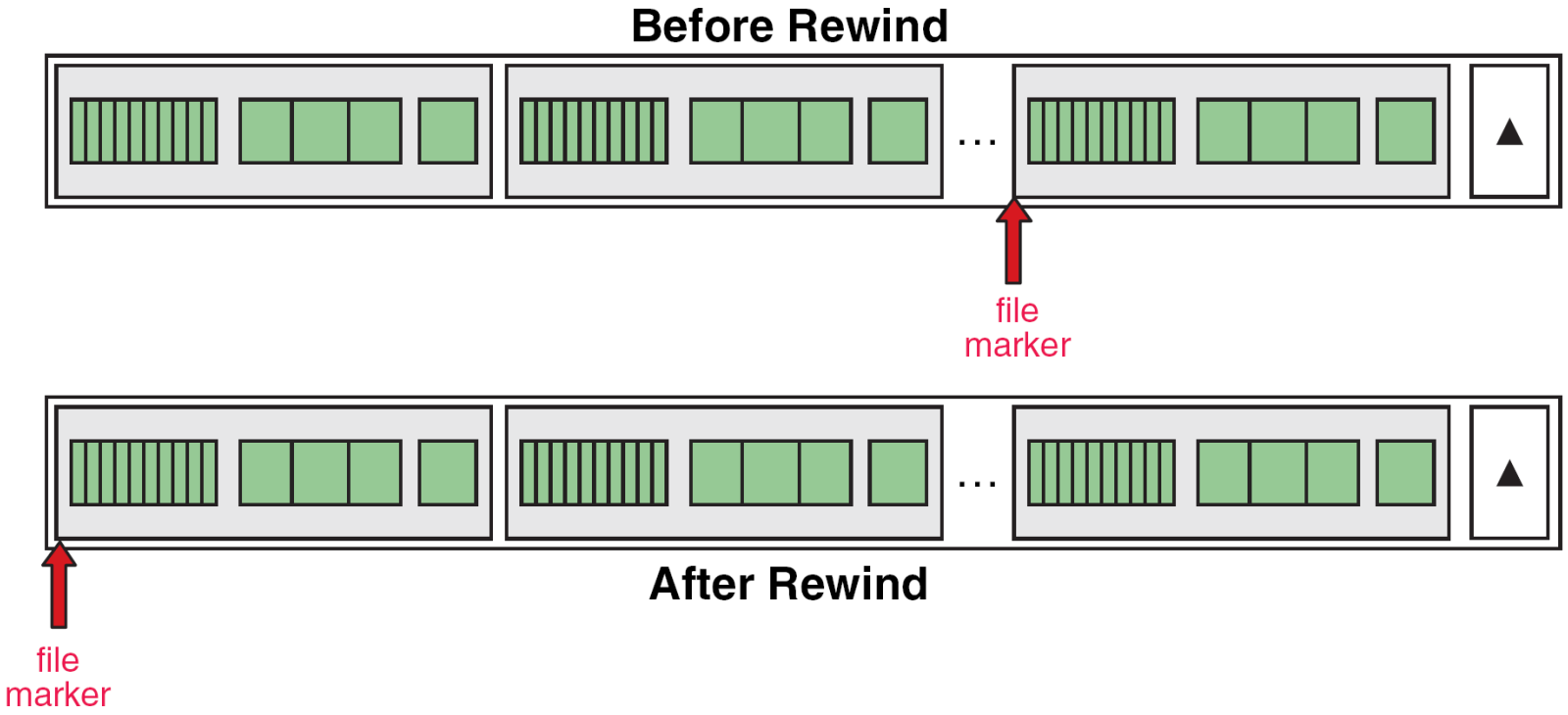


FIGURE 13-11 Rewind File

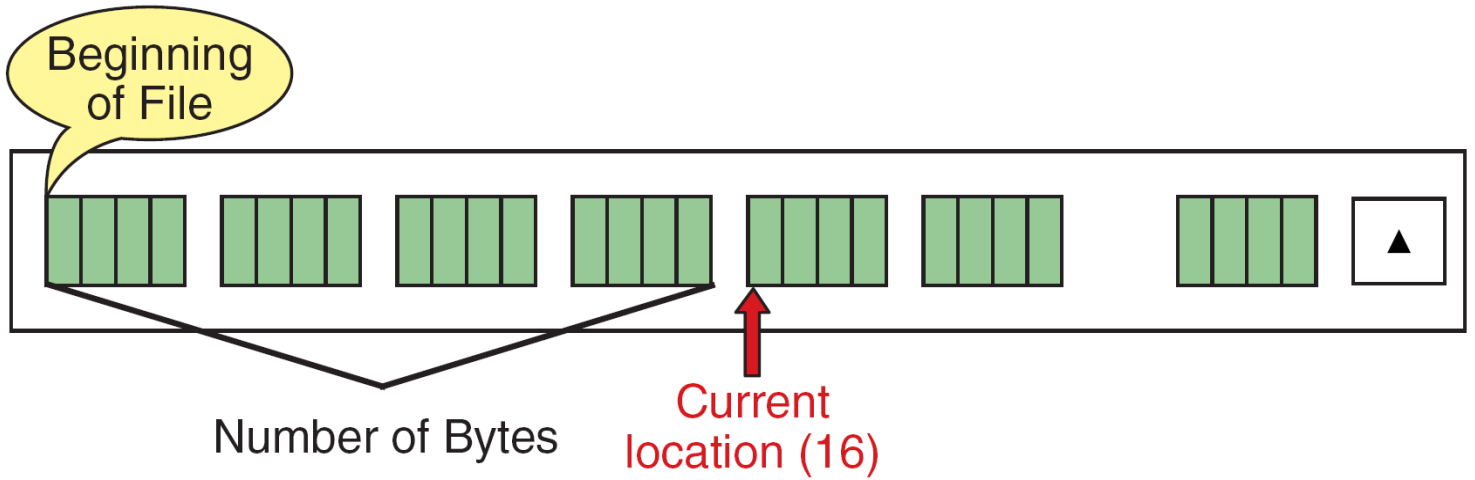


FIGURE 13-12 Current Location (*ftell*) Operation

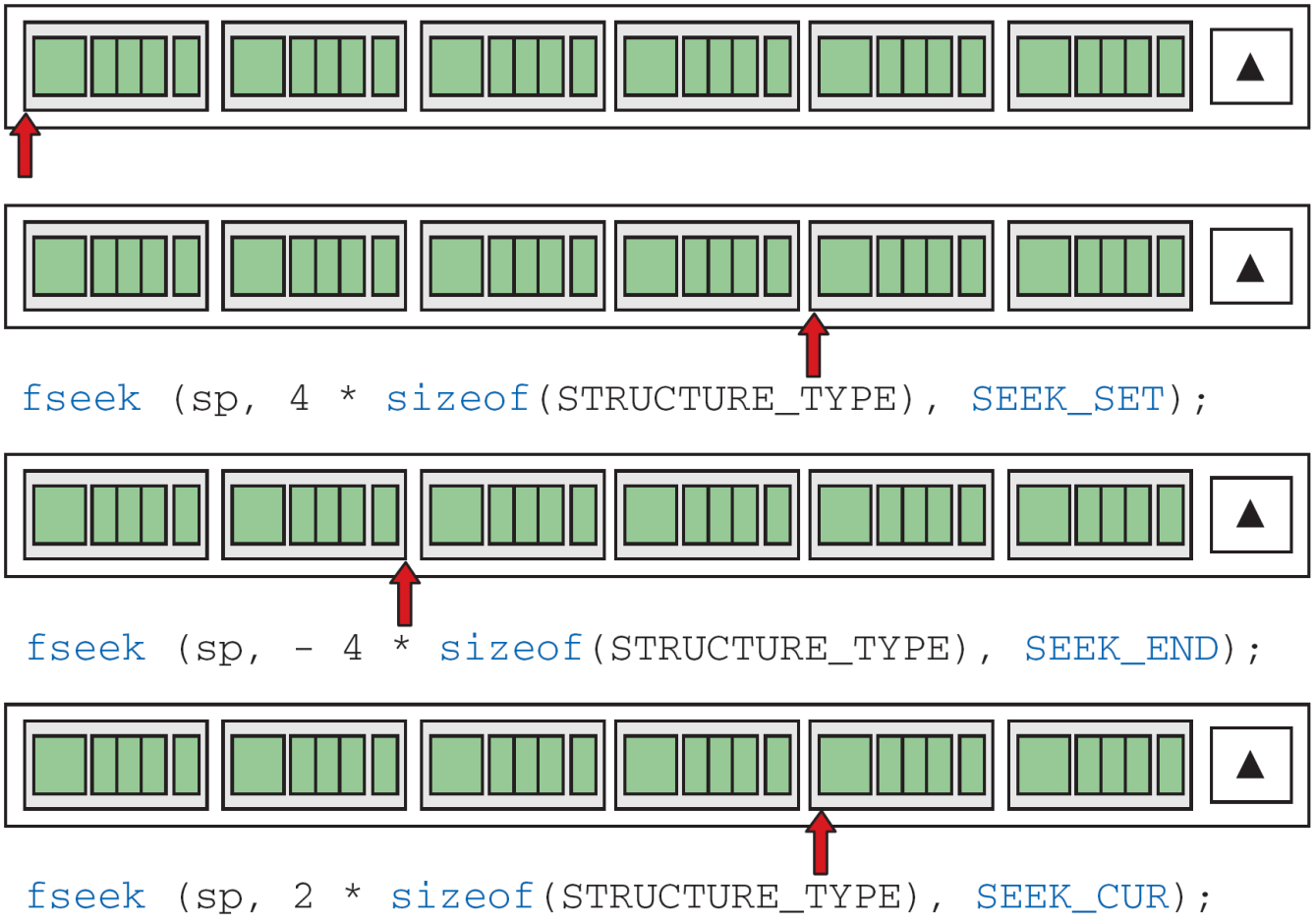


FIGURE 13-13 File Seek Operation

PROGRAM 13-4 Append Two Binary Files

```
1  /* This program appends two binary files of integers.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  #include <stdlib.h>
7
8  int main (void)
9  {
10 // Local Declarations
11 FILE* sp1;
12 FILE* sp2;
13 int   data;
14 long  dataCount;
15 char  fileID[13];
16
```

PROGRAM 13-4 Append Two Binary Files

```
17  // Statements
18     printf("This program appends two files.\n");
19     printf("Please enter file ID of the primary file: ");
20     scanf("%12s", fileID);
21     if (!(sp1 = fopen (fileID, "ab")))
22         printf("\aCan't open %s\n", fileID), exit (100);
23
24     if (!(dataCount = (ftell (sp1))))
25        printf("\a%s has no data\n", fileID), exit (101);
26     dataCount /= sizeof(int);
27
28     printf("Please enter file ID of the second file: ");
29     scanf("%12s", fileID);
30     if (!(sp2 = fopen (fileID, "rb")))
31        printf("\aCan't open %s\n", fileID), exit (110);
32
33     while (fread (&data, sizeof(int), 1, sp2) == 1)
34        {
35         fwrite (&data, sizeof(int), 1, sp1);
36         dataCount++;
37        } // while
```

PROGRAM 13-4 Append Two Binary Files

```
38
39      if (! feof(sp2))
40         printf("\aRead Error. No output.\n"), exit (120);
41
42      fclose (sp1);
43      fclose (sp2);
44
45      printf("Append complete: %ld records in file\n",
46              dataCount);
47      return 0;
48  }   // main
```

System File Operations

- remove
- rename
- tmpfile

13-3 Converting File Type

A rather common but somewhat trivial problem is to convert a text file to a binary file and vice versa. C has no standard functions for these tasks. We must write a program to make the conversion. We describe the file conversion logic in this section.

Topics discussed in this section:

Creating a Binary File from a Text File
Creating a Text File from a Binary File

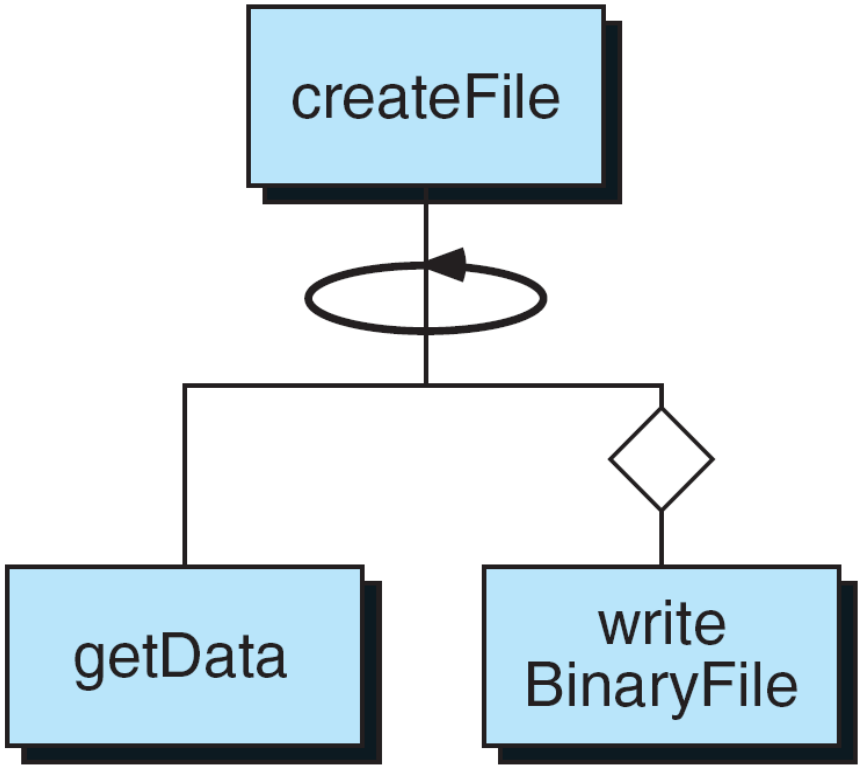


FIGURE 13-14 Create Binary File Structure Chart

PROGRAM 13-5 Text to Binary Student File

```
1  /* Reads text file of student data & creates binary file.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <stdbool.h>
8
9  // Type Declarations
10 typedef struct stuData
11    {
12     char  name[26];
13     char  id[5];
14     int   exams[3];
15     int   problems[8];
16     char  grade;
17    } STU_DATA;
18
```

PROGRAM 13-5 Text to Binary Student File

```
19 // Function Declarations
20 bool getData          (FILE*     textFile,
21                        STU_DATA* aStudent);
22 void writeBinaryFile (STU_DATA* aStudent,
23                        FILE*     binFile);
24
25 int main (void)
26 {
27 // Local Declarations
28    char* textFileID = "P13-stu.txt";
29    char* binFileID  = "P13-stu.bin";
30
31    STU_DATA aStudent;
32
33    FILE* textFile;
34    FILE* binFile;
35
```

PROGRAM 13-5 Text to Binary Student File

```
36  // Statements
37     printf("\nBegin Student Binary File Creation\n ");
38     if (!(textFile = fopen(textFileID, "r")))
39        {
40         printf("\nCannot open %s\n", textFileID);
41         exit (100);
42        } // if textFile
43     if (!(binFile = fopen(binFileID, "wb")))
44        {
45         printf("\nCannot open %s\n", binFileID);
46         exit (200);
47        } // if binFile
48
49     while (getData (textFile, &aStudent))
50        writeBinaryFile (&aStudent, binFile);
51
52     fclose(textFile);
53     fclose(binFile);
```

PROGRAM 13-5 Text to Binary Student File

```
54       printf("\n\nFile creation complete\n");
55       return 0;
56    }  // main
57
58    /* ==================== getData ====================
59       This function reads the text file.
60          Pre   textFile is opened for reading
61          Post  data read and returned
62    */
63    bool getData (FILE* textFile, STU_DATA* aStu)
64    {
65    // Local Declarations
66       char  buffer[100];
67
68    // Statements
69       fgets(buffer, sizeof(buffer), textFile);
70       if (!feof(textFile))
71          {
```

PROGRAM 13-5 Text to Binary Student File

```
71          {
72           sscanf(buffer, "%s %s %d%d%d%d%d%d%d%d%d%d%d %c",
73              aStu->name, aStu->id,
74             &aStu->exams[0],
75             &aStu->exams[1],&aStu->exams[2],
76             &aStu->problems[0], &aStu->problems[1],
77             &aStu->problems[2], &aStu->problems[3],
78             &aStu->problems[4], &aStu->problems[5],
79             &aStu->problems[6], &aStu->problems[7],
80             &aStu->grade);
81           return true;
82          } // if
83       return false;
84    } // getData
85
86    /* ================= writeBinaryFile ================
87       Write student data to a binary file.
88          Pre   binFile is opened as a binary output file
89                aStudent is complete
90          Post  Record written
91    */
```

PROGRAM 13-5 Text to Binary Student File

```
 92 void writeBinaryFile (STU_DATA* aStudent,
 93                       FILE* binFile)
 94 {
 95 // Local Declarations
 96    int amtWritten;
 97
 98 // Statements
 99    amtWritten = fwrite (aStudent,
100                         sizeof(STU_DATA), 1, binFile);
101    if (amtWritten != 1)
102       {
103        printf("Can't write student file. Exiting\n");
104        exit (201);
105       } // if
106    return;
107 } // writeBinaryFile
```

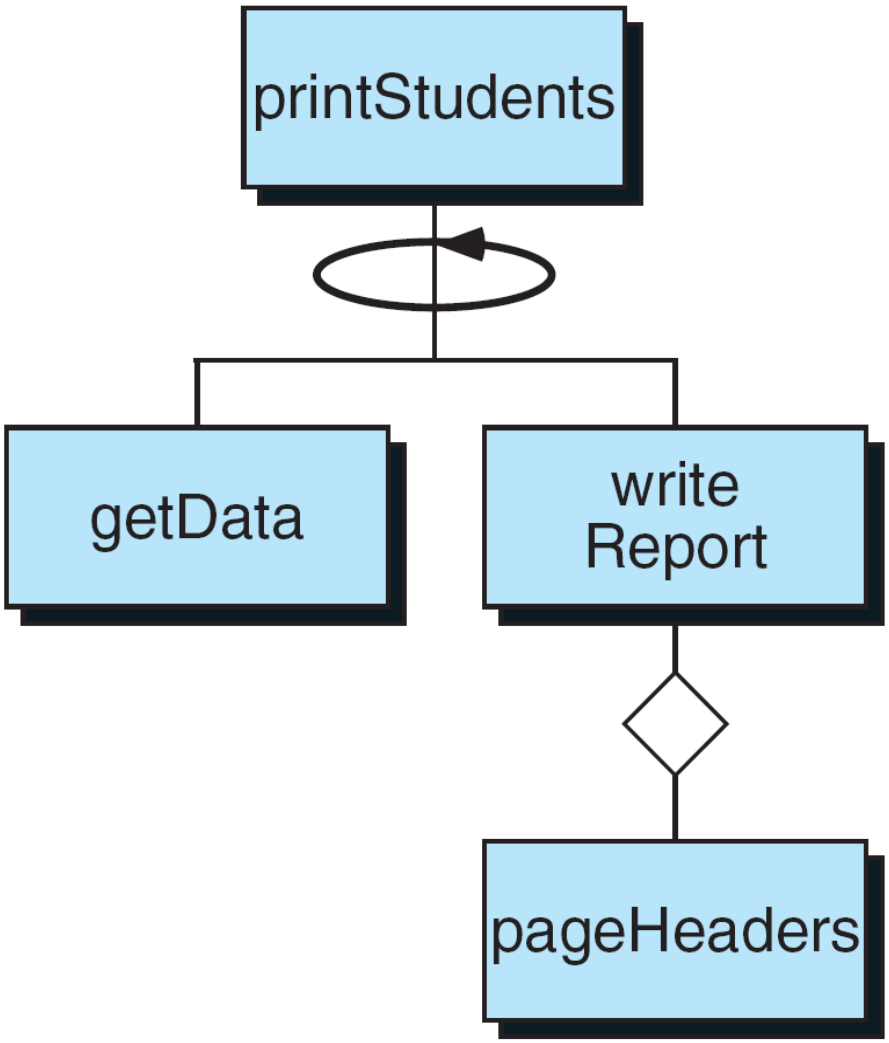


FIGURE 13-15 Design for Print Student Data

PROGRAM 13-6 Print Student Data

```
 1  /* Reads a binary file of student data, and prints it.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6  #include <stdlib.h>
 7
 8  #define MAX_LINES_PER_PAGE  50
 9  #define BUFFER_SIZE        133
10  #define FORM_FEED          '\f'
11
12  // Type Declarations
13  typedef struct stuData
14     {
15      char  name[26];
16      char  id[5];
17      int   exams[3];
18      int   problems[8];
19      char  grade;
20     } STU_DATA;
```

PROGRAM 13-6 Print Student Data

```
21
22  // Function Declarations
23  STU_DATA getData      (FILE* binFile);
24  void     writeReport  (STU_DATA aStudent,
25                         FILE* prtFile);
26  void     pageHeaders  (FILE* prtFile);
27
28  int main (void)
29  {
30  // Local Declarations
31     char       stuFileID[]  = "P13-stu.bin";
32     char       prtFileID[]  = "P13-stu.prt";
33     STU_DATA   aStudent;
34     FILE*      stuFile;
35     FILE*      prtFile;
36
```

PROGRAM 13-6 Print Student Data

```
37  // Statements
38     printf("\nBegin Student Report Creation\n ");
39
40     if(!(stuFile = fopen(stuFileID, "rb")))
41        {
42         printf("\nCannot open %s\n", stuFileID);
43         exit (100);
44        } // if stuFile
45     if (!(prtFile = fopen(prtFileID, "w")))
46        {
47         printf("\nCannot open %s\n", prtFileID);
48         exit (200);
49        } // if prtFile
50
51     aStudent = getData (stuFile);
52     while (!feof(stuFile))
53        {
54         writeReport (aStudent, prtFile);
55         aStudent = getData (stuFile);
```

PROGRAM 13-6 Print Student Data

```
56          } // while
57
58      fprintf(prtFile, "\nEnd of Report\n");
59      fclose(stuFile);
60      fclose(prtFile);
61      printf("\n\nEnd Student Report Creation\n");
62      return 0;
63  }   // main
64
65  /* ==================== getData ====================
66      This function reads the student binary file.
67         Pre   stuFile is opened for reading
68         Post  one student record read and returned
69  */
70  STU_DATA getData (FILE* stuFile)
71  {
72  // Local Declarations
73      int        ioResult;
74      STU_DATA   aStu;
75
```

PROGRAM 13-6 Print Student Data

```
76  // Statements
77     ioResult = fread(&aStu,
78                      sizeof(STU_DATA), 1, stuFile);
79     if (!ioResult)
80        if (!feof(stuFile))
81           {
82            printf("\n\nError reading student file\n");
83            exit (100);
84           } // if !feof
85     return aStu;
86  } // getData
87
88  /* ================== writeReport ====================
89     Write student report to a text file.
90        Pre   prtFile is opened as a text output file
91              aStudent is complete
92        Post  Report line written
93  */
```

PROGRAM 13-6 Print Student Data

```
 94 void writeReport (STU_DATA  aStu, FILE* prtFile)
 95 {
 96 // Local Declarations
 97    static int lineCount = MAX_LINES_PER_PAGE + 1;
 98    char   buffer[BUFFER_SIZE];
 99
100 // Statements
101    if (++lineCount > MAX_LINES_PER_PAGE)
102       {
103        pageHeaders  (prtFile);
104        lineCount = 1;
105       } // if
106
107    sprintf (buffer,
108      "%-25s %4s %4d%4d%4d%4d%4d%4d%4d%4d%4d%4d%4d %c\n",
109       aStu.name, aStu.id,
110       aStu.exams[0],    aStu.exams[1],    aStu.exams[2],
111       aStu.problems[0],
112       aStu.problems[1], aStu.problems[2],
113       aStu.problems[3], aStu.problems[4],
```

PROGRAM 13-6 Print Student Data

```
114          aStu.problems[5],
115          aStu.problems[6], aStu.problems[7],
116          aStu.grade);
117       fputs (buffer, prtFile);
118       return;
119    } // writeReport
120
121    /* ================== pageHeaders ==================
122       Writes the page headers for the student report.
123          Pre   prtFile is opened as a text output file
124          Post  Report headers and captions written
125    */
126    void pageHeaders (FILE* prtFile)
127    {
128    // Local Declarations
129       static int pageNo = 0;
130
```

PROGRAM 13-6 Print Student Data

```
131 // Statements
132    pageNo++;
133    fprintf(prtFile, "%c", FORM_FEED);
134    fprintf(prtFile, "%-66s  Page %4d\n",
135            "Student Report ", pageNo);
136    fprintf(prtFile, "%-25s %-6s %-10s %-27s Grade\n\n",
137            "Student Name", "ID", "Exams", "Problems");
138    return;
139 }  // pageHeaders
140 // ================= End of Program =================
```

13-4 File Program Examples

This section contains two common file applications. The first uses the file positioning functions to randomly process the data in a file. The second merges two files.

Topics discussed in this section:

Random File Processing

Merge Files

PROGRAM 13-7 Random File Application

```
 1  /* Shows application of some functions we have studied
 2     in this chapter. The program first creates a binary
 3     file of integers. It then prints the file, first
 4     sequentially and then randomly using rand().
 5        Written by:
 6        Date:
 7  */
 8  #include <stdio.h>
 9  #include <stdlib.h>
10
11  // Function Declarations
12  void buildFile   (FILE** sp);
13  void printFile   (FILE*  sp);
14  void randomPrint (FILE*  sp);
15
16  int main (void)
17  {
18  // Local Declarations
19     FILE* fpData;
```

PROGRAM 13-7 Random File Application

```
20
21  // Statements
22      buildFile   (&fpData);
23      printFile   (fpData);
24      randomPrint (fpData);
25      return 0;
26  }   // main
```

PROGRAM 13-8 Random File: Build File

```
1  /* ================== buildFile ==================
2     Creates a disk file that we can process randomly.
3        Pre   nothing
4        Post  file has been built
5  */
6  void buildFile (FILE** spData)
7  {
8  // Local Declarations
9     int  data;
10
11 // Statements
12    if (!(*spData = fopen("SAMPLE.DAT", "w+b")))
13       {
14        printf("\aError opening file.\n");
15        exit (100);
16       } // if open
17    for (int i = 1; i <= 10; i++)
```

PROGRAM 13-8 Random File: Build File

```
18          {
19           data = i * i;
20           fwrite(&data, sizeof(int), 1, *spData);
21          } // for
22      return;
23  }  // buildFile
```

PROGRAM 13-9 Random File: Sequential Print

```
1  /* ==================== printFile ====================
2     Prints the file starting at the first record.
3        Pre   spData is an open file
4        Post  file has been printed
5  */
6  void printFile (FILE* spData)
7  {
8  // Local Declarations
9     int data;
10    int recNum;
11
12 // Statements
13    recNum = 0;
14    rewind(spData);
15    fread(&data , sizeof(int), 1, spData);
16    while (!feof(spData))
17       {
18        printf("Record %2d: %3d\n", recNum++, data);
19        fread(&data, sizeof(int), 1, spData);
```

PROGRAM 13-9 Random File: Sequential Print

```
20          } // while
21      return;
22  }  // printFile
```

```
Results:
Record 0:   1
Record 1:   4
Record 2:   9
Record 3:  16
Record 4:  25
Record 5:  36
Record 6:  49
Record 7:  64
Record 8:  81
Record 9: 100
```

PROGRAM 13-10 Random File: Random Print

```
1  /* ================== randomPrint ==================
2     This function randomly prints the file. Some data
3     may be printed twice, depending on the random
4     numbers generated.
5        Pre  spData is an open file
6        Post Ten records have been printed
7  */
8  void randomPrint (FILE* spData)
9  {
10 // Local Declarations
11    int data;
12    int randomSeek;
13
14 // Statements
15    printf("\nFile contents in random sequence.\n");
16    for (int i = 0; i < 10; i++)
17       {
18        randomSeek =  (rand() % 10);
19        fseek(spData,
20              sizeof(int) * randomSeek, SEEK_SET);
21        fread(&data, sizeof(int), 1, spData);
```

PROGRAM 13-10 Random File: Random Print

```
22          printf("Record %3d ==> %3d\n",
23                  randomSeek, data);
24        } // for
25     return;
26  }  // randomPrint
```

```
Results:
File contents in random sequence.
Record   8 ==>  81
Record   8 ==>  81
Record   3 ==>  16
Record   5 ==>  36
Record   1 ==>   4
Record   7 ==>  64
Record   0 ==>   1
Record   9 ==> 100
Record   2 ==>   9
Record   6 ==>  49
```

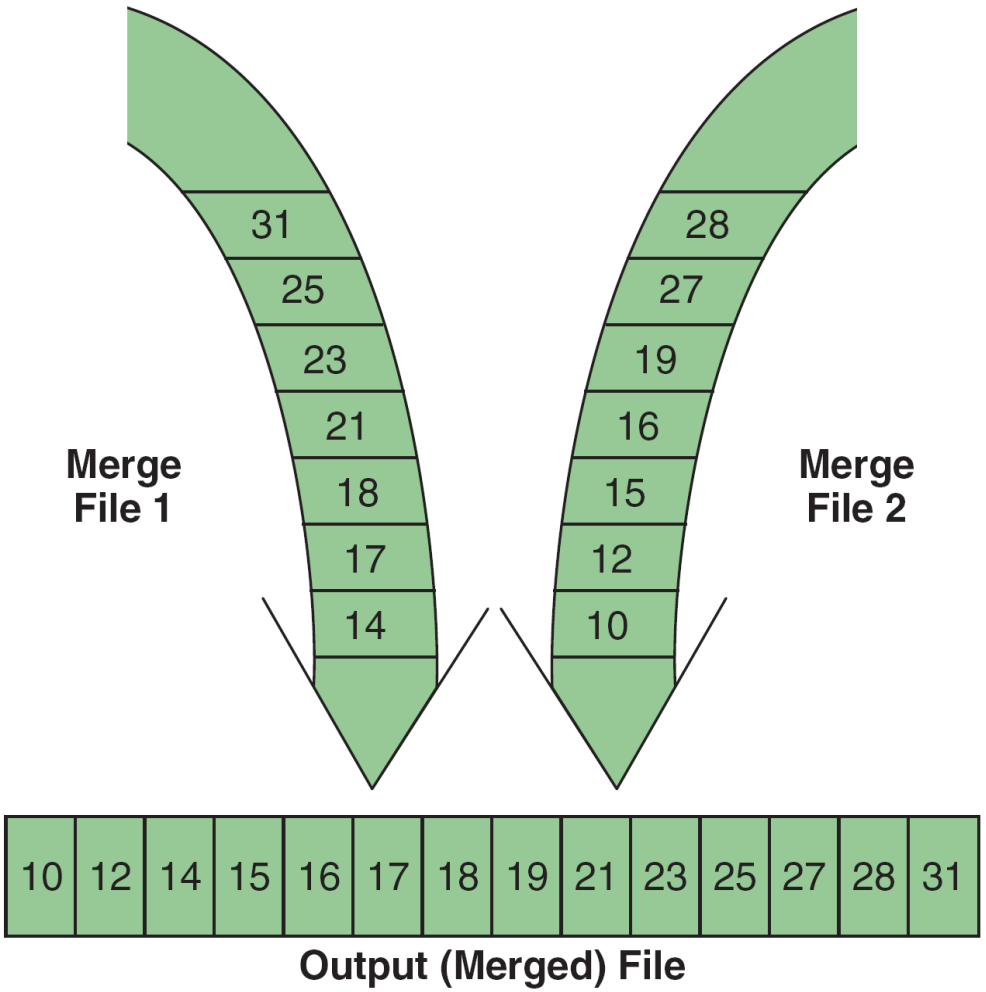


FIGURE 13-16 File Merge Concept

ALGORITHM 13-1 Pseudocode for Merging Two Files

```
Algorithm MergeTwoFiles
This program merges two files
 1  input (File1, Rec1)
 2  input (File2, Rec2)
 3  HighSentinel = high-value
 4  loop (not eof(File1)) OR (not eof(File2))
    1   if  Rec1.Key  <=  Rec2.Key  then
        1  output (File3, Rec1)
        2  input  (File1, Rec1)
        3  if eof(File1)
           1  Rec1.Key = HighSentinel
        4  end if
    2   else
        1  output (File3, Rec2)
        2  input  (File2, Rec2)
        3  if eof(File2)
           1  Rec2.Key = HighSentinel
        4  end if
    3   end if
 5  end loop
End MergeTwoFiles
```

PROGRAM 13-11 Merge Two Files

```
1  /* This program merges two files
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <limits.h>
8
9  #define READ_MODE  "rb"
10 #define WRITE_MODE "wb"
11
12 int main (void)
13 {
14 // Local Declarations
15    FILE*  spM1;
16    FILE*  spM2;
17    FILE*  spOut;
18
```

PROGRAM 13-11 Merge Two Files

```
19      int  recM1;
20      int  recM2;
21      int  sentinel    = INT_MAX;
22      int  mergeCnt    = 0;
23
24      char file1ID[]   = "P13Mrg1.bin";
25      char file2ID[]   = "P13Mrg2.bin";
26      char fileOutID[] = "P13Mrg3.bin";
27
28  // Statements
29      printf("Begin File Merge:\n");
30      if (!(spM1 = fopen (file1ID, READ_MODE)))
31          printf("\aError on %s\n", file1ID), exit (100);
32
33      if (!(spM2 = fopen (file2ID, READ_MODE)))
34          printf("\aError on %s\n", file2ID), exit (200);
35
36      if (!(spOut = fopen (fileOutID, WRITE_MODE)))
37         printf("\aError on %s\n", fileOutID), exit (300);
38
```

PROGRAM 13-11 Merge Two Files

```
39      fread (&recM1, sizeof(int), 1, spM1);
40      if (feof(spM1))
41           recM1 = sentinel;
42      fread (&recM2, sizeof(int), 1, spM2);
43      if (feof(spM2))
44          recM2 = sentinel;
45
46      while (!feof(spM1) || !feof(spM2))
47         {
48          if (recM1 <=  recM2)
49             {
50              fwrite (&recM1, sizeof(int), 1, spOut);
51              mergeCnt++;
52              fread  (&recM1, sizeof(int), 1, spM1);
53              if (feof(spM1))
54                 recM1 = sentinel;
55             } // if
56          else
57             {
```

PROGRAM 13-11 Merge Two Files

```
58              fwrite (&recM2, sizeof(int), 1, spOut);
59              mergeCnt++;
60              fread  (&recM2, sizeof(int), 1, spM2);
61              if (feof(spM2))
62                 recM2 = sentinel;
63            } // else
64        } // while
65     fclose (spM1);
66     fclose (spM2);
67     fclose (spOut);
68     printf("End File Merge. %d items merged.\n",
69              mergeCnt);
70     return 0;
71  }  // main
```

13-5 Software Engineering

Any file environment requires some means of keeping the file current. The function that keeps files current is known as* updating. *To complete our discussion of files, we discuss some of the software engineering design considerations for file updating.

Topics discussed in this section:

Update Files
Sequential File Update
The Update Program Design
Update Errors
Update Structure Chart and Logic

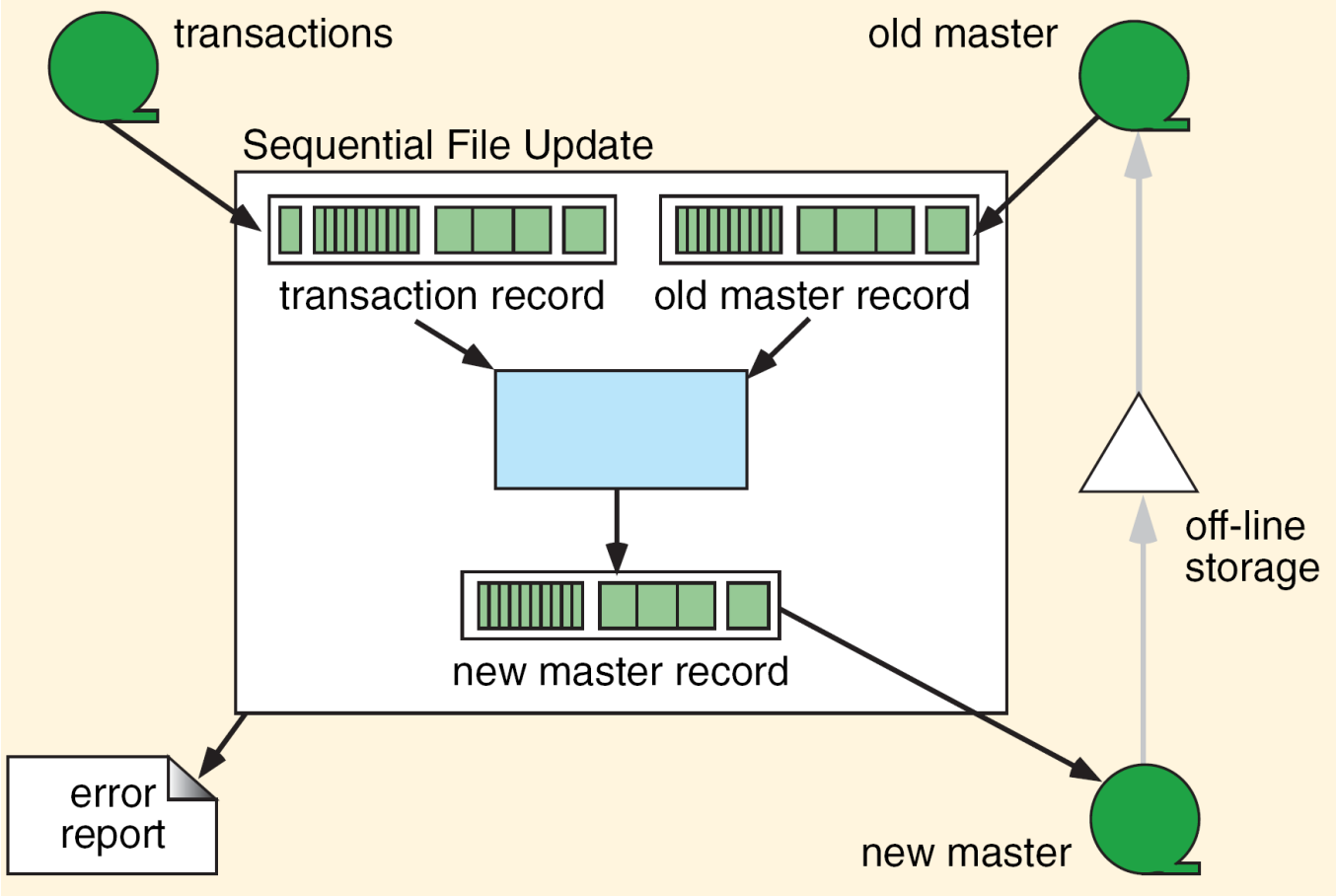


FIGURE 13-17 Sequential File Update Environment

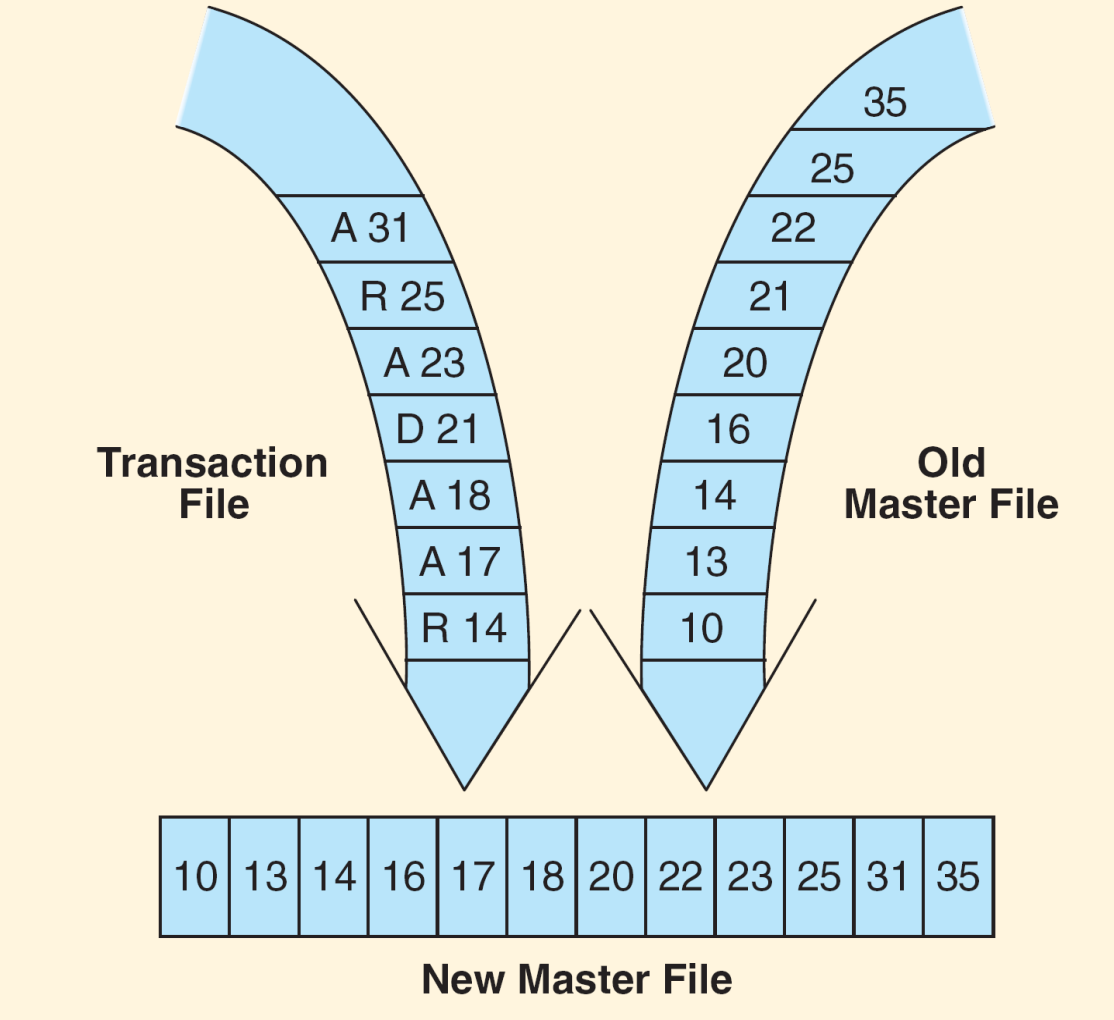


FIGURE 13-18 File Updating Example

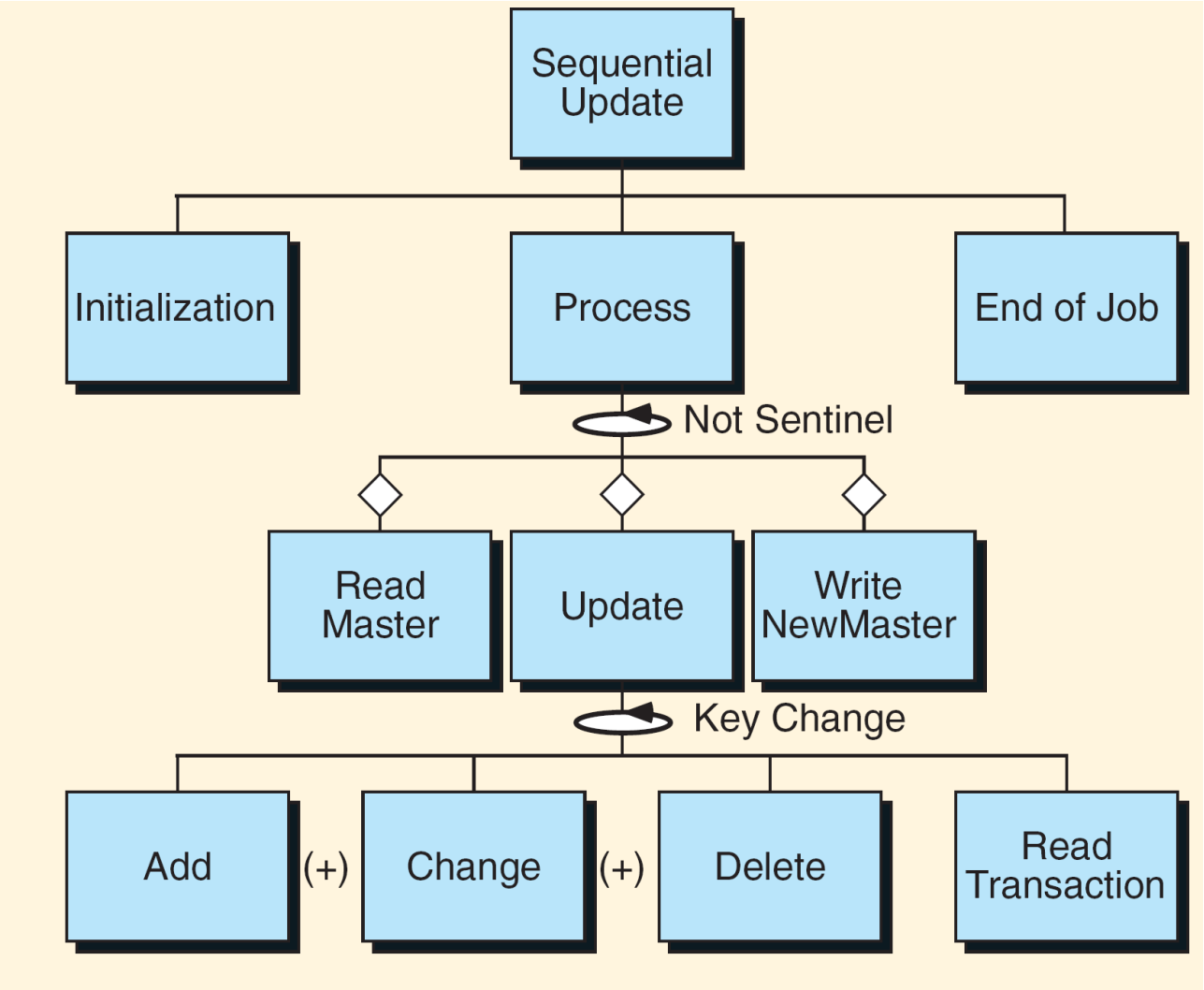


FIGURE 13-19 Update Structure Chart

ALGORITHM 13-2 Pseudocode for File Update

```
Algorithm Sequential Update
 1  read first record from transaction file
 2  read first record from old master file
 3  select next entity to be processed
 4  loop current entity not sentinel
    1   if current entity equals old master entity
        1   copy old master to new master work area
        2   read old master file
    2   end if
    3   if current entity equals transaction entity
        1   update new master work area
    4   end if
    5   if current entity equals new master entity
        1   write new master file
    6   end if
    7   select next entity to be processed
 5  end loop
End Sequential Update
```