Solutions to Chapter 13

Review Questions

1. a. True

3. a. True

5. b. False

7. a. Because they are more flexible, binary files are more portable.

9. a. The file is placed in an error state regardless of the file mode.

11. a. *fwrite*

13. d. *fseek*

15. b. Sequential files are often updated in an online environment.

Exercises

17.

File opened in write mode is being read.
`char* m = "wb"` should be `char* m = "rb"`

19.

- **a.** No error
- **b.** Error: *ftell* needs only one parameter, stream pointer.
- **c.** Error: `sp` must be the first parameter and the seek code is the last.
- **d.** No error
- **e.** No error

21. 'C' is printed. When 'C' is read, the file marker is advanced to 'D.' See Figure 13-1.

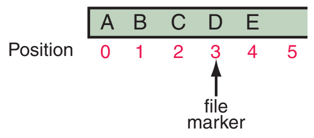

Figure 13-1 Solution for Exercise 21.

23. 'D' is printed. When 'D' is read, the file marker is advanced to 'E.' See Figure 13-2.

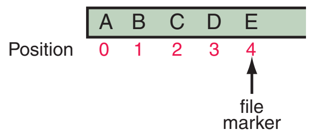

Figure 13-2 Solution for Exercise 23.

25. When the file is read, the file marker is at location 8. Because there are no data there, the read fails and actual results are unpredictable. Also, because the read fails, the file marker is not advanced. See Figure 13-3.

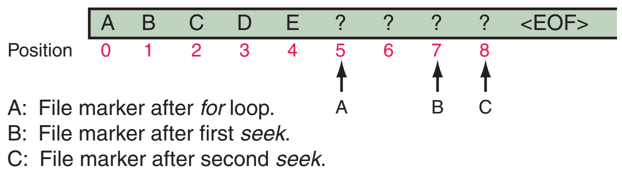
A B C D E ? ? ? ? <EOF>
Position 0 1 2 3 4 5 6 7 8
A
B
C
A: File marker after *for* loop.
B: File marker after first *seek*.
C: File marker after second *seek*.

Figure 13-3 Solution for Exercise 25.

27. 3 is printed. See Figure 13-4.

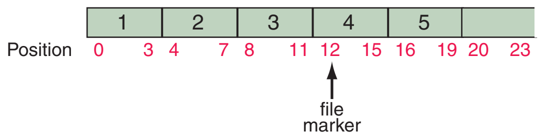

Figure 13-4 Solution for Exercise 27.

29. In the call to *fseek*, it does not move in an increment of `sizeof(int)`, so garbage is printed because the file marker is located at the middle of integer data (second byte of 4). See Figure 13-5.

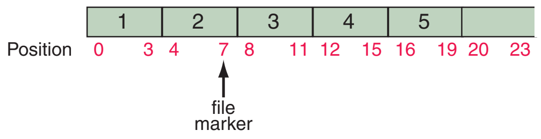

Figure 13-5 Solution for Exercise 29.

Problems

31. See Program 13-1.

Program 13-1 Solution to Problem 31

```
/* ==================== cpyFile =====================
   This function copies the contents of a binary file
   of integers to a second file.
      Pre    fp1 is file pointer to open read file
             fp2 is file pointer to open write file
      Post   file copied
      Return 1 is successful or zero if error
*/
int cpyFile (FILE* fp1, FILE* fp2)
{
// Local Declarations
   int data;

// Statements
   fseek (fp1, 0, SEEK_END);
```

Program 13-1 Solution to Problem 31 (continued)

```
   if (!ftell (fp1))
      {
       printf ("\n\acpyFile Error : file empty\n\n");
       return 0;
      } // if open error
   if (fseek (fp1, 0, SEEK_SET))
      return 0;
   if (fseek (fp2, 0, SEEK_SET))
      return 0;

   while (fread  (&data, sizeof (int), 1, fp1))
          fwrite (&data, sizeof (int), 1, fp2);
   return 1;
}  // cpyFile
```

33. See Program 13-2.

Program 13-2 Solution to Problem 33

```
/* ==================== fileCmp =====================
   This function compares two files
      Pre    two pointers to opened binary files
      Post   result of comparison returned
                --if equal, returns true
                --if not equal, returns false
*/
bool fileCmp (FILE* binfile1, FILE* binfile2)
{
// Local Declarations
   char data1;
   char data2;
   int  check  = 0;
   bool retval = false;

// Statements
   fseek (binfile1, 0, SEEK_END);
   fseek (binfile2, 0, SEEK_END);

   if (ftell (binfile1)  !=  ftell (binfile2))
      retval = false;
   else
      {
       fseek (binfile1, 0, SEEK_SET);
       fseek (binfile2, 0, SEEK_SET);

       while (fread (&data1, sizeof (char), 1, binfile1)
          &&  fread (&data2, sizeof (char), 1, binfile2)
          &&  !check)
              check = data1 - data2;

       if (check)
          retval = true;
      } // else files are equal length
   return retval;
}  // fileCmp
```

35. See Program 13-3.

Program 13-3 Solution to Problem 35

```
/* =================== printLast ===================
   This function prints the last integer in a binary
   file of integers.
      Pre    pointer to binary file opened for reading
```

Program 13-3 Solution to Problem 35 (continued)

```
      Post  last integer printed
*/
void printLast (FILE* binfile)
{
// Local Declarations
   int data;

// Statements
   fseek (binfile, 0, SEEK_END);
   if (ftell (binfile)  ==  0L)
      printf ("\n\aThe file is empty\n\n");
   else
      {
       fseek (binfile, -1 * sizeof(int), SEEK_CUR);
       fread (&data, sizeof(int), 1, binfile);
       printf ("\nThe last integer is %d\n", data);
      } // else file !empty
   return;
}  // printLast
```

37. See Program 13-4.

Program 13-4 Solution to Problem 37

```
/* =================== appendFile ===================
   This function appends one binary file to another.
      Pre   binfile1 is binary write file
            binfile2 is binary read file
      Post  binfile2 appended to binfile1
*/
void appendFile (FILE* binfile1, FILE* binfile2)
{
// Local Declarations
   STR rec;

// Statements
   fseek (binfile1, 0, SEEK_END);
   fseek (binfile2, 0, SEEK_SET);

   while (fread  (&rec, sizeof (STR), 1, binfile2))
          fwrite (&rec, sizeof (STR), 1, binfile1);
   return;
}  // apendFile
```

39. See Program 13-5.

Program 13-5 Solution to Problem 39

```
/* ===================== allocAry ====================
   This function reads items from a binary file and
   copies them to a dynamically allocated array.
      Pre   a pointer to a binary file open for reading
      Post  array loaded and returned – NULL if error
*/
STR* allocAry (FILE* binfile)
{
// Local Declarations
   long int count;
   int      i = 0;
   STR      rec;
   STR*     ary;

// Statements
   fseek (binfile, 0, SEEK_END);
```

Program 13-5 Solution to Problem 39 (continued)

```
   count = ( ftell (binfile) / sizeof (STR) );
   rewind (binfile);

   ary = (STR*) calloc ((int)count, sizeof (STR));

   if (ary)
      {
       while (fread (&rec, sizeof (STR), 1, binfile))
             {
              ary[i] = rec;
              i++;
             } // while
      } // if
   return ary;
}  // allocAry
```