16 Advanced I/O Concepts

REVIEW QUESTIONS

1. A file can be read only if it is opened in the input state.

   a. true

3. Using *seekg* or *seekp* to position a file beyond the current end of file places the file in an error state.

   b. false

5. The ______ results when a failure occurs during an open or during either a read or write operation.

   a. Error state

7. The ______ function may be used to position a file at the beginning for writing.

   e. seekp

9. Which of the following statements about sequential file updating is false?

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

EXERCISES

11. The file opened is an output file but then an attempt is made to read a record from the file.

13.

   a. The `tellp()` function passes the file object as a parameter instead of being called as a member function of the file object.

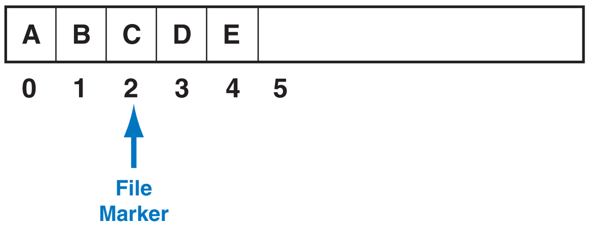
   b. The `tellp()` function does not accept any parameters.

   c. The `seekp()` function passes the file object as a parameter instead of being called as a member function of the file object. Also, the offset and wherefrom parameters are reversed and the wherefrom parameter is not resolved to the ios scope (`ios::0`).

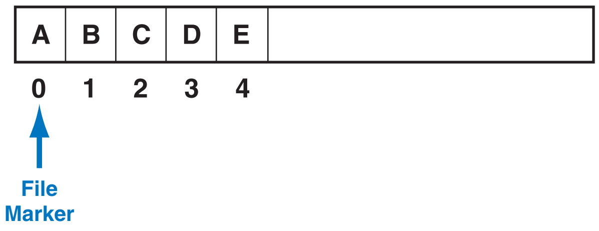
   d. The wherefrom parameter is not resolved to the ios scope (`ios::`beg). It is also recommended that the offset be cast to a long.

e. The call to `seekp()` passes the file object as a parameter instead of being called as a member function of the file object. Also, the wherefrom parameter is not resolved to the ios scope (`ios::end`). It is also recommended that the offset be cast to a long.

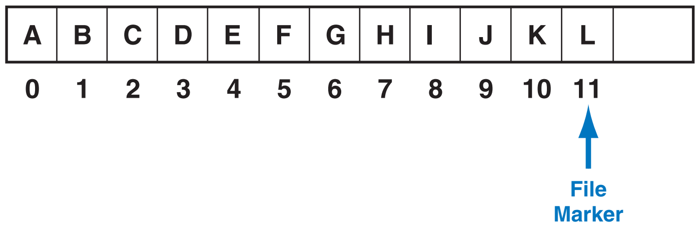
15. C (After the read, the file marker moves to D.)



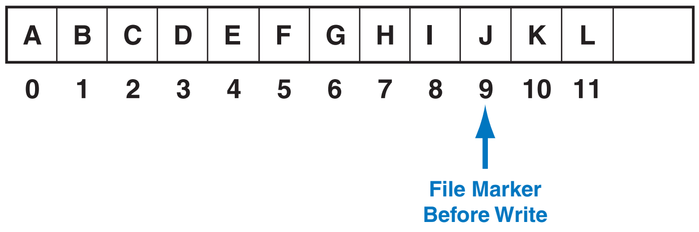
17. A (After the read, the file marker moves to B.)



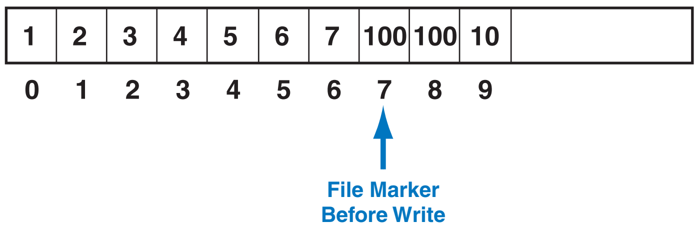
19. L L (on separate lines.) After the first read, the file marker is at the end of file. It is repositioned to the last item, and `L` is read a second time.



21. ABCDEFGHI?KL



23. 100 100 10 (on separate lines) File marker is shown before first read.



PROBLEMS

25.

```
/* =================== cpy_file =====================
   Copy contents of binary file to second file.
      Pre   Two open file objects passed by reference
      Post  File copied
            -or- returns boolean error (false)
*/
bool cpy_file (ifstream& binfile1, ofstream& binfile2)
{
   binfile1.seekg (0L, ios::beg);
   binfile2.seekp (0L, ios::beg);

   int data;
   while (binfile1.read  ((char*)&data, sizeof(int)))
      if (!binfile2.write((char*)&data, sizeof(int)))
         {
          cerr << "\n\a**Error writing to file\n\n";
          return false;
         } // if
   return true;
}  // end of cpy_file
```

27.

```
/* ==================== file_cmp ====================
   This function compares two files
      Pre  Binary file objects passed by reference
      Post The result of comparison returned
*/
bool file_cmp (ifstream& binfile1, ifstream& binfile2)
{
   binfile1.seekg (0L, ios::beg);
   binfile2.seekg (0L, ios::beg);

   char data1;
   char data2;
   while (binfile1.read (&data1, sizeof (char)))

      {
       binfile2.read (&data2, sizeof (char));
       if (data1 !=  data2 || binfile2.eof())
          //File contents are not equal
          return false;
      } // while

   // if equal files, next read should be eof
   binfile2.read (&data2, sizeof (char));
   if (binfile2.good())
      // extra data on binfile2
      return false;

   if (binfile1.eof() && binfile2.eof())
        return true;
   else
        //Files are of different lengths
        return false;
}  // end of file_cmp
```

29.

```
/* =================== print_last ===================
   Print the last integer in a binary file of integers.
      Pre   binary file object passed by reference
      Post  last integer printed
*/
void print_last (ifstream & binfile)
{

   binfile.seekg (0L, ios::end);

   int data;
   if (binfile.tellg() != 0)
      {
       binfile.seekg (-(sizeof (int)), ios::cur);
       binfile.read ((char*)&data, sizeof (int));
       cout << "\nLast integer in file is :  "
            << data << endl;
      } // if
   else
      cout << "\n\aThe file is empty\n";
   return;
}  // end of print_last
```

31.

```
/* ================== apend_file ====================
   Append one binary file at the end of the other.
      Pre   binary file objects passed by reference
            (opened for reading and writing)
      Post  file 2 appended to file 1
*/
void apend_file (fstream& binfile1, fstream& binfile2)
{
   STR rec;
   binfile1.seekp (0L, ios::end);
   binfile2.seekg (0L, ios::beg);

   while (binfile2.read ((char*)&rec, sizeof (STR)))
      binfile1.write ((char*)&rec, sizeof (STR));
   return;
}  // end of apend_file
```

33. Assumes record structure from Problem 30.

```
/* ==================== alloc_ary ===================
   This function reads items from a binary file and
   copies them to a dynamically allocated array.
      Pre   binary file object passed by reference
            pointer to array to be allocated passed
```

```
       Post  array loaded
             returns number of elements in array
*/
int alloc_ary (ifstream& binfile, STR** p_ary)
{
   STR  rec;
   STR* ary;
   int cnt = 8;
   binfile.seekg (0L, ios::beg);
   while ( binfile.read ((char*)&rec, sizeof (STR)))
      cnt++;

   ary  =  new STR [ cnt ];
   if (!ary)
      {
       cout << "\aMemory error in alloc_ary\a\n";
        exit (200);
      } // if

   int i  =  0;
   binfile.seekg (0L, ios::beg);
   while (binfile.read ((char*)&rec, sizeof (STR)))
      {
       *(ary + i)  =  rec;
       i++;
      } // while
   *p_ary = ary;
   return cnt;
}  // end of alloc_ary
```