C++ PRIMER PLUS, 5th EDITION
PROGRAMMING EXERCISES
CHAPTER 17

1. Write a program that counts the number of characters up to the first $ in input and that leaves the $ in the input stream.

2. Wite a program that copies your keyboard input (up to the simulated end-of-file) to a file named on the command line.

3. Write a program that copies one file to another. Have the program take the filenames from the command line. Have the program report if it cannot open a file.

4. Write a program that opens two text files for input and one for output. The program should concatenate the corresponding lines of the input files, use a space as a separator, and write the results to the output file. If one file is shorter than the other, the remaining lines in the longer file should also be copied to the output file. For example, suppose the first input file has these contents.

```
eggs kites donuts
balloons hammers
stones
```

And suppose the second input file has these contents:

```
zero lassitude
finance drama
```

The resulting file woul dhave these contents:

```
eggs kites donuts zero lassitude
balloons hammers finance drama
stones
```

5. Mat and Pat want to invite their friends to a party, much as they did in Programming Exercise 8 in Chapter 16, except now they want a program that uses files. They have asked you to write a program that does the following:

- Reads a list of Mat's friends' names from a text file called `mat.dat`, which lists one friend per line. The names are stored in a container and then displayed in sorted order.
- Reads a list of Pat's friends names from a text file called `pat.dat`, which lists one friend per line. The names are stored in a container and then displayed in sorted order.
- Merges the two lists, eliminating duplicates, and stores the result in the file `matnpat.dat`, one friend per line.

6. Consider the class definitions of Programming Exercise 5 in Chapter 14. If you haven't yet done that exercise, do so now. Then do the following:

Write a program that uses standard C++ I/O and file I/O in conjunction with data of types `employee`, `manager`, `fink`, and `highfink`, as defined in Programming Exercise 5 in Chapter 14. The program should be along the general lines of Listing 17.17 in that it should let you add new data to a file. The first time through, the program should solicit data from the user, show all the entries, and save the information in a file. On subsequent uses, the program should first read and display the file data, let the user add data, and show all the data. One difference is that data should be handled by an array of pointers to type `abstr_emp`. That way, a pointer can point to an `abstr_emp` object or to objects of any of the three derived types. Keep the array small to facilitate checking the program; for example, you might limit the array to 10 elements.

```
const int MAX = 10;         // no more than 10 objects
...abstr_emp * pc[MAX];
```

For keyboard entry, the program should use a menu to offer the user the choice of which type of ojbect to create. The menu should use a swtich to use `new` to create an object of the desired type and to assign the object's address to a pointer in the `pc` array. Then that object can use the virtual `setall()` function to elicit the appropriate data from the user.

```
pc[i]->setall();   // invokes function corresponding to type of
object
```

To save the data to a file, devise a virtual `writeall()` function for that purpose.

```
for(i = 0; i < index; i++)
    pc[i]->writeall(fout);
    // fout ofstream connected to output file
```

The tricky part is recovering the data from the file. The problem is, how can the program know wheter the next item to be recovered is type `employee` object, a `manager` object, `fink` object, or a `highfink` type? One approach is, when writing the data for an object file, precede the data with an integer that indicates the type of object to follow. Then, on file input, the program can read the inteer and then use `switch` to create the appropriate object to receive the data:

```
enum classkind{Employee, Manager, Fink, Highfink}; // in class header
...
int classtype;
while((fin >> classtype).get(ch)){ // newline separates int from data
    switch(classtype {
        case Employee :  pc[i] = new employee;
                         break;
```

Then you can use the pointer to invoke the virtual `getall()` function to read the information:

```
pc[i++]->getall();
```

7. Here is part of a program that reads keyboard input into a vector of `string` objects, stores the string contents (not the objects) in a file, and then copies the file contents back into a vector of `string` objects:

```
int main()
{
    using namespace std;
    vector<string> vostr;
    string temp;

// acquire strings
    cout << "Enter strings (empty line to quit):\n";
    while (getline(cin,temp) && temp[0] != '\0')
        vostr.push_back(temp);
    cout << "Here is your input.\n";
    for_each(vostr.begin(), vostr.end(), ShowStr);

// store in a file
    ofstream fout("strings.dat", ios_base::out | ios_base::binary);
    for_each(vostr.begin(), vostr.end(), Store(fout));
    fout.close();

// recover file contents
    vector<string> vistr;
    ifstream fin("strings.dat", ios_base::in | ios_base::binary);
    if (!fin.is_open())
    {
        cerr << "Could not open file for input.\n";
        exit(EXIT_FAILURE);
    }
    GetStrs(fin, vistr);
    cout << "\nHere are the strings read from the file:  \n";
    for_each(vistr.begin(), vistr.end(), ShowStr);

    return 0;
}
```

7. (continued)
Note that the file is opened in binary format and that the intention is that I/O be accomplished with `read()` and `write()`. Quite a bit remains to be done:

- Write a `void ShowStr(const string &)` function that displays a `string` object followed by a newline character.

- Write a `Store` functor that writes string information a file. The `Store` constructor should specify an `ifstream` object, and the overloaded `operator()(const string &)` should indicate the string to write. A workable plan is to first write the string's size to the file and then write the string's contents. For example, if `len` holds the string size, you could use this:

```
os.write((char *)&len, sizeof(std::size_t));     // store length
os.write(s.data(), len);                         // store characters
```

  The `data()` member returns a pointer to an array that holds the characters in the string. It's similar to the `c_str()` member except that the latter appends a null character.

- Write a `GetStr()` function that recovers information from the file. It can use `read()` to obtain the size of a string and then use a loop to read that many characters from the file, appending them to an initially empty temporary string. Because a string's data is private, you have to use a class method to get data into the string rather than read directly into it.