فصل
هفدهم

پردازش فایل

اهداف

- **ایجاد، خواندن، نوشتن و به روز کردن فایل‌ها.**
- **پردازش فایل‌های ترتیبی.**
- **پردازش فایل‌ها با دسترسی تصادفی.**
- **استفاده از عملیات I/O قالب‌بندی نشده با کارایی بالا.**
- **ایجاد یک برنامه تراکنشی با استفاده از پردازش فایل با دسترسی تصادفی.**

رئوس مطالب

۱۷-۱ مقدمه

متغیرها و آرایه‌ها، فقط قادر به نگهداری موقت داده‌ها هستند. زمانیکه یک متغیر محلی به خارج از قلمرو خود می‌رود یا هنگامی که برنامه خاتمه می‌یابد، داده‌ها از بین می‌روند. در مقابل، از فایل‌ها برای نگهداری طولانی مدت حجم زیادی از اطلاعات، حتی در زمانیکه برنامه ایجاد کننده آنها خاتمه می‌پذیرد، استفاده می‌شود. کامپیوترها، فایل‌ها را بر روی دستگاههای ذخیره‌سازی ثانویه، همانند دیسک‌های مغناطیسی، دیسک‌های نوری و نوارهای مغناطیسی ذخیره می‌کنند. در این فصل، به بررسی نحوه ایجاد، به روز کردن و پردازش داده‌های فایل‌ها در برنامه‌های C++ می‌پردازیم. در مورد هر دو نوع نحوهٔ دسترسی به فایل یعنی ترتیبی و تصادفی صحبت خواهیم کرد. یکی از قابلیت‌های بسیار مهم در هر زبان برنامه‌نویسی، پردازش فایل است چرا که با وجود این توانایی، می‌توان برنامه‌های تجاری ایجاد کرد. چنین برنامه‌های می‌توانند حجم زیادی از اطلاعات را پردازش کنند.

۲-۱۷ سلسله مراتب داده

عاقبت تمام ایتم‌های داده توسط کامپیوتر به ترکیب‌هایی از صفرها و یک‌ها تبدیل می‌شوند. دلیل این امر ساده و اقتصادی بودن ساخت قطعات الکترونیکی است که براساس دو وضعیت پایدار یعنی 0 و 1 کار می‌کنند.

کوچکترین ایتم داده که کامپیوترها از آن پشتیبانی می‌کنند، *بیت* نامیده می‌شود (کوتاه شده عبارت "*binary digit*" یا رقم باینری است، یک رقم می‌تواند یکی از دو مقدار صفر یا یک باشد). هر ایتم داده یا بیت، می‌تواند بعنوان مقدار صفر یا یک فرض گردد. مدارات کامپیوتر اعمال ساده‌ای بر روی بیت‌ها انجام می‌دهند، اعمالی مانند بررسی مقدار یک بیت، تنظیم مقدار بیت و معکوس کردن بیت (از **1** به **0** یا از **0** به **1**).

اصولاً برنامه‌نویس ترجیح می‌دهد که با این سطح از داده‌ها کار نکند و بجای آن با داده‌هایی مانند ارقام دهدهی (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) حروف (از A تا Z و a تا z)، نمادهای ویژه (همانند $، @، %، &، *، (،)، –، +، "، :، ?، /) کار کند. به ارقام، حروف و نمادهای ویژه *کاراکتر* گفته می‌شود. از مجموعه این کاراکترها برای نوشتن برنامه‌ها استفاده می‌شود و مجموعه این کاراکترها در یک کامپیوتر *مجموعه کاراکترهای* آن کامپیوتر نامیده می‌شوند. بدلیل اینکه کامپیوترها می‌توانند فقط با **1**ها و **0**ها کار کنند، هر کاراکتری در مجموعه کاراکتری یک کامپیوتر با استفاده از تعدادی 1 و 0 ارائه می‌شود که ترکیب آنها با یکدیگر یک یا چند *بایت* را تشکیل می‌دهد. بایت‌ها از ترکیب (مجموع) هشت بیت ساخته می‌شوند. برنامه‌نویس برنامه و داده‌ها را با استفاده از کاراکترها ایجاد می‌کند. کاراکترها ترکیبی از بیت‌ها هستند و ترکیبی از کاراکترها یا بایت‌ها، فیلدها را بوجود می‌آورند. یک فیلد گروهی از کاراکترهاست که معنی و مفهوم خاصی ایجاد می‌کند. برای مثال، یک فیلد شامل حروف بزرگ و کوچک می‌تواند در نشان دادن نام شخصی مورد استفاده قرار گیرد.

عناصر داده توسط کامپیوتر از طریق سلسله مراتب داده (شکل ۱-۱۷)، که این عناصر را به ساختار بزرگ و پیچیده‌ای که آنرا از بیت‌ها ساخته‌ایم، به کاراکترها تبدیل می‌کند.

بطور کلی، یک *رکورد* ترکیبی از چند فیلد است. برای مثال، در یک سیستم پرداخت حقوق، یک رکورد برای کارمند ممکن است دارای فیلدهای زیر باشد:

۱- شمارهٔ شناسایی کارمند

۲- نام

۳- آدرس

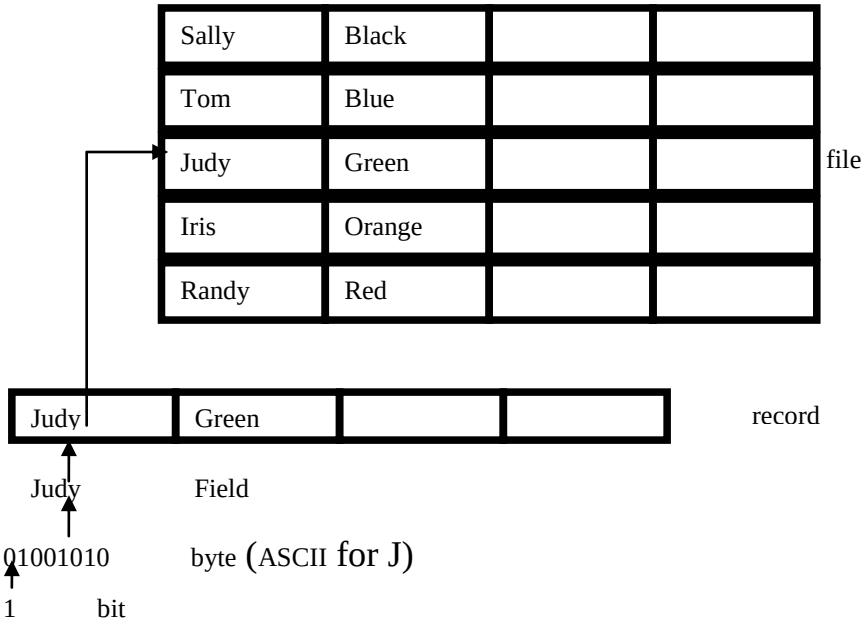
۴- نرخ دستمزد ساعتی

۵- تعداد مرخصی

۶- سال استخدام

۷- مقدار مالیات بر درآمد

بنابراین یک رکورد گروهی از فیلدهای مرتبط باهم است. در مثالی که آورده شدهٔ هر کدام یک از این فیلدها متعلق به یک کارمند است. البته یک شرکت ممکن است تعداد زیادی کارمند داشته باشد و در اینحالت هر کارمند رکورد مخصوص خود را خواهد داشت. یک فایل گروهی از رکوردهای مرتبط با یکدیگر است. یک فایل دستمزد بطور عادی شامل یک رکورد برای هر کارمند می‌باشد. برای یک شرکت کوچک فایل دستمزد ممکن است فقط 22 رکورد داشته باشد، در حالیکه یک شرکت بزرگ ممکن است بیشتر از صد هزار رکورد در فایل دستمزد خود داشته باشد.



شکل ۱-۱۷ | سلسله مراتب داده.

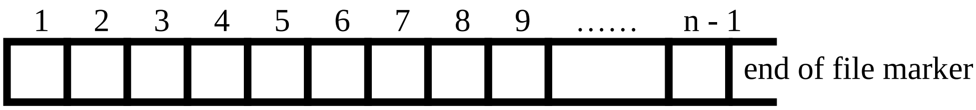
برای آسانتر کردن دستیابی به رکوردهای موجود در یک فایل یکی از فیلدهای هر رکورد بعنوان *کلید* رکورد انتخاب می‌شود. کلید رکورد، شناسه یک رکورد است که متعلق به یک شخص یا موجودیت می‌باشد و آن رکورد را از تمام رکوردهای دیگر متمایز می‌کند. در رکورد دستمزد، شماره شناسایی کارمند می‌تواند بعنوان کلید رکورد بکار گرفته شود. روش‌های متعددی برای سازماندهی رکوردها در

یک فایل وجود دارد. یکی از عمومی‌ترین نوع سازماندهی، *فایل ترتیبی* نامیده می‌شود. در این نوع از سازماندهی رکوردها به ترتیب فیلد کلید رکورد ذخیره می‌شوند. در فایل دستمزد، رکوردها به ترتیب شماره شناسایی کارمند در فایل قرار می‌گیرند. اولین رکورد در این فایل کوچکترین شماره‌شناسایی را خواهد داشت و به همین ترتیب رکوردهای بعدی دارای شماره‌شناسایی بالاتری خواهند بود.

بیشتر سازمان‌های تجاری از فایل‌های متفاوتی برای ذخیره داده‌ها استفاده می‌کنند. برای مثال یک کمپانی ممکن است دارای فایل‌های دستمزد، حقوق، درآمد، مشتری و غیره باشد. ارتباط چندین فایل با یکدیگر *پایگاه داده* (*Database*) نامیده می‌شود. به مجموعه برنامه‌های طراحی، ایجاد و مدیریت پایگاه داده، *سیستم مدیریت پایگاه داده* (DBMS[1]) اطلاق می‌شود.

۱۷-۳ فایل‌ها و استریم‌ها

نگاه ++C به هر فایل بفرم یک *استریم* (*stream*) متوالی از بایت‌ها است (شکل ۱۷-۲). انتهای هر فایل با یک نماد پایان فایل یا به تعداد بایت‌های مشخص شده که در سیستم مدیریت نگهداری ساختار داده به ثبت رسیده، تعیین می‌شود. هنگامی که یک فایل باز می‌شود، ++C یک شیی ایجاد کرده و سپس آنرا به یک استریم مرتبط می‌کند. در فصل ۱۵ مشاهده کردید که شی‌های cin, cout, cerr و clog به هنگام استفاده از <iostram> ایجاد می‌شوند. این شیی‌ها ارتباط مابین یک برنامه و یک فایل مشخص یا دستگاه را تسهیل می‌بخشند. برای مثال شی **cin** به برنامه امکان می‌دهد تا داده را از طریق صفحه کلید بعنوان ورودی بپذیرد. شی **cout** به برنامه امکان می‌دهد تا داده به روی صفحه نمایش منتقل شود (خروجی). شی‌های **cerr** و **clog** به برنامه اجازه می‌دهند تا پیغام خطا را بر روی صفحه نمایش به نمایش در آورد.



شکل ۱۷-۲ | نگاه ++C به یک فایل n بایت‌ی.

برای پردازش فایل در ++C، لازم است تا فایل‌های سرآیند **<iostream>** و **<fstream>** بکار گرفته شوند. سرآیند **<fstream>** شامل تعاریفی برای الگوهای استریم کلاس **basic_ifstream** (برای فایل ورودی)، **basic_ofstream** (برای فایل خروجی) و **basic_fstream** (برای فایل ورودی و خروجی) است. هر الگوی کلاس دارای یک الگوی تخصصی از پیش تعریف شده است که **char I/O** را امکان‌پذیر می‌سازد. علاوه بر این، کتابخانه **fstream** مجموعه‌ای از *typedef‌ها* تدارک دیده است که برای این الگوهای تخصصی اسامی مستعار تهیه می‌کنند. برای مثال **typedef ifstream** نشاندهنده یک

۱- Database Management System

basic_ifstream تخصصی شده است که ورودی **char** از یک فایل را فراهم می‌آورد. به همین ترتیب **typedef ofstream** نشاندهنده یک **basic_ofstream** تخصصی است که خروجی **char** به فایل‌ها را فراهم می‌آورد.

فایل‌ها با ایجاد شی‌ها از این الگوهای تخصصی استریم باز می‌شوند. این الگوها از الگوهای کلاس **basic_istream**، **basic_ostream** و **basic_iostream** مشتق می‌شوند. از اینرو، تمام توابع عضو، عملگرها و دستکاری کنندهایی که متعلق به این الگوها هستند نیز می‌تواند در استریم‌های فایل بکار گرفته شوند. شکل ۳-۱۷ بطور خلاصه رابطه توارث کلاس I/O را که تا بدین جا مطرح کرده‌ایم را نشان می‌دهد.

شکل ۳-۱۷ | بخشی از استریم سلسله مراتب الگوی I/O.

۱۷-٤ ایجاد فایل ترتیبی

C++ ساختاری بر روی فایل تحمیل نمی‌کند. از اینرو، مفاهیمی همانند "رکورد" در فایل‌های C++ وجود ندارد. به این معنی که، برنامه‌نویس بایستی ساختار فایل را به نحوی تدارک ببیند که نیاز برنامه را جوابگو باشد. در مثال بعدی، از کاراکترهای متنی و ویژه، برای سازماندهی مفهوم خاصی که برای "رکورد" قائل هستیم استفاده خواهیم کرد.

برنامه شکل ۴-۱۷ یک فایل ترتیبی ایجاد می‌کند که می‌تواند در یک سیستم دریافت کننده حساب به منظور مدیریت پول بکار گرفته شود. برای هر مشتری، برنامه یک شماره حساب، نام، نام‌خانوادگی و موجودی را فراهم می‌آورد. اطلاعات دریافتی برای هر مشتری، یک رکورد برای آن مشتری تشکیل می‌دهند. در این برنامه، شماره حساب، نشاندهنده کلید رکورد است. ایجاد و دستکاری کردن فایل‌ها براساس ترتیب شماره حساب صورت می‌گیرد. برنامه بر این فرض کار می‌کند که کاربر، براساس ترتیب شماره حساب رکوردها را وارد می‌کند. با این همه، یک سیستم کارآمد باید دارای قابلیت مرتب‌سازی نیز باشد. کاربر می‌تواند با هر ترتیبی، رکوردها را وارد کرده و سپس رکوردها مرتب شده و بصورت منظم در فایل نوشته شوند.

```
1  // Fig. 17.4: Fig17_04.cpp
2  // Create a sequential file.
3  #include <iostream>
4  using std::cerr;
5  using std::cin;
6  using std::cout;
7  using std::endl;
8  using std::ios;
9
10 #include <fstream> // file stream
11 using std::ofstream; // output file stream
12
13 #include <cstdlib>
14 using std::exit; // exit function prototype
15
```

```
16 int main()
17 {
18    // ofstream constructor opens file
19    ofstream outClientFile( "clients.dat", ios::out );
20
21    // exit program if unable to create file
22    if ( !outClientFile ) // overloaded ! operator
23    {
24       cerr << "File could not be opened" << endl;
25       exit( 1 );
26    } // end if
27
28    cout << "Enter the account, name, and balance." << endl
29       << "Enter end-of-file to end input.\n? ";
30
31    int account;
32    char name[ 30 ];
33    double balance;
34
35    // read account, name and balance from cin, then place in file
36    while ( cin >> account >> name >> balance )
37    {
38       outClientFile <<account <<' ' <<name <<' '<< balance << endl;
39       cout << "? ";
40    } // end while
41
42    return 0; // ofstream destructor closes file
43 } // end main
```

```
Enter the account, name, and balance.
Enter end-of-file to end input.
? 100 Jones 24.98
? 200 Doe 345.67
? 300 White 0.00
? 400 Stone -42.16
? 500 Rich 224.62
? ^Z
```

شکل ٤-١٧ | ایجاد فایل ترتیبی.

اجازه دهید تا به بررسی این برنامه بپردازیم. همانطوری که قبلاً گفته شد، فایلها با ایجاد شی‌ها **ifstream**، **ofstream** یا **fstream** باز می‌شوند. در شکل ۴-۱۷، فایل برای خروجی باز شده است، از اینرو یک شی **ofstream** ایجاد شده است. دو آرگومان به سازنده شی ارسال شده است، نام فایل و مد باز کردن فایل (خط 19). برای یک شی **ofsteam**، مد باز کردن فایل می‌تواند **ios::out** برای خارج کردن داده به یک فایل یا **ios::app** برای الحاق داده به انتهای فایل باشد (بدون اینکه تغییری در داده‌های حاضر در فایل اعمال کند). باز کردن فایل‌های موجود در مد **iso::out** سبب بریده شدن فایل می‌شود، به این معنی که تمام داده‌های موجود در فایل از بین می‌روند. اگر فایل از قبل وجود نداشته باشد، پس **ofstream** فایل را با استفاده از نام فایل ایجاد می‌کند.

خط 19 یک شی **ofstream** بنام **outClientFile** مرتبط با فایل **clients.dat** ایجاد می‌کند که برای خروجی باز شده است. آرگومان‌های **"clients.dat"** و **ios::out** به سازنده **ofstream** ارسال می‌شوند که فایل را باز کند. اینکار یک "خط ارتباطی" با فایل بنا می‌کند.

بطور پیش‌فرض شی‌های **ofstream** برای خروجی باز می‌شوند، از اینرو خط 19 می‌توانست عبارت زیر را به اجرا در آورد

```
ofstream outClientFile( "clients.dat" );
```

تا **clients.dat** را برای خروجی باز نماید. جدول ۵-۱۷ مدل‌های باز کردن فایل را لیست کرده است.

مد	توضیح
ios::app	تمام خروجی را به انتهای فایل الصاق می‌کند.
ios::ate	یک فایل برای خروجی باز کرده و به انتهای فایل حرکت می‌کند (معمولاً برای الصاق داده به فایل بکار گرفته می‌شود). داده می‌تواند در هر کجای فایل نوشته شود.
ios::in	فایل را برای ورودی باز می‌کند.
ios::out	فایل را برای خروجی باز می‌کند.
ios::trunc	اگر فایل حاوی اطلاعات باشد، آنها را از بین می‌برد (پیش‌فرض **ios:out** است).
ios:binary	فایل را برای باینری باز می‌کند (یعنی غیرمتنی) ورودی یا خروجی.

شکل ۵-۱۷ | مدهای باز کردن فایل.

شی **ofstream** می‌تواند بدون باز کردن یک فایل خاص، فایلی که می‌تواند بعداً به شی الصاق شود، ایجاد گردد. برای مثال، عبارت

```
ofstream outClienFile;
```

یک شی **ofstream** بنام **outClientFile** ایجاد می‌کند. تابع عضو **open** از **ofstream** یک فایل باز کرده و آنرا به یک شی **ofstream** موجود الصاق می‌کند، همانند:

```
outClientFile.open( "clients.dat", ios::out );
```

پس از ایجاد یک شی **ofstream** و اقدام به باز کردن آن، برنامه تست می‌کند که آیا عملیات باز کردن با موفقیت همراه بوده است یا خیر. عبارت **if** در خطوط 26-22 از عملگر تابع عضو سربارگذاری شده **operator!** استفاده کرده تا تعیین کند که آیا عملیات باز کردن با موفقیت همراه شده است یا خیر. اگر **failbit** یا **badbit** برای استریم در عملیات باز کردن تنظیم شده باشد، شرط **true** برگشت خواهد داد. برخی از خطاهای که ممکن است به هنگام باز کردن فایل رخ دادند عبارتند از عدم وجود فایل برای خواندن، اقدام به باز کردن فایل برای خواندن یا نوشتن بدون مجوز و باز کردن فایلی برای نوشتن زمانیکه بر روی دیسک فضای کافی در اختیار نیست.

اگر شرط دلالت بر عدم موفقیت در باز کردن فایل داشته باشد، خط 24 پیغام خطای "File could not be opened" را چاپ کرده و خط 25 برای خاتمه دادن به برنامه تابع **exit** را فراخوانی می‌کند. آرگومانی از **exit** به محیط برگشت داده می‌شود. آرگومان صفر دلالت بر خاتمه عادی برنامه دارد، هر مقدار دیگری دلالت بر این می‌کند که برنامه به علت خطا خاتمه یافته است. محیط فراخوانی (غالبا سیستم عامل) از مقدار برگشتی توسط **exit** برای واکنش مناسب در برابر خطا استفاد می‌کنند.

یکی دیگر از عملگرهای تابع عضو سربارگذاری شده *** operator void** است که استریم را تبدیل به یک اشاره‌گر می‌کند، از اینرو می‌تواند برای تست صفر (یعنی اشاره‌گر **null**) یا غیر صفر (یعنی هر مقدار دیگر اشاره‌گر) بکار گرفته شود. زمانیکه مقدار یک اشاره‌گر بعنوان یک شرط بکار گرفته می‌شود، ++C اشاره‌گر **null** را به مقدار بولی **false** و اشاره‌گر غیر **null** را به مقدار بولی **true** تبدیل می‌کند. اگر **failbit** یا **badbit** برای استریمی تنظیم شده باشد، صفر (**false**) برگشت داده خواهد شد. شرط موجود در عبارت **while** خطوط 40-36 تابع عضو *** operator void** را بر روی تابع **cin** بصورت ضمنی اعمال می‌کند. با رسیدن به انتهای فایل، شاخص مبادرت به تنظیم **failbit** برای **cin** می‌کند. تابع **operator void *** می‌تواند برای تست شی ورودی برای انتهای فایل بجای فراخوانی صریح تابع عضو **eof** بر روی شی ورودی بکار گرفته شود.

اگر خط 19 فایل را با موفقیت باز کند، برنامه شروع به پردازش داده می‌کند. خطوط 29-28 به کاربر اعلان می‌کنند تا فیلدهای مختلف را برای هر رکورد وارد کرده یا انتهای فایل را پس از وارد کردن داده‌ها مشخص سازد. جدول شکل ۶-۱۷ حاوی ترکیبات کلیدی برای وارد کردن انتهای فایل در سیستم‌های مختلف کامپیوتری است.

خط 36 هر مجموعه از داده‌ها را استخراج کرده و تعیین می‌کند که آیا انتهای فایل وارد شده است یا خیر. زمانیکه با انتهای فایل مواجه شود یا داده نامعتبری وارد شده باشد، *** operator void** اشاره‌گر **null** را برگشت می‌دهد (که به مقدار بولی **false** تبدیل می‌کند) و عبارت **while** خاتمه می‌یابد. کاربر با وارد کردن انتهای فایل به برنامه اطلاع می‌دهد که اطلاعات دیگری برای پردازش وجود ندارد. زمانیکه کاربر کلید انتهای فایل را وارد کرد، شاخص انتهای فایل تنظیم می‌شود. حلقه عبارت **while** تا تنظیم شاخص انتهای فایل ادامه می‌یابد.

سیستم کامپیوتری	ترکیب کلیدهای صفحه کلید
UNIX/Linux/Mac OS X	*<ctrl-d>*
Microsoft Windows	*<ctrl-z>*
VAX (VMS)	*<ctrl-z>*

شکل ٦-١٧ | کلیدهای ترکیبی نشاندهنده انتهای فایل.

خط 38 مجموعه‌ای از داده‌ها را به فایل **clients.dat** با استفاده از عملگر >> و شی **outClientFile** مرتبط با فایل در ابتدای برنامه، می‌نویسد. داده‌ها را می‌توان از فایل خواند (بخش ۵-۱۷). توجه کنید بدلیل اینکه فایل ایجاد شده در شکل ۴-۱۷ یک فایل متنی ساده است، می‌توان آنرا توسط هر برنامه ویرایشگر متنی مورد بازبینی قرار داد.

زمانیکه کاربر شاخص انتهای فایل را وارد می‌کند، **main** خاتمه می‌یابد. با اینکار نابود کننده شی **outClientFile** بصورت ضمنی فراخوانی می‌شود، که مبادرت به بستن فایل **client.dat** می‌کند. همچنین برنامه می‌تواند شی **ofstream** را با استفاده از تابع عضو **close** ببندد، همانند عبارت زیر،

```
outClientFile.close();
```

در اجرای نمونه برنامه شکل ۴-۱۷ کاربر اطلاعاتی برای پنج حساب وارد کرده و با فشردن کلیدهای *ctrl*-z نشان داده که ورود اطلاعات به پایان رسیده است. این پنجره نحوه ظاهر شدن رکوردهای داده در فایل را نشان نداده است. برای بازبینی اینکه برنامه فایل را با موفقیت ایجاد کرده است، بخش بعدی نحوه خواندن این فایل و چاپ محتویات آنرا نشان داده است.

۵-۱۷ خواندن داده از یک فایل ترتیبی

فایل‌ها ذخیره کننده داده‌ها هستند، از اینرو در زمان پردازش داده‌ها نیاز است تا این داده‌ها بازیابی شوند. در بخش قبلی با نحوه ایجاد یک فایل با دسترسی ترتیبی آشنا شدید. در این بخش، با نحوه خواندن ترتیبی داده‌ها از فایل آشنا می‌شوید.

برنامه شکل ۷-۱۷ رکوردها را از فایل **clients.dat** می‌خواند، که آنرا با استفاده ازبرنامه ۴-۱۷ ایجاد کرده‌ایم و محتویات رکوردهای آنرا به نمایش در می‌آورد. با ایجاد یک شی **ifstream** یک فایل برای ورودی باز می‌شود. سازنده **ifstream** می‌تواند نام فایل و مد باز شدن فایل را به عنوان آرگومان دریافت کند. خط 31 یک شی **ifstream** بنام **inClientFile** ایجاد کرده و آنرا با فایل **clients.dat** مرتبط می‌کند. آرگومان‌های موجود در درون پرانتزها به تابع سازنده **ifstream** ارسال می‌شوند که فایل را باز کرده و یک خط ارتباطی با فایل را بنا می‌کنند.

```
1  // Fig. 17.7: Fig17_07.cpp
2  // Reading and printing a sequential file.
3  #include <iostream>
4  using std::cerr;
5  using std::cout;
6  using std::endl;
7  using std::fixed;
8  using std::ios;
9  using std::left;
10 using std::right;
11 using std::showpoint;
12
13 #include <fstream> // file stream
14 using std::ifstream; // input file stream
15
16 #include <iomanip>
17 using std::setw;
18 using std::setprecision;
19
20 #include <string>
21 using std::string;
22
23 #include <cstdlib>
24 using std::exit; // exit function prototype
25
26 void outputLine( int, const string, double ); // prototype
27
```

```
28 int main()
29 {
30    // ifstream constructor opens the file
31    ifstream inClientFile( "clients.dat", ios::in );
32
33    // exit program if ifstream could not open file
34    if ( !inClientFile )
35    {
36       cerr << "File could not be opened" << endl;
37       exit( 1 );
38    } // end if
39
40    int account;
41    char name[ 30 ];
42    double balance;
43
44    cout << left << setw( 10 ) << "Account" << setw( 13 )
45       << "Name" << "Balance" << endl << fixed << showpoint;
46
47    // display each record in file
48    while ( inClientFile >> account >> name >> balance )
49       outputLine( account, name, balance );
50
51    return 0; // ifstream destructor closes the file
52 } // end main
53
54 // display single record from file
55 void outputLine( int account, const string name, double balance )
56 {
57    cout << left << setw( 10 ) << account << setw( 13 ) << name
58       << setw( 7 ) << setprecision( 2 ) << right << balance << endl;
59 } // end function outputLine
```

```
Account    Name          Balance
100        Jones           24.98
200        Doe            345.67
300        White            0.00
400        Stone          -42.16
500        Rich           224.62
```

شکل ۷-۱۷ | خواندن و چاپ از یک فایل ترتیبی.

شی‌های از کلاس **ifstream** بطور پیش فرض برای ورودی باز می‌شوند. می‌توانیم از عبارت زیر استفاده کنیم

```
ifstream inClientFile( "clients.dat" );
```

تا **clients.dat** را برای ورودی باز کند. همانند یک شی **ofstream**، یک شی **ifstream** می‌تواند بدون باز کردن یک فایل خاص ایجاد شود، چرا که فایل می‌تواند بعدها الصاق گردد.

برنامه از شرط **!inclientFile** برای تعیین اینکه آیا فایل قبل از مبادرت به بازیابی داده‌ها از آن با موفقیت باز شده است یا خیر، استفاده کرده است. خط 48 یک مجموعه از داده‌ها (منظور رکورد است) را از فایل می‌خواند. پس از اینکه خط قبلی یک بار اجرا شد، **account** دارای مقدار 100، **name** دارای مقدار "Jones" و **balance** دارای مقدار 24.98 خواهد بود. هر بار که خط 48 اجرا می‌شود، یک رکورد دیگر از فایل را به درون متغیرهای **account**، **name** و **balance** می‌خواند. خط 49 با استفاده از تابع **outputLine** در خطوط 59-55 رکوردها را به نمایش در می‌آورد، که از دستکاری کننده‌های پارامتری شده استریم برای قالب‌بندی نمایش داده‌ها استفاده کرده است. زمانیکه به انتهای فایل می‌رسد، بطور ضمنی

عملگر * **void** در شرط **while**، اشاره‌گر **null** برگشت می‌دهد (که به مقدار بولی **false** تبدیل می‌شود)، تابع نابود کننده **ifstream** فایل را بسته و برنامه خاتمه می‌پذیرد.

برای بازیابی ترتیبی داده‌ها از یک فایل، معمولاً برنامه‌ها کار خواندن داده‌ها را از ابتدای فایل شروع کرده و بطور پیوسته داده‌ها را می‌خوانند تا اینکه به داده مورد نظر دست یابند. امکان انجام چندین باره اینکار در فایل ترتیبی (از ابتدای فایل) وجود دارد. هر دو شی **istream** و **ostream** توابع عضوی برای موقعیت‌دهی *اشاره‌گر موقعیت فایل* در نظر گرفته‌اند. این توابع عضو عبارتند از: **seekg** از **istream** و تابع **seekp** از **ostream**. هر شی **istream** دارای یک "*get pointer*" است که دلالت بر تعداد بایت‌ها در فایل از مکانی می‌کند که ورودی بعدی صورت می‌گیرد و هر شی **ostream** دارای یک "*put pointer*" است که دلالت بر تعداد بایت‌ها در فایل از مکانی می‌کند که باید خروجی بعدی جای داده شود. عبارت

```
inClientFile.seekg(0);
```

اشاره‌گر موقعیت فایل را به ابتدای فایل (موقعیت صفر) متصل به **inClientFile** انتقال می‌دهد. معمولاً آرگومان **seekg** یک مقدار صحیح **long** است. آرگومان دوم می‌تواند برای نشان دادن جهت جستجو بکار گرفته شود. جهت جستجو می‌تواند **ios::beg** (حالت پیش‌فرض) برای موقعیت‌یابی نسبی از ابتدای یک استریم، **ios::cur** برای موقعیت‌یابی نسبی با موقعیت جاری در استریم یا **ios::end** برای موقعیت‌یابی نسبی با انتهای استریم باشد. اشاره‌گر موقعیت فایل یک مقدار صحیح است که تصریح کننده مکانی در فایل بصورت عددی از بایت‌ها از مکان شروع فایل است (به اینحالت *افست* (*offset*) از ابتدای فایل گفته می‌شود). برخی از مثال‌های موقعیت‌یابی مکان اشاره‌گر فایل در زیر آورده شده است:

```
//position to the nth byte of fileObject (assumes ios::beg)
fileObject.seekg( n );

//position n bytes forward in fileObject
fileObject.seekg(n, ios::cur);

//position n bytes back from end of fileObject
fileObject.seekg( n, ios::end);

//position at end of fileObject
fileobject.seekg(0, ios::end);
```

همین عملیات‌ها را می‌توان با استفاده تابع عضو **seekp** از **ostream** انجام داد. توابع عضو **tellg** و **tellp** برای باز گرداندن مکان جاری اشاره‌گرهای "get" و "put" تدارک دیده شده‌اند. عبارت زیر مقدار اشاره‌گر موقعیت فایل "get" را به متغیر **location** از نوع **long** تخصیص می‌دهد:

```
location = fileObject.tellg();
```

برنامه شکل ۷-۱۸ به یک برنامه مدیریت اعتبار امکان می‌دهد تا اطلاعات حساب را برای مشتریانی با مانده حساب صفر (یعنی مشتریانی که به شرکت بدهکار نیستند)، مانده حساب اعتباری (منفی) و مانده حساب بدهکار (مثبت) به نمایش در آورد. برنامه یک منو به نمایش در آورده و به مدیر اعتبارات اجازه می‌دهد

تا یکی از سه گزینه را برای بدست آوردن اطلاعات اعتباری وارد سازد. گزینه1 لیستی از حساب‌ها با مانده حساب صفر، گزینه2 لیستی از حساب‌ها با مانده حساب اعتباری، گزینه3 لیستی از حساب‌های مانده بدهکار تولید می‌کند. گزینه4 به اجرای برنامه خاتمه می‌دهد. با وارد کردن یک گزینه غیرمعتبر، برنامه به کاربر اعلان می‌کند تا گزینه معتبری را وارد سازد.

```
1   // Fig. 17.8: Fig17_08.cpp
2   // Credit inquiry program.
3   #include <iostream>
4   using std::cerr;
5   using std::cin;
6   using std::cout;
7   using std::endl;
8   using std::fixed;
9   using std::ios;
10  using std::left;
11  using std::right;
12  using std::showpoint;
13
14  #include <fstream>
15  using std::ifstream;
16
17  #include <iomanip>
18  using std::setw;
19  using std::setprecision;
20
21  #include <string>
22  using std::string;
23
24  #include <cstdlib>
25  using std::exit; // exit function prototype
26
27  enum RequestType {ZERO_BALANCE=1,CREDIT_BALANCE,DEBIT_BALANCE,END};
28  int getRequest();
29  bool shouldDisplay( int, double );
30  void outputLine( int, const string, double );
31
32  int main()
33  {
34     // ifstream constructor opens the file
35     ifstream inClientFile( "clients.dat", ios::in );
36
37     // exit program if ifstream could not open file
38     if ( !inClientFile )
39     {
40        cerr << "File could not be opened" << endl;
41        exit( 1 );
42     } // end if
43
44     int request;
45     int account;
46     char name[ 30 ];
47     double balance;
48
49     // get user's request (e.g., zero, credit or debit balance)
50     request = getRequest();
51
52     // process user's request
53     while ( request != END )
54     {
55        switch ( request )
56        {
57           case ZERO_BALANCE:
58              cout << "\nAccounts with zero balances:\n";
59              break;
60           case CREDIT_BALANCE:
```

```
61             cout << "\nAccounts with credit balances:\n";
62             break;
63          case DEBIT_BALANCE:
64             cout << "\nAccounts with debit balances:\n";
65             break;
66       } // end switch
67
68       // read account, name and balance from file
69       inClientFile >> account >> name >> balance;
70
71       // display file contents (until eof)
72       while ( !inClientFile.eof() )
73       {
74          // display record
75          if ( shouldDisplay( request, balance ) )
76             outputLine( account, name, balance );
77
78          // read account, name and balance from file
79          inClientFile >> account >> name >> balance;
80       } // end inner while
81
82       inClientFile.clear();    // reset eof for next input
83       inClientFile.seekg( 0 ); // reposition to beginning of file
84       request = getRequest();  // get additional request from user
85    } // end outer while
86
87    cout << "End of run." << endl;
88    return 0; // ifstream destructor closes the file
89 } // end main
90
91 // obtain request from user
92 int getRequest()
93 {
94    int request; // request from user
95
96    // display request options
97    cout << "\nEnter request" << endl
98       << " 1 - List accounts with zero balances" << endl
99       << " 2 - List accounts with credit balances" << endl
100      << " 3 - List accounts with debit balances" << endl
101      << " 4 - End of run" << fixed << showpoint;
102
103   do // input user request
104   {
105      cout << "\n? ";
106      cin >> request;
107   } while ( request < ZERO_BALANCE && request > END );
108
109   return request;
110 } // end function getRequest
111
112 // determine whether to display given record
113 bool shouldDisplay( int type, double balance )
114 {
115    // determine whether to display zero balances
116    if ( type == ZERO_BALANCE && balance == 0 )
117       return true;
118
119    // determine whether to display credit balances
120    if ( type == CREDIT_BALANCE && balance < 0 )
121       return true;
122
123    // determine whether to display debit balances
124    if ( type == DEBIT_BALANCE && balance > 0 )
125       return true;
126
127    return false;
128 } // end function shouldDisplay
129
130 // display single record from file
```

```
131 void outputLine( int account, const string name, double balance )
132 {
133    cout << left << setw( 10 ) << account << setw( 13 ) << name
134       << setw( 7 ) << setprecision( 2 ) << right << balance << endl;
135 } // end function outputLine
```

```
Enter request
    1-  List accounts with zero balances
    2-  List accounts with credit balances
    3-  List accounts with debit balances
    4-  End of run
    ? 1

    Accounts with zero balances:
    300      White            0.00

    Enter request
    1-  List accounts with zero balances
    2-  List accounts with credit balances
    3-  List accounts with debit balances
    4-  End of run
    ? 2

    Accounts with credit balances:
    400      Stone          -42.16

    Enter request
    1-  List accounts with zero balances
    2-  List accounts with credit balances
    3-  List accounts with debit balances
    4-  End of run
    ? 3

    Accounts with debit balances:
    100      Jones           24.98
    200      Doe            345.67
    500      Rich           224.62

    Enter request
    1-  List accounts with zero balances
    2-  List accounts with credit balances
    3-  List accounts with debit balances
    4-  End of run
    ? 4
    End of run.
```

شکل ۸-۱۷ | برنامه پرس و جوی اعتبار.

۶-۱۷ به روز کردن فایل‌های ترتیبی

داده‌ای که قالب‌بندی شده و در یک فایل ترتیبی همانند برنامه شکل ۴-۱۷ نوشته شده باشد، بدون ریسک از بین رفتن سایر داده‌ها در فایل امکان تغییر و اصلاح در آن وجود ندارد. برای مثال، اگر نام "White" نیاز به تغییر به "Worthington" داشته باشد، نام قدیمی نمی‌تواند بدون معیوب ساختن فایل بازنویسی شود. رکورد White در فایل بصورت زیر نوشته شده است

```
300 White 0.00
```

اگر این رکورد از ابتدای همان مکان در فایل با استفاده از یک نام طولانی‌تر مجدداً نوشته شود، رکورد بصورت زیر در خواهد آمد

```
300 Worthingont 0.00
```

رکورد جدید حاوی شش کاراکتر بیش از رکورد قبلی است. از اینرو کاراکترهای قرار گرفته پس از دومین "o" در "Worthington" بر روی ابتدای رکورد بعدی در فایل نوشته خواهند شد. مشکل اینجاست

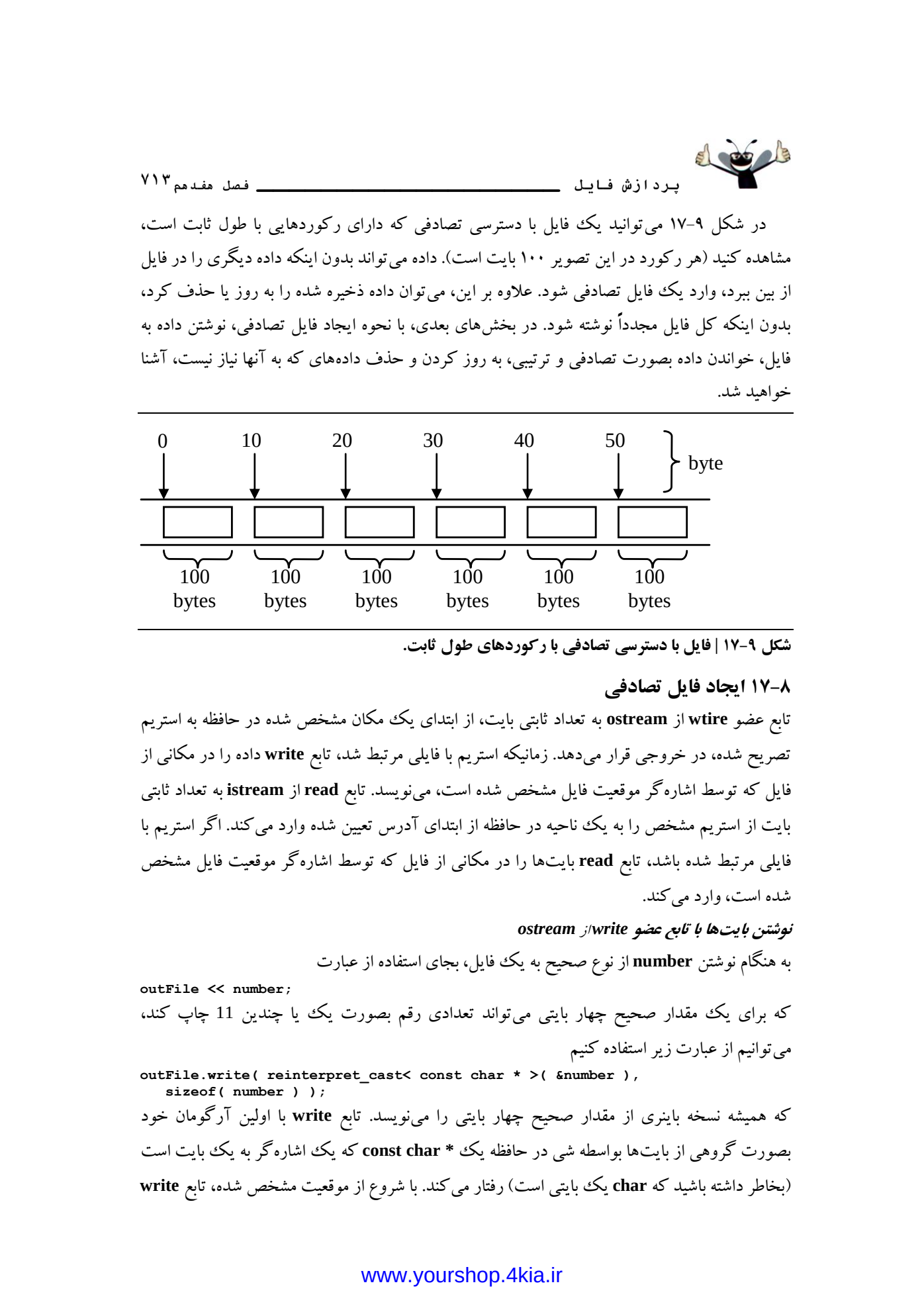
که، در مدل ورودی/خروجی قالب‌بندی شده از عملگرهای >> و <<، فیلدها یا رکوردها می‌توانند هر سایزی داشته باشند. برای مثال مقادیر 7, 14, 117-، 2047 و 27383 همگی از نوع صحیح هستند که در تعداد بایت یکسانی «داده خام» ذخیره می‌شوند. (در کامپیوترهای 32 بیتی، این مقدار چهار بایت است). با این وجود، این مقادیر صحیح به هنگام خروجی بعنوان متن قالب‌بندی شده به فیلدهای با سایز متفاوت تبدیل می‌شوند. بنابر این، معمولاً مدل ورودی/خروجی قالب‌بندی شده برای به روز کردن رکوردها بکار گرفته نمی‌شود.

به روز کردن چنین فایلهای به روش ضعیفی صورت می‌گیرد. برای مثال، برای تغییر نام مطرح شده در فوق، می‌توان رکوردهای قبل از 300 White 0.00 در یک فایل ترتیبی را به یک فایل جدید کپی کرده، سپس رکورد به روز شده را به فایل جدید بنویسیم و رکوردهای پس از 300 White 0.00 را به فایل جدید کپی نمائیم. این فرآیند برای به روز کردن هر رکوردی در فایل لازم است.

۱۷-۷ فایل با دسترسی تصادفی

تا بدین جا به توضیح نحوه ایجاد فایل‌ها با دسترسی ترتیبی و جستجو در آنها پرداخته‌ایم. با این همه، فایل‌ها با دسترسی ترتیبی برای برنامه‌های که نیاز به "دسترسی آنی" دارند مناسب نیستند. در چنین برنامه‌هایی باید سرعت و بطور آنی به اطلاعات یک رکورد خاص دسترسی پیدا کرد. از جمله برنامه‌های دسترسی آنی می‌توان سیستم‌های رزرو خطوط هوایی، سیستم‌های بانکی، سیستم‌های فروش، ماشین‌های پرداخت اتوماتیک و دیگر سیستم‌های پردازش تراکنشی نام برد، که مستلزم دسترسی سریع به داده خاصی هستند. در یک سیستم بانکی ممکن است حساب صدها، هزاران یا میلیون‌ها مشتری وجود داشته باشد، در چنین سیستمی باید در عرض چند ثانیه حساب مورد نظر پیدا شود. چنین دسترسی آنی به کمک فایل‌های تصادفی امکان‌پذیر است. به رکوردهای متمایز در یک فایل با دسترسی تصادفی می‌توان بطور مستقیم و بسرعت دسترسی پیدا کرد بدون اینکه جستجوهای زیادی در میان رکوردهای دیگر انجام داد که در فایل‌هایی با دسترسی ترتیبی انجام آن ضروری است. فایل‌ها با دسترسی تصادفی با عنوان فایل‌هایی با دسترسی مستقیم نیز شناخته می‌شوند.

همانطوری که در اوایل این فصل هم گفته شد، C++ ساختار خاصی بر روی فایل‌ها اعمال نمی‌کند، از اینرو برنامه‌هایی که از فایل‌های تصادفی استفاده می‌کنند، بایستی از قابلیت دسترسی تصادفی برخوردار باشند. تکنیک‌های گوناگونی برای ایجاد فایل‌های تصادفی وجود دارد. شاید ساده‌ترین روش این باشد که تمام رکوردها دارای طول ثابت در فایل باشند. به هنگام استفاده از رکوردهایی با طول ثابت، برنامه می‌تواند مکان دقیق هر رکورد را با توجه به ابتدای فایل محاسبه کند.



در شکل ۹-۱۷ می‌توانید یک فایل با دسترسی تصادفی که دارای رکوردهایی با طول ثابت است، مشاهده کنید (هر رکورد در این تصویر ۱۰۰ بایت است). داده می‌تواند بدون اینکه داده دیگری را در فایل از بین ببرد، وارد یک فایل تصادفی شود. علاوه بر این، می‌توان داده ذخیره شده را به روز یا حذف کرد، بدون اینکه کل فایل مجدداً نوشته شود. در بخش‌های بعدی، با نحوه ایجاد فایل تصادفی، نوشتن داده به فایل، خواندن داده بصورت تصادفی و ترتیبی، به روز کردن و حذف داده‌های که به آنها نیاز نیست، آشنا خواهید شد.

شکل ۹-۱۷ | فایل با دسترسی تصادفی با رکوردهای طول ثابت.

۸-۱۷ ایجاد فایل تصادفی

تابع عضو **wtire** از **ostream** به تعداد ثابتی بایت، از ابتدای یک مکان مشخص شده در حافظه به استریم تصریح شده، در خروجی قرار می‌دهد. زمانیکه استریم با فایلی مرتبط شد، تابع **write** داده را در مکانی از فایل که توسط اشاره‌گر موقعیت فایل مشخص شده است، می‌نویسد. تابع **read** از **istream** به تعداد ثابتی بایت از استریم مشخص را به یک ناحیه در حافظه از ابتدای آدرس تعیین شده وارد می‌کند. اگر استریم با فایلی مرتبط شده باشد، تابع **read** بایت‌ها را در مکانی از فایل که توسط اشاره‌گر موقعیت فایل مشخص شده است، وارد می‌کند.

نوشتن بایت‌ها با تابع عضو write از ostream

به هنگام نوشتن **number** از نوع صحیح به یک فایل، بجای استفاده از عبارت

```
outFile << number;
```

که برای یک مقدار صحیح چهار بایتی می‌تواند تعدادی رقم بصورت یک یا چندین 11 چاپ کند، می‌توانیم از عبارت زیر استفاده کنیم

```
outFile.write( reinterpret_cast< const char * >( &number ),
   sizeof( number ) );
```

که همیشه نسخه باینری از مقدار صحیح چهار بایتی را می‌نویسد. تابع **write** با اولین آرگومان خود بصورت گروهی از بایت‌ها بواسطه شی در حافظه یک **const char *** که یک اشاره‌گر به یک بایت است (بخاطر داشته باشید که **char** یک بایتی است) رفتار می‌کند. با شروع از موقعیت مشخص شده، تابع **write**

تعداد بایت تعیین شده توسط آرگومان دوم را خارج می‌سازد، یک مقدار صحیح از نوع **size_t**. همانطور که مشاهده خواهید کرد، تابع **read** می‌تواند متعاقبا برای خواندن چهار بایت و قرار دادن آنها در متغیر **number** بکار گرفته شود.

تبدیل مابین انواع اشاره‌گر با عملگر reinterpret_cast

متاسفانه، اغلب اشاره‌گرهای که به تابع **write** بعنوان اولین آرگومان ارسال می‌کنیم از نوع **const char *** نیستند. برای خارج ساختن شی‌ها از انواع دیگر، بایستی اشاره‌گر به این شی‌ها را به نوع **const char *** تبدیل کنیم، در غیر اینصورت کامپایلر قادر به کامپایل فراخوانی تابع **write** نخواهد بود. C++ عملگر ***reinterpret_cast*** را برای چنین مواردی آماده کرده است. همچنین می‌توانید از این عملگر تبدیل برای تبدیل مابین اشاره‌گر و نوع‌های صحیح و برعکس استفاده کنید. بدون حضور **reinterpret_cast**، عبارت **write** که می‌خواهد **number** را خارج سازد، کامپایل نخواهد شد، چرا که کامپایلر اجازه نمی‌دهد تا اشاره‌گر از نوع **int *** (نوع برگشتی توسط عبارت **&number**) به تابعی ارسال شود که در انتظار آرگومانی از نوع **const char *** است، تا آنجا که به کامپایلر مربوط می‌شود، این نوع‌ها با هم سازگار نیستند.

عملگر **reinterpret_cast** در زمان کامپایل وارد عمل می‌شود و تغییری در مقدار شی که عملوند آن به آن اشاره می‌کند بوجود نمی‌آورد. بجای آن، از کامپایلر تقاضا می‌کند تا عملوند را بعنوان نوع هدف دوباره تفسیر کند (مشخص شده در درون < > که پس از کلمه کلیدی **reinterpret_cast** آورده می‌شود). در برنامه شکل ۱۲-۱۷ از این عملگر برای تبدیل یک اشاره‌گر **ClientData** به یک **const char *** استفاده کرده‌ایم، که شی **ClientData** را بصورت بایت‌های در خروجی یک فایل مجدداً تفسیر می‌کند. برنامه پردازش فایل با دسترسی تصادفی بندرت یک فیلد منفرد را در یک فایل می‌نویسد. معمولاً، آنها یک شی از یک کلاس را در هر بار می‌نویسند که در مثال‌های بعدی شاهد آن خواهید بود.

برنامه پردازش اعتبار

به صورت مسئله زیر توجه کنید:

یک برنامه پردازش کننده اعتبار بنویسید که قادر به ذخیره‌سازی بیش از صد رکورد با طول ثابت برای شرکتی باشد که می‌تواند بیش از صد مشتری داشته باشد. هر رکورد باید متشکل از یک شماره حساب (که همانند کلید رکورد عمل کند)، نام خانوادگی، نام و موجودی باشد. برنامه بایستی قادر به، به روز کردن حساب، وارد کردن حساب‌های جدید، حذف حساب و وارد ساختن کل رکوردهای حساب به یک فایل متنی قالب‌بندی شده با هدف چاپ باشد.

در چند بخش بعدی به معرفی تکنیک‌های بکار رفته در این برنامه خواهیم پرداخت. برنامه شکل ۱۲-۱۷ به بیان نحوه باز کردن یک فایل تصادفی، تعریف فرمت رکورد با استفاده از شی از کلاس **ClientData** (شکل‌های ۱۰-۱۷ و ۱۱-۱۷) و نوشتن داده به دیسک با فرمت باینری، پرداخته است.

این برنامه مبادرت به مقداردهی اولیه تمام صد رکورد از فایل **credit.dat** با شی‌های تهی و با استفاده از تابع **write** می‌کند هر شی تهی حاوی صفر برای شماره حساب، رشته **null** (که توسط جفت گوتیشن خالی مشخص می‌شود) برای نام خانوادگی و نام، 0.0 برای موجودی است.

```
// Fig. 17.10: ClientData.h
// Class ClientData definition used in Fig. 17.12-Fig. 17.15.
#ifndef CLIENTDATA_H
#define CLIENTDATA_H

#include <string>
using std::string;

class ClientData
{
public:
   // default ClientData constructor
   ClientData( int = 0, string = "", string = "", double = 0.0 );

   // accessor functions for accountNumber
   void setAccountNumber( int );
   int getAccountNumber() const;

   // accessor functions for lastName
   void setLastName( string );
   string getLastName() const;

   // accessor functions for firstName
   void setFirstName( string );
   string getFirstName() const;

   // accessor functions for balance
   void setBalance( double );
   double getBalance() const;
private:
   int accountNumber;
   char lastName[ 15 ];
   char firstName[ 10 ];
   double balance;
}; // end class ClientData

#endif
```

شکل ۱۰-۱۷ | فایل سرآیند ClientData.

شی‌ها از کلاس **string** دارای سایز واحدی نیستند چرا که از روش اخذ حافظه دینامیکی برای جا دادن رشته‌ها با طول‌های متفاوت استفاده می‌کنند. بایستی این برنامه رکوردهای با طول ثابت داشته باشد، از اینرو کلاس **ClientData** نام و نام خانوادگی مشتری را در آرایه‌های **char** با طول ثابت ذخیره می‌کند. توابع عضو **setLastName** (شکل ۱۱-۱۷، خطوط 37-45) و **setFirstName** (خطوط 54-62 از شکل ۱۱-۱۷) هر یک مبادرت به کپی کاراکترها از یک شی رشته بدون آرایه **char** متناظر می‌کنند. به تابع **setLanstName** توجه کنید. خط 40 مبادرت به مقداردهی اولیه **const char * lastNameValue** با نتیجه فراخوانی تابع عضو **data** می‌کند که یک آرایه حاوی کاراکترهای از رشته برگشت می‌دهد. خط 41 تابع عضو **size** را برای بدست آوردن طول رشته **lastNameString** فراخوانی می‌کند. خط 42 مطمئن می‌شود که **length** (طول) کمتر از 25 کاراکتر است، سپس خط 43 طول کاراکترها را از

lastNameValue به آرایه **lastName** کپی می‌کند. تابع عضو **setFirstName** همین مراحل را برای نام انجام می‌دهد.

```
1  // Fig. 17.11: ClientData.cpp
2  // Class ClientData stores customer's credit information.
3  #include <string>
4  using std::string;
5
6  #include "ClientData.h"
7
8  // default ClientData constructor
9  ClientData::ClientData( int accountNumberValue,
10    string lastNameValue,string firstNameValue,double balanceValue )
11 {
12    setAccountNumber( accountNumberValue );
13    setLastName( lastNameValue );
14    setFirstName( firstNameValue );
15    setBalance( balanceValue );
16 } // end ClientData constructor
17
18 // get account-number value
19 int ClientData::getAccountNumber() const
20 {
21    return accountNumber;
22 } // end function getAccountNumber
23
24 // set account-number value
25 void ClientData::setAccountNumber( int accountNumberValue )
26 {
27    accountNumber = accountNumberValue; // should validate
28 } // end function setAccountNumber
29
30 // get last-name value
31 string ClientData::getLastName() const
32 {
33    return lastName;
34 } // end function getLastName
35
36 // set last-name value
37 void ClientData::setLastName( string lastNameString )
38 {
39    // copy at most 15 characters from string to lastName
40    const char *lastNameValue = lastNameString.data();
41    int length = lastNameString.size();
42    length = ( length < 15 ? length : 14 );
43    strncpy( lastName, lastNameValue, length );
44    lastName[ length ] = '\0'; // append null character to lastName
45 } // end function setLastName
46
47 // get first-name value
48 string ClientData::getFirstName() const
49 {
50    return firstName;
51 } // end function getFirstName
52
53 // set first-name value
54 void ClientData::setFirstName( string firstNameString )
55 {
56    // copy at most 10 characters from string to firstName
57    const char *firstNameValue = firstNameString.data();
58    int length = firstNameString.size();
59    length = ( length < 10 ? length : 9 );
60    strncpy( firstName, firstNameValue, length );
61    firstName[ length ] = '\0'; // append null character to firstName
62 } // end function setFirstName
63
64 // get balance value
65 double ClientData::getBalance() const
```

```
66 {
67    return balance;
68 } // end function getBalance
69
70 // set balance value
71 void ClientData::setBalance( double balanceValue )
72 {
73    balance = balanceValue;
74 } // end function setBalance
```

شکل ۱۱-۱۷ | کلاس ClientData عرضه کننده اطلاعات اعتباری مشتری.

در شکل ۱۲-۱۷، خط 18 یک شی از **ofstream** برای فایل **credit.dat** ایجاد می‌کند. آرگومان دوم در سازنده، **ios::binary**، بر این نکته دلالت دارد که فایل را برای خروجی در مد باینری باز کرده‌ایم، که برای نوشتن رکوردهای با طول ثابت در فایل ضروری است. خطوط 32-31 سبب می‌شوند که **blankClient** در فایل **credit.dat** مرتبط با شی **outCredit** نوشته شود. بخاطر داشته باشید که عملگر **sizeof** سایز شی احاطه شده در درون پرانتز را برحسب بایت برگشت می‌دهد. آرگومان اول در تابع **write**، خط 31 بایستی از نوع **const char *** باشد. اما نوع داده **&blankClient** از نوع **ClientData *** است. برای تبدیل **&blankClient** به **const char ***، خط 31 از عملگر تبدیل **reinterpret_cast** استفاده کرده است. از اینرو فراخوانی **write** بدون اینکه خطای کامپایل بدنبال داشته باشد، صورت می‌گیرد.

```
1  // Fig. 17.12: Fig17_12.cpp
2  // Creating a randomly accessed file.
3  #include <iostream>
4  using std::cerr;
5  using std::endl;
6  using std::ios;
7
8  #include <fstream>
9  using std::ofstream;
10
11 #include <cstdlib>
12 using std::exit; // exit function prototype
13
14 #include "ClientData.h" // ClientData class definition
15
16 int main()
17 {
18    ofstream outCredit( "credit.dat", ios::binary );
19
20    // exit program if ofstream could not open file
21    if ( !outCredit )
22    {
23       cerr << "File could not be opened." << endl;
24       exit( 1 );
25    } // end if
26
27    ClientData blankClient; // constructor zeros out each data member
28
29    // output 100 blank records to file
30    for ( int i = 0; i < 100; i++ )
31     outCredit.write(reinterpret_cast< const char *>(&blankClient),
32          sizeof( ClientData ) );
33
34    return 0;
35 } // end main
```

شکل ۱۲-۱۷ | ایجاد فایل با دسترسی تصادفی با 100 رکورد خالی پشت سرهم.

۹-۱۷ نوشتن داده بصورت تصادفی در فایل با دسترسی تصادفی

برنامه شکل ۱۳-۱۷ مبادرت به نوشتن داده به فایل **credit.dat** کرده و از توابع **seekp** و **write** برای ذخیره‌سازی داده در مکان مشخص شده در فایل استفاده می‌کند. تابع **seekp** اشاره‌گر موقعیت فایل را در مکان مشخص در فایل قرار می‌دهد، سپس تابع **write** داده را می‌نویسد. دقت کنید که خط 19 شامل فایل سرآیند *ClientData.h* تعریف شده در شکل ۱۰-۱۷ است، از اینرو برنامه می‌تواند از شی‌های **ClientData** استفاده کند.

```
1  // Fig. 17.13: Fig17_13.cpp
2  // Writing to a random-access file.
3  #include <iostream>
4  using std::cerr;
5  using std::cin;
6  using std::cout;
7  using std::endl;
8  using std::ios;
9
10 #include <iomanip>
11 using std::setw;
12
13 #include <fstream>
14 using std::fstream;
15
16 #include <cstdlib>
17 using std::exit; // exit function prototype
18
19 #include "ClientData.h" // ClientData class definition
20
21 int main()
22 {
23    int accountNumber;
24    char lastName[ 15 ];
25    char firstName[ 10 ];
26    double balance;
27
28    fstream outCredit("credit.dat",ios::in | ios::out | ios::binary );
29
30    // exit program if fstream cannot open file
31    if ( !outCredit )
32    {
33       cerr << "File could not be opened." << endl;
34       exit( 1 );
35    } // end if
36
37    cout << "Enter account number (1 to 100, 0 to end input)\n? ";
38
39    // require user to specify account number
40    ClientData client;
41    cin >> accountNumber;
42
43    // user enters information, which is copied into file
44    while ( accountNumber > 0 && accountNumber <= 100 )
45    {
46       // user enters last name, first name and balance
47       cout << "Enter lastname, firstname, balance\n? ";
48       cin >> setw( 15 ) >> lastName;
49       cin >> setw( 10 ) >> firstName;
50       cin >> balance;
51
52       // set record accountNumber, lastName, firstName and balance values
53       client.setAccountNumber( accountNumber );
54       client.setLastName( lastName );
55       client.setFirstName( firstName );
56       client.setBalance( balance );
```

```
57
58       // seek position in file of user-specified record
59       outCredit.seekp( ( client.getAccountNumber() - 1 ) *
60          sizeof( ClientData ) );
61
62       // write user-specified information in file
63       outCredit.write( reinterpret_cast< const char * >( &client ),
64          sizeof( ClientData ) );
65
66       // enable user to enter another account
67       cout << "Enter account number\n? ";
68       cin >> accountNumber;
69    } // end while
70
71    return 0;
72 } // end main
```

```
Enter account number (1 to 100, 0 to end input)
? 37
Enter lastname, firstname, balance
? Barker Doug 0.00
Enter account number
? 29
Enter lastname, firstname, balance
? Brown Nancy -24.54
Enter account number
?96
Enter lastname, firstname, balance
? Stone Sam 34.98
Enter account number
? 88
Enter lastname, firstname, balance
? Smith Dave 258.34
Enter account number
? 33
Enter lastname, firstname, balance
? Dunn Stacey 314.33
Enter account number
? 0
```

شکل ۱۳-۱۷ | نوشتن داده در فایل با دسترسی تصادفی.

خطوط 60-59 اشاره‌گر موقعیت فایل را برای شی **outCredit** برحسب بایت و با استفاده از محاسبه زیر انتقال می‌دهند

```
( client.getAccountNumber() - 1) * sizeof( ClientData )
```

چون شماره حساب مابین 1 و 100 است، 1 به هنگام محاسبه موقعیت بایت رکورد به هنگام محاسبه کسر می‌شود. از اینرو، برای رکورد 1، اشاره‌گر موقعیت فایل با بایت صفر در فایل تنظیم می‌شود. در خط 28 از شی **outCredit** برای باز کردن فایل **credit.data** استفاده شده است. فایل در مد باینری برای خروجی و ورودی باز شده است که ترکیبی از مدهای **ios::in**، **ios::out** و **ios:binary** است. با استفاده از عملگر **OR** انحصاری (|) می‌توان چندین مد باز کردن فایل را در کنار هم بکار گرفت. باز کردن فایل موجود **credit.dat** به این روش، ما را مطمئن می‌سازد که این برنامه می‌تواند رکوردهای نوشته شده در فایل توسط برنامه ۱۲-۱۷ را نگهداری کند، بجای اینکه فایل را از اول ایجاد کنیم.

۱۰-۱۷ خواندن داده از فایل با دسترسی تصادفی بفرم ترتیبی

در بخش‌های قبلی، یک فایل با دسترسی تصادفی ایجاد کرده و داده‌های در آن فایل نوشتیم. در این بخش برنامه‌ای ایجاد می‌کنیم که فایل را بصورت ترتیبی یا پشت سرهم خوانده و فقط رکوردهای که حاوی اطلاعات هستند چاپ کند.

تابع **read** تعداد بایت‌های مشخص شده از موقعیت جاری در استریم تصریح شده را وارد می‌سازد. برای مثال، خطوط 58-57 از شکل ۱۴-۱۷ تعداد بایت‌های مشخص شده توسط **sizeof(ClientData)** را از طریق **inCredit** خوانده و داده را در رکورد **client** ذخیره می‌سازد. توجه کنید که تابع **read** مستلزم آن است که آرگومان اول از نوع *** char** باشد. از آنجا که **&Client** از نوع *** ClientData** است بایستی با استفاده از عملگر تبدیل **reinterpret_cast** تبدیل به *** char** شود. خط 24 شامل فایل سرآیند ***clientData.h*** تعریف شده در شکل ۱۰-۱۷ است و از اینرو برنامه می‌تواند از شی‌های **ClientData** استفاده کند.

برنامه شکل ۱۴-۱۷ بصورت ترتیبی هر رکورد موجود در فایل **credit.dat** را می‌خواند و بررسی می‌کند که آیا رکورد حاوی داده است یا خیر و رکوردهای حاوی داده را بصورت قالب‌بندی شده به نمایش در می‌آورد. شرط موجود در خط 50 از تابع عضو **eof** برای تعیین اینکه به انتهای فایل رسیده است یا خیر استفاده می‌کند و سبب می‌شود تا اجرای عبارت **while** خاتمه پذیرد. همچنین اگر خطای به هنگام خواندن از فایل رخ دهد، حلقه خاتمه می‌یابد، چرا که **inCredit** با **false** ارزیابی می‌شود. داده وارد شده به فایل توسط تابع **outputLine** خارج می‌شود (خطوط 72-65) که دو آرگومان دریافت می‌کند، یک شی **ostream** و یک ساختار **clientData** برای خروجی. نوع پارامتر **ostream** جالب توجه است چرا که هر شی **ostream** (همانند **out**) یا هر شی از کلاس مشتق شده از **ostream** (همانند یک شی از نوع **ofstream**) می‌تواند بعنوان آرگومان در نظر گرفته شود. به این معنی که همان تابع می‌تواند بکار گرفته شود.

```
1  // Fig. 17.14: Fig17_14.cpp
2  // Reading a random access file sequentially.
3  #include <iostream>
4  using std::cerr;
5  using std::cout;
6  using std::endl;
7  using std::fixed;
8  using std::ios;
9  using std::left;
10 using std::right;
11 using std::showpoint;
12
13 #include <iomanip>
14 using std::setprecision;
15 using std::setw;
16
17 #include <fstream>
18 using std::ifstream;
19 using std::ostream;
20
```

```
21 #include <cstdlib>
22 using std::exit; // exit function prototype
23
24 #include "ClientData.h" // ClientData class definition
25
26 void outputLine( ostream&, const ClientData & ); // prototype
27
28 int main()
29 {
30    ifstream inCredit( "credit.dat", ios::in );
31
32    // exit program if ifstream cannot open file
33    if ( !inCredit )
34    {
35       cerr << "File could not be opened." << endl;
36       exit( 1 );
37    } // end if
38
39    cout << left << setw( 10 ) << "Account" << setw( 16 )
40       << "Last Name" << setw( 11 ) << "First Name" << left
41       << setw( 10 ) << right << "Balance" << endl;
42
43    ClientData client; // create record
44
45    // read first record from file
46    inCredit.read( reinterpret_cast< char * >( &client ),
47       sizeof( ClientData ) );
48
49    // read all records from file
50    while ( inCredit && !inCredit.eof() )
51    {
52       // display record
53       if ( client.getAccountNumber() != 0 )
54          outputLine( cout, client );
55
56       // read next from file
57       inCredit.read( reinterpret_cast< char * >( &client ),
58          sizeof( ClientData ) );
59    } // end while
60
61    return 0;
62 } // end main
63
64 // display single record
65 void outputLine( ostream &output, const ClientData &record )
66 {
67    output << left << setw( 10 ) << record.getAccountNumber()
68       << setw( 16 ) << record.getLastName()
69       << setw( 11 ) << record.getFirstName()
70       << setw( 10 ) << setprecision( 2 ) << right << fixed
71       << showpoint << record.getBalance() << endl;
72 } // end function outputLine
```

```
Account    Last Name       First Name      Balance
29         Brown           Nancy            -24.54
33         Dunn            Stacey           314.33
37         Barker          Doug               0.00
88         Smith           Dave             258.34
96         Stone           Sam               34.98
```

شکل ۱۷-۱۴ | خواندن از یک فایل تصادفی بصورت ترتیبی.

۱۷-۱۱ مبحث آموزشی: برنامه پردازش تراکنشی

در این بخش به معرفی اصول یک برنامه پردازش تراکنشی (شکل ۱۵-۱۷) با استفاده از یک فایل تصادفی می‌پردازیم که پردازش‌های را با دسترسی فوری انجام می‌دهد. این برنامه اطلاعات حساب بانکی را در خود نگهداری می‌کند. برنامه قادر به، به روز کردن حساب‌های موجود، افزودن حساب‌های جدید، حذف

حساب و ذخیره‌سازی تمام حساب‌های جاری بصورت یک لیست قالب‌بندی شده در یک فایل متنی است. فرض می‌کنیم که برنامه شکل ۱۲–۱۷ برای ایجاد فایل **credit.dat** اجرا شده است و برنامه شکل ۱۳–۱۷ هم برای وارد ساختن داده‌های اولیه بکار گرفته شده باشد.

```
1  // Fig. 17.15: Fig17_15.cpp
2  // This program reads a random access file sequentially, updates
3  // data previously written to the file, creates data to be placed
4  // in the file, and deletes data previously in the file.
5  #include <iostream>
6  using std::cerr;
7  using std::cin;
8  using std::cout;
9  using std::endl;
10 using std::fixed;
11 using std::ios;
12 using std::left;
13 using std::right;
14 using std::showpoint;
15
16 #include <fstream>
17 using std::ofstream;
18 using std::ostream;
19 using std::fstream;
20
21 #include <iomanip>
22 using std::setw;
23 using std::setprecision;
24
25 #include <cstdlib>
26 using std::exit; // exit function prototype
27
28 #include "ClientData.h" // ClientData class definition
29
30 int enterChoice();
31 void createTextFile( fstream& );
32 void updateRecord( fstream& );
33 void newRecord( fstream& );
34 void deleteRecord( fstream& );
35 void outputLine( ostream&, const ClientData & );
36 int getAccount( const char * const );
37
38 enum Choices { PRINT = 1, UPDATE, NEW, DELETE, END };
39
40 int main()
41 {
42    // open file for reading and writing
43    fstream inOutCredit( "credit.dat", ios::in | ios::out );
44
45    // exit program if fstream cannot open file
46    if ( !inOutCredit )
47    {
48       cerr << "File could not be opened." << endl;
49       exit ( 1 );
50    } // end if
51
52    int choice; // store user choice
53
54    // enable user to specify action
55    while ( ( choice = enterChoice() ) != END )
56    {
57       switch ( choice )
58       {
59          case PRINT: // create text file from record file
60             createTextFile( inOutCredit );
61             break;
62          case UPDATE: // update record
63             updateRecord( inOutCredit );
```

```
64             break;
65          case NEW: // create record
66             newRecord( inOutCredit );
67             break;
68          case DELETE: // delete existing record
69             deleteRecord( inOutCredit );
70             break;
71          default://display error if user does not select valid choice
72             cerr << "Incorrect choice" << endl;
73             break;
74       } // end switch
75
76       inOutCredit.clear(); // reset end-of-file indicator
77    } // end while
78
79    return 0;
80 } // end main
81
82 // enable user to input menu choice
83 int enterChoice()
84 {
85    // display available options
86    cout << "\nEnter your choice" << endl
87       << "1 - store a formatted text file of accounts" << endl
88       << "    called \"print.txt\" for printing" << endl
89       << "2 - update an account" << endl
90       << "3 - add a new account" << endl
91       << "4 - delete an account" << endl
92       << "5 - end program\n? ";
93
94    int menuChoice;
95    cin >> menuChoice; // input menu selection from user
96    return menuChoice;
97 } // end function enterChoice
98
99 // create formatted text file for printing
100 void createTextFile( fstream &readFromFile )
101 {
102    // create text file
103    ofstream outPrintFile( "print.txt", ios::out );
104
105    // exit program if ofstream cannot create file
106    if ( !outPrintFile )
107    {
108       cerr << "File could not be created." << endl;
109       exit( 1 );
110    } // end if
111
112    outPrintFile << left << setw( 10 ) << "Account" << setw( 16 )
113       << "Last Name" << setw( 11 ) << "First Name" << right
114       << setw( 10 ) << "Balance" << endl;
115
116    // set file-position pointer to beginning of readFromFile
117    readFromFile.seekg( 0 );
118
119    // read first record from record file
120    ClientData client;
121    readFromFile.read( reinterpret_cast< char * >( &client ),
122       sizeof( ClientData ) );
123
124    // copy all records from record file into text file
125    while ( !readFromFile.eof() )
126    {
127       // write single record to text file
128       if ( client.getAccountNumber() != 0 ) // skip empty records
129          outputLine( outPrintFile, client );
130
131       // read next record from record file
132       readFromFile.read( reinterpret_cast< char * >( &client ),
133          sizeof( ClientData ) );
```

```
134    } // end while
135 } // end function createTextFile
136
137 // update balance in record
138 void updateRecord( fstream &updateFile )
139 {
140    // obtain number of account to update
141    int accountNumber = getAccount( "Enter account to update" );
142
143    // move file-position pointer to correct record in file
144    updateFile.seekg( ( accountNumber - 1 ) * sizeof( ClientData ) );
145
146    // read first record from file
147    ClientData client;
148    updateFile.read( reinterpret_cast< char * >( &client ),
149       sizeof( ClientData ) );
150
151    // update record
152    if ( client.getAccountNumber() != 0 )
153    {
154       outputLine( cout, client ); // display the record
155
156       // request user to specify transaction
157       cout << "\nEnter charge (+) or payment (-): ";
158       double transaction; // charge or payment
159       cin >> transaction;
160
161       // update record balance
162       double oldBalance = client.getBalance();
163       client.setBalance( oldBalance + transaction );
164       outputLine( cout, client ); // display the record
165
166       // move file-position pointer to correct record in file
167       updateFile.seekp((accountNumber - 1) * sizeof( ClientData ) );
168
169       // write updated record over old record in file
170       updateFile.write( reinterpret_cast< const char * >( &client ),
171          sizeof( ClientData ) );
172    } // end if
173    else // display error if account does not exist
174       cerr << "Account #" << accountNumber
175          << " has no information." << endl;
176 } // end function updateRecord
177
178 // create and insert record
179 void newRecord( fstream &insertInFile )
180 {
181    // obtain number of account to create
182    int accountNumber = getAccount( "Enter new account number" );
183
184    // move file-position pointer to correct record in file
185    insertInFile.seekg((accountNumber - 1) * sizeof( ClientData ) );
186
187    // read record from file
188    ClientData client;
189    insertInFile.read( reinterpret_cast< char * >( &client ),
190       sizeof( ClientData ) );
191
192    // create record, if record does not previously exist
193    if ( client.getAccountNumber() == 0 )
194    {
195       char lastName[ 15 ];
196       char firstName[ 10 ];
197       double balance;
198
199       // user enters last name, first name and balance
200       cout << "Enter lastname, firstname, balance\n? ";
201       cin >> setw( 15 ) >> lastName;
202       cin >> setw( 10 ) >> firstName;
203       cin >> balance;
```

```
204
205       // use values to populate account values
206       client.setLastName( lastName );
207       client.setFirstName( firstName );
208       client.setBalance( balance );
209       client.setAccountNumber( accountNumber );
210
211       // move file-position pointer to correct record in file
212       insertInFile.seekp((accountNumber - 1) * sizeof(ClientData) );
213
214       // insert record in file
215       insertInFile.write(reinterpret_cast<const char * >( &client ),
216          sizeof( ClientData ) );
217    } // end if
218    else // display error if account already exists
219       cerr << "Account #" << accountNumber
220          << " already contains information." << endl;
221 } // end function newRecord
222
223 // delete an existing record
224 void deleteRecord( fstream &deleteFromFile )
225 {
226    // obtain number of account to delete
227    int accountNumber = getAccount( "Enter account to delete" );
228
229    // move file-position pointer to correct record in file
230    deleteFromFile.seekg(( accountNumber - 1) * sizeof(ClientData));
231
232    // read record from file
233    ClientData client;
234    deleteFromFile.read( reinterpret_cast< char * >( &client ),
235       sizeof( ClientData ) );
236
237    // delete record, if record exists in file
238    if ( client.getAccountNumber() != 0 )
239    {
240       ClientData blankClient; // create blank record
241
242       // move file-position pointer to correct record in file
243       deleteFromFile.seekp( ( accountNumber - 1 ) *
244          sizeof( ClientData ) );
245
246       // replace existing record with blank record
247       deleteFromFile.write(
248          reinterpret_cast< const char * >( &blankClient ),
249          sizeof( ClientData ) );
250
251       cout << "Account #" << accountNumber << " deleted.\n";
252    } // end if
253    else // display error if record does not exist
254       cerr << "Account #" << accountNumber << " is empty.\n";
255 } // end deleteRecord
256
257 // display single record
258 void outputLine( ostream &output, const ClientData &record )
259 {
260    output << left << setw( 10 ) << record.getAccountNumber()
261       << setw( 16 ) << record.getLastName()
262       << setw( 11 ) << record.getFirstName()
263       << setw( 10 ) << setprecision( 2 ) << right << fixed
264       << showpoint << record.getBalance() << endl;
265 } // end function outputLine
266
267 // obtain account-number value from user
268 int getAccount( const char * const prompt )
269 {
270    int accountNumber;
271
272    // obtain account-number value
273    do
```

```
274    {
275       cout << prompt << " (1 - 100): ";
276       cin >> accountNumber;
277    } while ( accountNumber < 1 || accountNumber > 100 );
278
279    return accountNumber;
280 } // end function getAccount
```

شکل ۱۵-۱۷ | برنامه حساب بانکی.

برنامه دارای پنج گزینه است (گزینه5 برای خاتمه دادن به برنامه). گزینه1 تابع **createTextFile** را برای ذخیره یک لیست قالب‌بندی شده از تمام اطلاعات حساب در یک فایل متنی بنام **print.txt** که می‌توان از آن چاپ گرفت، فراخوانی می‌کند. تابع **createTextFile** در خطوط 135-100 یک شی از **fstream** بعنوان آرگومان دریافت و برای وارد کردن داده از فایل **credit.dat** بکار می‌گیرد. تابع **createTextFile** تابع عضو **read** را فراخوانی کرده (خطوط 133-132) و از تکنیک دسترسی پشت سرهم یا ترتیبی فایل (شکل ۱۴-۱۷) برای وارد کردن داده از **credit.dat** استفاده می‌کند. تابع **outputLine** که در بخش ۱۰-۱۷ توضیح داده شده است، برای نوشتن داده در فایل **print.txt** بکار گرفته شده است. توجه کنید تابع **createTextFile** از تابع عضو **seekg** در خط 117 برای اطمینان از اینکه اشاره‌گر موقعیت فایل در ابتدای فایل قرار گرفته باشد، استفاد کرده است. پس از انتخاب گزینه**1**، فایل **print.txt** حاوی داده‌های زیر خواهد بود

Account	Last Name	First Name	Balance
29	Brown	Nancy	-24.54
33	Bahram	Stacey	314.33
37	Barker	Doug	0.00
88	Smith	Dave	258.34
96	Stone	Sam	34.98

گزینه2 تابع **updateRecord** را برای به روز کردن یک حساب فراخوانی می‌کند (خطوط 176-138). این تابع فقط یک رکورد موجود را به روز می‌کند از اینرو، ابتدا تابع تعیین می‌کند که آیا رکورد مشخص شده خالی است یا خیر. خطوط 149-148 داده را بدرون شی **client** با استفاده از تابع عضو **read** می‌خواند. سپس خط 152 به مقایسه مقدار برگشتی توسط **getAccountNumber** از ساختار **client** با صفر می‌کند تا تعیین کند که آیا حاوی اطلاعات است یا خیر. اگر این مقدار صفر باشد، خطوط 175-174 یک پیغام خطا چاپ می‌کنند که دلالت بر خالی (تهی) بودن رکورد دارد. اگر رکورد حاوی اطلاعات باشد، خط 154 رکورد را با استفاده از تابع **outputLine** به نمایش در آورده، خط 159 مقدار تراکنشی را وارد، خطوط 171-162 موجودی جدید را محاسبه و رکورد مجدداً در فایل نوشته می‌شود. خروجی نمونه از گزینه2 در زیر آورده شده است

```
Enter account to update (1 – 100): 37
37         Barker         Doug           0.00
```

Enter charge (+) or payment (-): **+87.99**
37 Barker Doug 87.99

گزینه3، تابع **newRecord** را برای افزودن یک حساب جدید به فایل فراخوانی می‌کند (خط 221-179). اگر کاربر شماره حسابی وارد کند که از قبل وجود داشته باشد، **newRecord** یک پیغام خطا مبنی بر وجود حساب به نمایش در می‌آورد (خطوط 220-219). این تابع یک حساب جدید به همان روش بکار رفته در برنامه شکل ۱۲-۱۷ اضافه می‌کند. خروجی نمونه از گزینه3 در زیر آورده شده است

Enter new account number (1 – 100): 22
Enter lastname, firstname, balance
? **Johnston Sarah 247.45**

گزینه4 تابع **deleteRecord** را برای حذف یک رکورد از فایل فراخوانی می‌کند (خطوط 255-224). خط 227 به کاربر اعلان می‌کند تا شماره حسابی را وارد سازد. فقط امکان حذف از میان رکوردهای موجود وجود دارد، از اینرو اگر حساب انتخاب شده تهی باشد، خط 254 یک پیغام خطا به نمایش در می‌آورد. اگر حساب موجود باشد، خطوط 249-247 آن حساب را با کپی کردن یک رکورد خالی (**blankClient**) بر روی آن مجدداً مقداردهی اولیه می‌کنند. خط 251 پیغامی به نمایش درآورده و به کاربر اطلاع می‌دهد که رکورد حذف شده است. خروجی نمونه از گزینه4 در زیر آورده شده است.

Enter account to delete (1 – 100): **29**
Account #29 deleted

۱۲-۱۷ شی‌های از ورودی/خروجی

در این فصل و فصل پانزدهم به معرفی روش شی‌گرای ++C در ورودی/خروجی پرداختیم. با این همه، مثال‌های مطرح شده در اینجا متمرکز بر عملیات I/O بر روی نوع داده‌های متداول بجای شی‌های از نوع تعریف شده توسط کاربر بودند. در فصل یازدهم، نشان دادیم که چگونه شی‌ها با استفاده از سربارگذاری عملگر وارد و خارج می‌شوند. ورودی شی را با سربارگذاری عملگر >> بر روی **istream** متناسب، انجام دادیم و خروجی شی را توسط سربارگذاری عملگر << بر روی **osream** متناسب پیاده‌سازی کردیم. در هر دو مورد، فقط اعضای داده شی‌ها وارد یا خارج شدند و در هر مورد، دارای قالب‌بندی با معنی فقط بر روی شی‌ها از نوع داده انتزاعی خاص بودند. توابع عضو شی با داده شی وارد یا خارج نمی‌شوند، بجای آن، یک کپی از توابع عضو کلاس بصورت داخلی نگهداری می‌شوند و توسط تمام شی‌های کلاس به اشتراک گذاشته می‌شوند.

زمانیکه اعضای داده شی بر روی یک فایل دیسک نوشته می‌شوند، اطلاعات نوع شی از دست می‌رود، فقط بایت‌های داده و نه اطلاعات نوع را بر روی دیسک ذخیره می‌کنیم. اگر برنامه‌ای که این داده‌ها را می‌خواند از نوع شی مرتبط با داده مطلع باشد، برنامه داده‌ها را از درون شی‌های از آن نوع خواهد خواند.

از همه جالب‌تر اینجاست که شی‌های از نوع‌های متفاوت را در یک فایل ذخیره سازیم. چگونه می‌توانیم مابین آنها تمایز قائل شویم؟ مشکل اینجاست که معمولاً شی‌ها دارای فیلدهای نوع نیستند.

یک راه حل برای این مشکل می‌تواند این باشد که هر عملگر سربارگذاری شده خروجی، یک کد نوع قبل از هر مجموعه اعضای داده که نشاندهنده یک شی هستند، به نمایش در آورد (یا بنویسد). سپس وارد شدن شی همیشه با خواندن فیلد کد نوع شروع شده و با استفاده از یک عبارت **switch** می‌توان تابع سربارگذاری شده مناسب را فراخوانی کرد. اگرچه این روش فاقد ظرافت برنامه‌نویسی چند ریختی است، اما یک مکانیزم عملی برای حفظ شی‌ها در فایل‌ها و بازیابی آنها هنگام نیاز است.