فصل چهارم

عبارات کنترلی: بخش ۱

اهداف

- آشنائی با تکنیک‌های اصول حل مسائل.
- توسعه الگوریتم‌ها به طریق فرآیندهای از بالا به پایین و اصلاح گام به گام.
- بکارگیری عبارات انتخاب if و if..else برای انتخاب از میان چندین عملکرد مختلف.
- بکارگیری عبارات تکرار while برای اجرای تکراری عبارات در برنامه.
- آشنائی با شمارنده-کنترل تکرار و مراقبت-کنترل تکرار.
- بکارگیری عملگرهای افزایشی، کاهشی و تخصیصی.

رئوس مطالب

۴-۱ مقدمه
۴-۲ الگوریتم
۴-۳ شبه‌کد
۴-۴ عبارات کنترل
۴-۵ عبارت انتخاب if
۴-۶ عبارت انتخاب if..else
۴-۷ عبارت تکرار while
۴-۸ فرموله کردن الگوریتم‌ها: شمارنده-کنترل تکرار
۴-۹ فرموله کردن الگوریتم‌ها: مراقبت-کنترل تکرار
۴-۱۰ فرموله کردن الگوریتم‌ها: عبارات کنترلی تودرتو
۴-۱۱ عملگرهای تخصیص‌دهنده
۴-۱۲ عملگرهای افزاینده و کاهنده
۴-۱۳ مبحث آموزشی مهندسی نرم‌افزار:شناسائی صفات کلاس در ATM

۴-۱ مقدمه

قبل از نوشتن یک برنامه که بتواند مسئله‌ای را به طور دقیق حل کند، ضروری است که درک صحیحی از مسئله داشته باشیم و بوسیله یک طراحی دقیق به آن اقدام کنیم. به هنگام نوشتن یک برنامه، سازماندهی انواع بلوک‌های ایجاد کننده برنامه به منظور بهبود قوانین عملی ساخت برنامه بسیار مهم است. در این فصل و فصل بعدی در مورد تئوری و قواعد علمی برنامه‌نویسی ساخت‌یافته بحث خواهیم کرد. تکنیک‌های معرفی شده در این فصل در اکثر زبان‌های سطح بالا نظیر کاربرد دارند. با معرفی مفاهیم مطرح شده در اینجا متوجه مزیت عبارتهای کنترل در ایجاد و دستکاری شی‌ها خواهید شد. عبارتهای کنترل معرفی شده در این فصل در ایجاد سریع و آسانتر شی‌ها کمک کننده هستند.

در این فصل به معرفی عبارات **if**، **if..else** و **while** در زبان C++ خواهیم پرداخت. سه بلوک سازنده که به برنامه‌نویسان امکان می‌دهند تا منطق مورد نیاز توابع عضو را برای انجام وظایف مشخص کنند. بخشی از این فصل و فصل‌های ۵ و ۷ را برای توسعه دادن کلاس **GradeBook** معرفی شده در فصل سوم اختصاص داده‌ایم. در واقع، یک تابع عضو به کلاس **GradeBook** اضافه می‌کنیم که از عبارات کنترلی برای محاسبه میانگین نمرات دانشجویان استفاده می‌کند. مثال دیگر به معرفی روش‌های ترکیب عبارات کنترلی برای حل مسئله مشابه است. همچنین به معرفی عملگرهای تخصیص‌دهنده و عملگرهای افزاینده و کاهنده در C++ خواهیم پرداخت. این عملگرها سبب کوتاه شدن عبارات و گاهاً سادگی کار می‌شوند.

۴-۲ الگوریتم

هر مسئله محاسباتی و کامپیوتری می‌تواند با یک سری از اعمال اجرائی که به ترتیب اجرا می‌شوند، حل شود. از روال‌ها برای حل مسائل کمک گرفته می‌شود و عبارات:

۱- فعالیت‌ها از نوع اجرای هستند و

۲- فعالیت‌ها به ترتیب اجرا می‌شوند،

تعریف الگوریتم می‌باشند. با ذکر مثالی که در زیر آمده می‌توانید ترتیب اجرا و فعالیت‌ها را ببینید و متوجه شوید که ترتیب اجرا چقدر مهم است. الگوریتم "rise - and - shine" در ارتباط با مراحلی است که یک شخص از هنگام برخواستن از خواب تا رفتن به سرکار انجام می‌دهد، مراحل: (۱) برخواستن از تختخواب، (۲) پوشیدن لباس راحتی، (۳) دوش گرفتن، (۴) پوشیدن لباس، (۵) خوردن صبحانه، (۶) رفتن به محل کار. این روتین در مورد نحوه انجام کار و ضوابط تصمیم‌گیری می‌تواند موثر باشد حال اگر ترتیب اجرا به صورت زیر جابجا شود:

(۱) برخواستن از تختخواب، (۲) پوشیدن لباس راحتی، (۳) پوشیدن لباس، (۴) دوش گرفتن، (۵) خوردن صبحانه، (۶) رفتن به محل کار. در این حالت شخص مورد نظر، بایستی به محل کار با لباس خیس برود.

مشخص کردن عبارات اجرائی در یک برنامه کامپیوتری، *کنترل برنامه (program control)* نامیده شود، که به اجرای صحیح و مرتب عبارات گفته می‌شود.

۳-٤ شبه‌کد

شبه‌کد (pseudocode) یک زبان مصنوعی و فرمال است که به برنامه‌نویس کمک می‌کند تا الگوریتم خود را توسعه دهد. شبه‌کد مخصوصاً برای ایجاد الگوریتم‌های مفید است که می‌خواهیم آنها را تبدیل به برنامه‌های ساخت یافته کنیم شبه‌کد، همانند زبان روزمره انگلیسی است و مزیت آن درک آسان توسط کاربر است، اگر چه جزء زبان‌های برنامه‌نویسی اجرائی نیست. برنامه‌های شبه‌کد توسط کامپیوتر قابل اجرا نیستند. با این همه، شبه‌کدها، می‌توانند قبل از نوشتن یک برنامه به زبان اجرای مانند C++، برنامه‌نویس را در راه صحیح قرار دهند.

مهندسی نرم‌افزار

شبه‌کد به برنامه‌نویسان کمک می‌کند تا در زمان فرآیند طراحی برنامه یک مفهوم منطقی از برنامه بدست آورند. برنامه شبه‌کد می‌تواند در مرحله بعد به زبان C++ برگردانده شود.

شبه‌کدهایی که از آنها استفاده می‌کنیم، فقط از کاراکترها تشکیل شده‌اند و برنامه‌نویس می‌تواند آنها را در یک برنامه ویرایشگر، تایپ کند. شبه‌کد، فقط شامل عبارات اجرائی است. زمانی که شبه‌کد تبدیل به کدهای C++ شود، برنامه حالت اجرایی پیدا خواهد کرد. اعلان‌ها جزء عبارات اجرائی نیستند. برای مثال، اعلان

```
int i;
```

فقط به کامپایلر نوع متغیر **i** را نشان داده و کامپایلر مکانی در حافظه به این متغیر اختصاص می‌دهد. اما این اعلان‌ها هیچ عمل اجرائی نظیر ورودی، خروجی یا یک عمل محاسباتی را در زمانی که برنامه اجرا می‌شود، از خود نشان نمی‌دهند. تعدادی از برنامه‌نویسان لیستی از متغیرها تهیه کرده و منظور از کاربرد هر متغیر را در ابتدای شبه‌کد قرار می‌دهند.

اکنون به مثالی از شبه کد نگاه می‌کنیم که می‌تواند برای کمک به برنامه‌نویس در ایجاد یک برنامه جمع معرفی شده در شکل ۵-۲ (فصل دوم) نوشته شده باشد. این شبه کد (شکل ۱-۴) متناظر با الگوریتمی است که دو عدد صحیح از کاربر دریافت، آنها را با هم جمع و مجموع آنها را به نمایش در می‌آورد. با اینکه لیست کامل شبه کد را به نمایش درآورده‌ایم، اما شما را با نحوه ایجاد یک شبه کد از صورت مسئله آشنا خواهیم کرد.

```
1  Prompt the user to enter the first integer
2  Input the first integer
3
4  Prompt the user to enter the second integer
5  Input the second integer
6
7  Add first integer and second integer
8  Display result
```

شکل ۱-٤ | شبه کد برنامه جمع شکل ۵-۲.

خطوط 2-1 متناظر با عبارات موجود در خطوط 14-13 از شکل ۵-۲ هستند. دقت کنید که عبارات شبه کد، عبارات ساده انگلیسی هستند که هر وظیفه در C++ را بیان می‌کنند. به همین ترتیب، خطوط 5-4 متناظر با عبارات موجود در خطوط 17-16 و خطوط 8-7 متناظر با عبارات موجود در خطوط 19 و 21 از شکل ۵-۲ هستند.

چندین نکته مهم در شبه کد شکل ۱-۴ وجود دارد. دقت کنید که شبه کد فقط متناظر با کد موجود در تابع **main** است، به این دلیل که معمولاً از شبه کد برای الگوریتم‌ها استفاده می‌شود، نه برای کل برنامه. در این مورد، از شبه کد برای عرضه الگوریتم استفاده شده است. تابع موجود در این کد به اندازه خود الگوریتم مهم نیست. به همین دلیل، خط 23 از شکل ۵-۲ (عبارت **return**) در شبه کد وارد نشده است. عبارت **return** در انتهای هر تابع **main** قرار دارد و در الگوریتم اهمیتی ندارد. سرانجام، خطوط 9-11 از شکل ۵-۲ در شبه کد وجود ندارد چرا که اعلان متغیرها جزء عبارات اجرایی نیستند.

٤-٤ عبارات کنترل

معمولاً، عبارات موجود در یک برنامه یکی پس از دیگری و به ترتیبی که نوشته شده‌اند اجرا می‌شوند، که به اینحالت *اجرای ترتیبی* می‌گویند. انواع متفاوتی از عبارات که بزودی درباره آنها بحث

خواهیم کرد، برنامه‌نویسان را قادر می‌سازند تا با مشخص کردن عبارتی که بایستی قبل از عبارت دیگری اجرا شود، این توالی اجرا را در دست گیرند، که به اینحالت، *کنترل انتقال* می گویند.

در دهه ۱۹۶۰، به دلیل وجود مشکلات فراوان در ایجاد نرم‌افزارهای کاربردی، استفاده از روش‌های کنترل انتقال به عنوان یک زمینه بکار گرفته شد و نشانه آن عبارت **goto** بود. این عبارت به برنامه‌نویس اجازه می‌دهد تا کنترل را به یک مکان ویژه در کل برنامه انتقال دهد. عقیده‌ای که برنامه‌نویسی ساخت یافته نام گرفته بود، تقریباً با برنامه‌نویسی حذف **goto** یا کاهش آن (goto-less) مترادف شده است.

با تحقیقات Jacopini و Bohm که نشان داد که برنامه‌ها بایستی بدون **goto** نوشته شوند، دعوت از برنامه‌نویسان به طرف برنامه‌نویسی کاهش استفاده از **goto** آغاز گردید. اما تا سال ۱۹۷۰ برنامه‌نویسی ساخت یافته جدی گرفته نشد. در نتیجه بکار بردن این روش، توسعه نرم‌افزار و کاهش بودجه‌های ساخت آن مشخص شد. کلید تمام این موفقیت‌ها برنامه‌های ساخت‌یافته‌ای بودند که هم وضوح بالاتر و خطاگیری آسانتری داشتند.

Bohm و Jacopini نشان دادند که تمام برنامه‌ها در سه عبارت کنترلی می‌توانند نوشته شوند:

- *عبارت توالی (sequence structure)*
- *عبارت انتخاب (selection structure)*
- *عبارت تکرار (repetition structure)*

ساختار توالی در C++

ساختار توالی بصورت توکار در++C وجود دارد. مگر اینکه آن را به نحوه دیگری هدایت کنید. کامپیوتر مبادرت به اجرای عبارات ++C یکی پس از دیگری و به ترتیبی که نوشته شده‌اند می کند، که این حالت اجرای ترتیبی یا متوالی است. دیاگرام فعالیت (Unified Modeling language) UML به نمایش درآمده در شکل ۲-۴ نشاندهنده یک ساختار توالی است که در آن دو محاسبه به ترتیب انجام می‌شود. زبان ++C اجازه می‌دهد تا به هر اندازه که لازم داریم از ساختار توالی استفاده کنیم. همانطوری که بزودی مشاهده خواهید کرد، در هر کجا که یک عمل می‌تواند موجود باشد، می‌توانیم چندین عمل را به صورت توالی جایگزین کنیم.

در این شکل، دو عبارت مبادرت به افزودن یک نمره به متغیر مجموع (**total**) و مقدار 1 به متغیر **counter** می کنند. چنین عباراتی را می‌توان در یک برنامه محاسبه میانگین نمره چند دانشجو مشاهده کرد. برای محاسبه میانگین، مجموع نمرات به تعداد نمرات تقسیم می‌شود. از متغیر شمارنده (**counter**) برای نگهداری تعداد نمرات وارد شده استفاده می‌شود. در بخش ۸-۴ با عبارات مشابهی مواجه خواهید شد.

دیاگرام‌های فعالیت بخشی از UML هستند. یک دیاگرام فعالیت مبادرت به مدل‌سازی *روندکار* (فعالیت) یک بخش از سیستم نرم‌افزاری می‌کند. چنین روندهایی می‌توانند در برگیرنده بخشی از یک الگوریتم، همانند یک ساختار توالی در شکل۲-۴ باشند.دیاگرام‌های فعالیت مرکب از نمادهای معنی‌دار، همانند نمادهای وضعیت عمل (یک مستطیل که گوشه‌های چپ و راست آن به سمت بیرون انحناء داده شده‌اند)، لوزی‌ها و دایره‌های کوچک است. این نمادها توسط *فلش‌های انتقال* به یکدیگر متصل می‌شوند که نشاندهنده روند فعالیت می‌باشند.

همانند شبه کد، دیاگرام‌های فعالیت به برنامه‌نویسان در توسعه و عرضه الگوریتم‌ها کمک می‌کنند، اگر چه برخی از برنامه‌نویسان شبه کد را ترجیح می‌دهند. دیاگرام‌های فعالیت به وضوح نحوه عملکرد ساختارهای کنترل را نشان می‌دهند.

به دیاگرام فعالیت ساختار متوالی در شکل ۲-۴ توجه کنید. این دیاگرام حاوی دو نماد وضعیت عمل است که نشاندهنده اعمالی هستند که اجرا می‌شوند. هر وضعیت عمل حاوی یک بیان‌کننده عمل است، "add grade to total" و "add 1 to counter" که مشخص‌کننده عملی هستند که انجام خواهند شد. سایر اعمال می‌توانند محاسباتی یا ورودی/ خروجی باشند.

شکل ۲-٤ | دیاگرام فعالیت یک ساختار توالی.

فلش‌ها در دیاگرام فعالیت، بنام فلش‌های انتقال شناخته می‌شوند. این فلش‌ها نشاندهنده *انتقال* هستند و بر این نکته دلالت دارند که ترتیب اجرای اعمال به چه صورتی است. برای مثال در شکل ۲-۴ ابتدا **grade** با **total** جمع شده و سپس **1** به **counter** افزوده می‌شود.

دایره توپر ساده که در بالای دیاگرام فعالیت قرار گرفته نشاندهنده وضعیت اولیه فعالیت است، ابتدای روندکار قبل از اینکه برنامه عملیات مدل شده را انجام دهد. دایره توپر احاطه شده با یک دایره توخالی که در پایین دیاگرام فعالیت دیده می‌شود، نشاندهنده وضعیت پایانی است، پایان روندکار پس از اینکه فعالیت‌های خود را انجام داده است.

همچنین شکل ۲-۴ شامل مستطیل‌های با گوشه‌های خم شده به داخل (سمت راست-بالا) است. این مستطیل‌ها، در UML، *نکته* (*note*) نامیده می‌شوند. نکته‌ها توضیحات اضافی در ارتباط با هدف نمادها در دیاگرام ارائه می‌کنند. از نکته‌ها می‌توان در هر دیاگرام UML و نه تنها در دیاگرام‌های فعالیت استفاده کرد. در شکل ۲-۴ از نکته‌های UML برای نمایش کد C++ مرتبط با هر وضعیت عمل در دیاگرام فعالیت استفاده شده است. خط نقطه چین مبادرت به متصل نمودن هر نکته با عنصری می‌کند که در ارتباط با آن توضیح ارائه می‌نماید. معمولاً دیاگرام‌های فعالیت، کدC++ پیاده‌سازی کننده فعالیت را عرضه نمی‌کنند.

اما از این نکته‌ها به این منظور در اینجا استفاده کرده‌ایم تا رابطه دیاگرام با کد C++ مربوطه را بهتر نشان دهیم. برای کسب اطلاعات بیشتر در مورد UML به بخش مبحث آموزشی مهندسی نرم‌افزار که در انتهای فصل ۱ الی ۷، ۹، ۱۰، ۱۲ و ۱۳ قرار دارند مراجعه کرده یا از وب سایت www.uml.org بازدید نمایید.

عبارات انتخاب در C++

زبان C++ سه نوع عبارت انتخاب تدارک دیده است که در مورد آنها در این فصل و فصل بعدی توضیح خواهیم داد. در عبارت انتخاب **if** اگر شرط برقرار باشد عبارت یا عبارات داخل بدنه اجرا شده و در صورتیکه شرط برقرار نباشد از روی آن عبارات بدون اجرای آنها عبور خواهد شد. عبارت **if..else** در صورتیکه شرط برقرار باشد، عبارت (یا دنباله‌ای از عبارات) را انجام داده و در صورتیکه شرط برقرار نباشد، عمل اجرای متفاوتی را به انجام می‌رساند (یا اجرای توالی از عبارات). عبارت **switch** که در فصل ۵ به بررسی آن خواهیم پرداخت با توجه به ارزش یک عبارت، یکی از چندین عمل اجرائی را به اجرا در می‌آورد.

عبارت **if** را *عبارت تک انتخابی* (*single-selection*) می‌نامند، چرا که یک عمل را انتخاب و اجرا یا آنرا رد می‌کند. عبارت **if..else** را *عبارت دو انتخابی* (*double-selection*) می‌نامند، چرا که انتخابی مابین دو حالت متفاوت انجام می‌دهد. عبارت **switch**، *عبارت چند انتخابی* (*multiple-selection*) نامیده می‌شود، چرا که از میان موارد متفاوت انتخاب خود را انجام می‌دهد.

عبارات تکرار در C++

C++ سه نوع عبارت تکرار بنام‌های زیر تدارک دیده است:

- **while**
- **do..while**
- **for**

عبارت تکرار **while**، در این فصل معرفی خواهد شد و عبارات **do..while**، و **for** در فصل ۵ توضیح داده خواهند شد. کلمات **if**، **else**، **switch**، **while**، **do** و **for** همگی جزء کلمات کلیدی C++ هستند (جدول شکل ۳-۴). با بسیاری از این کلمات کلیدی در این کتاب آشنا خواهید شد.

خطای برنامه‌نویسی

استفاده از کلمه کلیدی بعنوان یک شناسه خطای نحوی خواهد بود.

خطای برنامه‌نویسی

نوشتن یک کلمه کلیدی با حروف بزرگ خطای نحوی است تمام کلمات کلیدی در ++Cفقط شامل حروف کوچک هستند.

```
C++ Keywords
```

Keywords common to the C and C++ programming languages

auto	**break**	**case**	**char**	**const**
continue	**default**	**do**	**double**	**else**
enum	**extern**	**float**	**for**	**goto**
if	**int**	**long**	**register**	**return**
short	**signed**	**sizeof**	**static**	**struct**
switch	**typedef**	**union**	**unsigned**	**void**
volatile	**while**			

C++ only keywords

asm	**bool**	**catch**	**class**	**const_cast**
delete	**dynamic_cast**	**explicit**	**false**	**friend**
inline	**mutable**	**namespace**	**new**	**operator**
private	**protected**	**public**	**reinterpret_cast**	**and**
static_cast	**template**	**this**	**throw**	**true**
try	**typeid**	**typename**	**using**	**virtual**
wchar_t	**and_eq**	**bitand**	**bitor**	**export**
not	**not_eq**	**or**	**or_eq**	**xor_eq**

شکل ۳-٤ | کلمات کلیدی ++C

خلاصه‌ای بر عبارات کنترلی در ++C

++C دارای سه عبارت کنترلی است، که از این به بعد از آنها بعنوان عبارات کنترلی یاد خواهیم کرد: عبارت توالی، عبارات انتخابی (سه نوع- **if, if..else** و **switch**) و عبارات تکرار(سه نوع- **while, for** و **do..while**). هر برنامه ++C از ترکیب این عبارات کنترلی ایجاد می‌شود. همانند عبارت توالی در شکل ۲-۴، می‌توانیم هر عبارت کنترلی را بصورت یک دیاگرام فعالیت مدل‌سازی کنیم. هر دیاگرام حاوی یک حالت اولیه و یک حالت پایانی است، که به ترتیب نشاندهنده *نقطه ورودی* (*entry point*) به عبارت کنترلی و *نقطه خروجی* (*exit point*) آن می‌باشند. عبارات کنترلی تک‌ورودی/تک‌خروجی ایجاد آسانتر برنامه‌ها را ممکن می‌سازند. نقطه خروجی یک عبارت کنترلی را می‌توان به نقطه ورودی عبارت کنترلی دیگری

متصل کرد و به همین ترتیب ادامه داد. این فرآیند همانند قرار دادن بلوک‌های بر روی هم است، از اینرو این روش، *عبارت کنترلی پشته* (*control structure stacking*) نام دارد. این روش یکی از روش‌های موجود برای متصل کردن عبارات کنترلی به یکدیگر است. یک روش دیگر *عبارت کنترلی تودرتو* یا *آشیانه‌ای* (*control structure nesting*) می‌باشد که در آن یک عبارت کنترلی می‌تواند در درون عبارت دیگری قرار گیرد. بنابر این الگوریتم‌ها در برنامه‌های C++ فقط متشکل از سه نوع عبارت کنترلی ترکیب شده با این دو روش هستند.

مهندسی نرم‌افزار

*هر برنامه را می‌توان با هفت نوع عبارت کنترلی ایجاد کرد (توالی، **if..else if**، **switch**، **while**، **do..while** و **for**) که به دو روش با هم ترکیب می‌شوند (عبارت کنترلی پشه‌ای و تو در تو یا آشیانه‌ای).*

۵-٤ عبارت انتخاب if

در یک عبارت انتخاب، هدف برگزیدن یکی از گزینه‌های موجود برای انجام آن است. برای مثال، فرض کنید که شرط قبولی در یک امتحان نمره 60 است از (100). عبارت شبه‌کد آن بصورت زیر می‌باشد:

If student's grade is greater than or equal to 60
Print "Passed"

شرط "student's grade is greater than or equal to 60" می‌تواند برقرار باشد یا نباشد. اگر شرط برقرار باشد عبارت "Passed" به معنی قبول شدن به نمایش در می‌آید و عبارت پس از شبه‌کد به ترتیب اجرا می‌شود (بیاد داشته باشید که شبه‌کد یک زبان برنامه‌نویسی واقعی نیست). اگر شرط برقرار نباشد عبارت چاپ نادیده گرفته می‌شود و عبارت شبه‌کد بعدی به ترتیب اجرا خواهد شد. عبارت موجود در بدنه عبارت **if** رشته **"Passed"** را به چاپ می‌رساند. همچنین به دندانه‌دار بودن این عبارت در این عبارت انتخاب دقت کنید. دندانه گذاری امری اختیاری است، اما بکارگیری آن بسیار توصیه می‌شود چرا که ارتباط عبارتهای مختلف برنامه را بخوبی نشان می‌دهند. کامپایلر C++ کاراکترهای *whitespace* یعنی کاراکترهای فاصله، tab و خطوط جدید بکار رفته در ایجاد دندانه‌ها و فاصله‌گذاری عمودی را بجز کاراکترهای whitespace بکار رفته در رشته‌ها، در نظر نمی‌گیرد.

برنامه‌نویسی ایده‌ال

سعی کنید از روش دندانه‌گذاری ثابتی در برنامه‌های خود استفاده کنید تا خوانایی برنامه افزایش یابد.

می‌توان این عبارت شبه‌کد **if** را در زبان C++ بصورت زیر نوشت

```
if (grade >= 60 )
```

```
cout << "Passed";
```

اگر به کد C++ دقت کنید متوجه شباهت نزدیک آن با شبه‌کد خواهید شد و نقش شبه‌کد به عنوان یک ابزار توسعه برنامه بخوبی آشکار می‌شود.

در شکل ۴-۴ دیاگرام فعالیت عبارت تک‌انتخابی **if** نشان‌داده شده است. این دیاگرام حاوی یکی از مهمترین نمادها در یک دیاگرام فعالیت است. نماد لوزی یا *نماد تصمیم* نشان می‌دهد که باید در آن نقطه تصمیمی اتخاذ گردد. نماد تصمیم‌گیری بر این نکته دلالت دارد که روند کار در امتداد مسیری به کار ادامه خواهد داد که توسط نماد وابسته *نگهبان شرط* تعیین می‌شود (آیا شرط برقرار است یا خیر). هر فلش یا بردار انتقال خارج شده از یک نماد تصمیم دارای یک نگهبان شرط است (در درون براکت‌های مربعی در بالا یا کنار فلش انتقال جای می‌گیرد). اگر شرط یک نگهبان شرط برقرار باشد، روند کار وارد وضعیت عملی می‌شود که فلش انتقال به آن اشاره می‌کند. در شکل ۴-۴ اگر grade بزرگتر یا برابر 60 باشد، برنامه کلمه "Passed" را بر روی صفحه نمایش چاپ کرده و سپس انتقال به وضعیت پایانی در این فعالیت می‌رسد. اگر grade کوچکتر از 60 باشد، بلافاصله برنامه به وضعیت پایانی منتقل می‌شود، بدون اینکه پیغامی چاپ کند.

شکل ٤-٤ | دیاگرام فعالیت عبارت if.

در فصل اول آموختیم، تصمیم‌گیری می‌تواند براساس شرط‌هایی صورت گیرد که حاوی عملگرهای رابطه‌ای یا برابری هستند. در واقع، در C++ یک شرط می‌تواند بر پایه هر عبارتی ارزیابی گردد، اگر عبارت با صفر ارزیابی شود، با آن همانند *false* (عدم برقراری شرط) رفتار خواهد شد و اگر عبارت با مقداری غیر از صفر ارزیابی گردد، با آن همانند *true* (برقراری شرط) رفتار می‌شود. زبان C++ دارای نوع داده *بولی*(*bool*) برای متغیرهایی است که فقط قادر به نگهداری مقادیر **true** و **false** هستند، که هر دو جزء کلمات کلیدی در C++ می‌باشند.

قابلیت حمل

*برای حفظ سازگاری با نسخه‌های قبلی C که از اعداد صحیح برای مقادیر بولی استفاده می‌کردند، می‌توان برای عرضه یک مقدار بولی **true** از هر مقدار غیرصفری استفاده کرد (معمولاً کامپایلرها از **1** استفاده می‌کنند). برای عرضه یک مقدار بولی **false** نیز می‌توان از مقدار صفر استفاده کرد.*

دقت کنید که عبارت **if** یک عبارت *تک‌ورودی/ تک‌خروجی* است. همچنین دیاگرام‌های مابقی عبارات کنترلی نیز حاوی نمادهای وضعیت اولیه، فلش‌های انتقال، وضعیت اجرا، تصمیم‌گیری و وضعیت پایانی هستند. به این نحوه نمایش عبارات کنترلی روش مدل برنامه‌نویسی *اجرائی/تصمیم‌گیری* گفته می‌شود.

می‌توانیم هفت صندوق را تصور کنیم که هر کدام فقط حاوی دیاگرام‌های فعالیت UML خالی از یکی از هفت نوع عبارت (ساختار) کنترلی است. وظیفه برنامه‌نویس جفت‌وجور کردن برنامه با سرهمبندی کردن دیاگرام فعالیت به هر تعداد از هر نوع عبارت کنترلی تصریح شده در الگوریتم است که فقط به دو روش قابل انجام است، روش *پشته‌ای* و *تودرتو*. در ادامه وضعیت‌های عمل و تصمیم‌گیری را با عبارات اجرایی و نگهبانان شرط به روش مقتضی پر می‌کند. در ارتباط با نوشتن انواع روش‌هایی که می‌تواند در عبارات اجرایی و تصمیم‌گیری بکار گرفته شوند، صحبت خواهیم کرد.

شکل ٤-٤ | دیاگرام فعالیت عبارت تک انتخابی if.

٤-٦ عبارت انتخاب if..else

همانطوری که گفته شد عبارت انتخاب **if** فقط در صورت برقرار بودن شرط، عملی را به اجرا در می‌آورد، در غیر اینصورت از روی عبارت یا عبارات پرش می‌کند. عبارت انتخاب **if..else** این امکان را به برنامه‌نویس می‌دهد که تعیین کند چه اعمالی در برقرار بودن شرط اجرا شوند و چه اعمالی در حالت برقرار نبودن شرط به اجرا در آیند. برای مثال، در شبه‌کد زیر

If Student's grade is greater than or equal to 60
Print "Passed"
Else
Print "Failed"

اگر نمره دانش‌آموز برابر ۶۰ یا بالاتر باشد، عبارت "*Passed*" به نمایش در می‌آید و اگر کمتر از آن باشد عبارت "*Failed*". در هر دو حالت پس از انجام عمل چاپ، عبارت شبه‌کد بعدی به اجرا گذاشته خواهد شد.

عبارت شبه‌کد **if..else** مطرح شده را می‌توان در زبان C++ و به فرم زیر نوشت:

```
if (grade >= 60 )
     cout << "Passed";
else
     cout << "Failed";
```

به دندانه‌دار بودن بدنه شرط **else** دقت کنید که با خطوط بالای خود در شرط **if** یکسان قرار گرفته‌اند.

برنامه‌نویسی ایده‌ال

*دندانه‌دار نوشتن هر دو قسمت بدنه عبارت **if..else** خوانائی برنامه را افزایش می‌دهد.*

در شکل ۵-۴ روند کنترل جریان در یک عبارت (ساختار) **if..else** نشان داده شده است. مجدداً توجه کنید (در کنار وضعیت اولیه، فلش‌های انتقال و وضعیت پایانی) که نمادهای بکار رفته در این دیاگرام فعالیت عبارتند از نمادهای عمل و تصمیم‌گیری و تاکید ما بر مدل‌سازی عمل/ تصمیم‌گیری است. مجدداً

به صندوق‌های خالی از دیاگرام‌های فعالیت عبارات انتخاب دوگانه فکر کنید که برنامه‌نویس می‌تواند به روش پشته‌ای یا تودرتو با سایر دیاگرام‌های فعالیت ساختارهای کنترلی بکار گیرد تا مبادرت به پیاده‌سازی الگوریتم کند.

شکل ٥-٤ دیاگرام فعالیت عبارت دو انتخابی if..else

عملگر شرطی (:?)

زبان C++ حاوی عملگر شرطی (**:?**)، است که قرابت نزدیکی با عبارت **if..else** دارد. عملگر شرطی C++ تنها عملگر *ternary* است، به این معنی که سه عملوند دریافت می‌کند. عملوندها به همراه عملگر شرطی تشکیل *عبارت شرطی* را می‌دهند. عملوند اول نشاندهنده شرط می‌باشد، عملوند دوم مقداری است که در صورت **true** بودن شرط انتخاب می‌شود و عملوند سوم مقداری است که در صورت برقرار نبودن شرط یا **false** بودن آن انتخاب می‌شود. برای مثال عبارت زیر

```
cout << (grade >= 60 ? ″Passed″ : ″Failed″ );
```

حاوی یک عبارت شرطی، است که در صورت برقرار بودن شرط **grade >= 60** رشته **"Passed"** ارزیابی می‌شود، اما اگر شرط برقرار نباشد، رشته **"Failed"** بکار گرفته خواهد شد. از اینرو عملکرد این عبارت شرطی دقیقا همانند عملکرد عبارت **if..else** قبلی است. عملگر شرطی از تقدم پایین‌تری برخوردار است و از اینرو معمولا کل عبارت شرطی را در درون پرانتزها قرار می‌دهند.

اجتناب از خطا

برای اجتناب از مشکل اولویت و روشن شدن مطلب، سعی کنید عبارات شرطی (که در عبارات بزرگ شرکت می‌کنند) را در درون پرانتزها قرار دهید.

مقادیر موجود در یک عبارت شرطی قادر به اجرا شدن نیز هستند. برای مثال عبارت شرطی زیر مبادرت به چاپ "**Passed**" یا "**Failed**" می‌کند.

```
grade >= 60 ? cout << "Passed" : cout << "Failed";
```

این عبارت به صورت زیر تفسیر می‌شود «اگر **grade** بزرگتر یا مساوی **60** باشد، پس **cout<<"Passed"**، در غیر اینصورت **cout<<"Faild"**.» همچنین این عبارت قابل مقایسه با عبارت if..else قبلی است. عبارات شرطی را می‌توان در مکان‌های از برنامه که امکان استفاده از **if..else** وجود ندارد، بکار گرفت.

عبارات تودرتوی if..else

عبارت تودرتوی **if..else** برای تست چندین شرط با قرار دادن عبارتهای **if..else** در درون عبارتهای **if..else** دیگر است. برای مثال، عبارت شبه‌کد زیر، حرف "A" را برای نمره‌های بزرگتر یا برابر 90، "B" را برای نمره‌های در محدودهٔ 89-80، "C" را برای نمره‌های در محدودهٔ 79-70، "D" را برای نمره‌های در محدودهٔ 69-60 و "F" را سایر نمرات به چاپ می‌رساند.

```
If student's grade is greater then or equal to 90
        Print "A"
Else
        If student's grade is greater than or equal to 80
                Print "B"
        Else
                If student's grade is greater than or equal to 70
                        Print "C"
                Else
                        If student's grade is greater than or equal to 60
                                Print "D"
                Else
                                Print "F"
```

عبارت شبه‌کد بالا را می‌توان در زبان C++ و به فرم زیر نوشت:

```
if (studentGrade>=90) // 90 and above gets "A"
    cout << "A";
else
   if (studentGrade>=80) // 80-89 gets "B"
       cout << "B";
   else
      if (strudentGrade>=70) // 70-79 gets "C"
         cout << "C";
      else
         if (studentGrade>=60) // 60-69 gets "D"
            cout << "D";
         else // less than 60 gets "F"
            cout << "F";
```

اگر مقدار **studentGrade** بزرگتر یا مساوی 90 باشد، اولین شرط از پنج شرط برقرار شده و فقط عبارت **cout** قرار گرفته در بدنه اولین شرط به اجرا در می‌آید. پس از اجرای این عبارت از بخش **else** خارجی عبارت **if..else** عبور خواهد شد.

برنامه‌نویسی ایده‌ال

قبل و بعد از هر عبارت کنترل یک خط خالی قرار دهید تا بتوان عبارتهای کنترل را از سایر نقاط برنامه تشخیص داد.

اکثر برنامه‌نویسان C++ ترجیح می‌دهند که عبارت **if..else** را با استفاده از کلمه کلیدی **else if** و بصورت زیر در برنامه‌های خود بنویسند:

```
if (studentGrade>=90) // 90 and above gets "A"
     cout << "A";
else if (studentGrade>=80) // 80-89 gets "B"
     cout << "B";
else if (studentGrade>=70) // 70-79 gets "C"
     cout << "C";
else if (studentGrade>=60) // 60-69 gets "D"
     cout << "D";
else // less than 60 gets "F"
     cout << "F";
```

هر دو حالت معادل یکدیگرند، اما نوع آخر در نزد برنامه‌نویسان از محبوبیت بیشتری برخوردار است. چرا که از دندانه‌دار کردن عمیق کد به طرف راست اجتناب می‌شود.

کارائی

*یک عبارت **if..else** تودرتو می‌تواند بسیار سریعتر از یک سری عبارت **if** عمل کند، چرا که احتمال دارد شرطی در ابتدای عبارت **if..else** برقرار شود و کنترل برنامه زودتر از این قسمت خارج گردد.*

کارائی

*در یک عبارت **if..else** تودرتو، شرط‌هایی که امکان برقرار شدن **(true)** آنها بیشتر است در ابتدای عبارت **if..else** تودرتو قرار دهید. در اینحالت امکان اجرای سریعتر عبارت **if..else** فراهم می‌آید.*

مشکل dangling-else

همیشه کامپایلر C++ یک **else** را با یک **if** در نظر می‌گیرد، مگر اینکه خلاف آنرا با استفاده از براکت‌ها مشخص کنید. به این مشکل *dangling-else* می‌گویند. برای مثال،

```
if ( x > 5 )
   if ( y > 5 )
       cout << "x and y are > 5";
else
   cout << "x is <=5";
```

به نظر می‌رسد بر این نکته دلالت دارد که اگر **x** بزرگتر از 5 باشد، عبارت **if** تودرتو تعیین می‌کند که آیا **y** نیز بزرگتر از 5 است یا خیر. اگر چنین باشد، رشته "x and y are > 5" در خروجی چاپ می‌شود. در غیر اینصورت اگر **x** بزرگتر از 5 نباشد، بخش **else** از عبارت **if..else** رشته "x is <=5" را چاپ خواهد کرد.

با این همه امکان دارد عبارت **if** تودرتوی فوق مطابق با انتظار کار نکند. تفسیر کامپایلر از عبارت بصورت زیر خواهد بود

```
if ( x > 5 )
   if ( y > 5 )
      cout << "x and y are > 5";
   else
  cout << "x is <=5";
```

که در آن بدنه اولین عبارت **if** یک عبارت **if..else** تودرتو است. این عبارت مبادرت به تست بزرگتر بودن **x** از 5 می‌کند. اگر چنین باشد، اجرا با تست **y** بزرگتر از 5 ادامه می‌یابد. اگر شرط دوم برقرار باشد، رشته "x and y are >5" به نمایش در خواهد آمد. با این همه اگر شرط دوم برقرار نباشد، رشته "x is <=5" به نمایش در می‌آید، حتی اگر بدانیم که **x** بزرگتر از 5 است.

برای اینکه عبارت فوق بنحوی کار کند که از انتظار داریم، بایستی کل عبارت بصورت زیر نوشته شود:

```
if ( x > 5 )
{
   if ( y > 5 )
      cout << "x and y are > 5";
}
   else
  cout << "x is <=5";
```

براکت‌ها به کامپایلر نشان می‌دهند که دومین **if** در بدنه اولین **if** قرار دارد و **else** در ارتباط با اولین **if** می‌باشد.

بلوک‌ها

معمولا عبارت انتخاب **if** فقط منتظر یک عبارت در بدنه خود است. به همین ترتیب، هر یک از بخش‌های **else** و **if** در یک عبارت **if..else** انتظار مقابله با یک عبارت در بدنه خود را دارند. برای وارد کردن چندین عبارت در بدنه یک **if** یا در بخش‌های **if..else** ، عبارات را در درون براکت‌ها (**{ }**) قرار دهید. به مجموعه‌ای از عبارات موجود در درون یک جفت براکت، *بلوک* می‌گویند.

مهندسی نرم‌افزار

یک بلوک می‌تواند در هر کجای برنامه که یک عبارت منفرد می‌تواند در آنجا قرار داده شود، جای گیرد.

مثال زیر شامل یک بلوک در بخشی از **else** یک عبارت **if..else** است.

```
if (studentGrade>=60)
    cout << "Passed.\n";
else
{
   cout << "Failed.\n";
   cout << "You must take this course again.\n"
}
```

در این مورد، اگر **studentGrade** کمتر از 60 باشد، برنامه هر دو عبارت موجود در بدنه **else** را اجرا کرده و پیغام‌های زیر را چاپ می‌کند.

```
Failed
You must take this course agein.
```

به براکت‌های احاطه‌کننده دو عبارت در ضابطه **else** دقت کنید. این براکت‌ها مهم هستند. بدون این براکت‌ها، عبارت

```
cout << "You must take this course again.\n";
```

در خارج از بدنه بخش **else** قرار می‌گیرد و صرفنظر از اینکه شرط برقرار باشد یا خیر، اجرا خواهد شد. این مثال نمونه‌ای از یک خطای منطقی است.

خطای برنامه‌نویسی

فراموش کردن یک یا هر دو براکت که تعیین محدودهٔ یک بلوک هستند، می‌تواند موجب خطای نحوی یا منطقی در برنامه شود.

برنامه‌نویسی ایده‌ال

*همیشه از براکت‌ها در عبارت **if..else** (یا هر عبارت کنترلی) استفاده کنید تا جلوی حوادث ناخواسته گرفته شود، بویژه به هنگام افزودن عباراتی به یک **if** یا ضابطه **else** در ادامه کار. برای اینکه جلوی فراموش‌کاری گرفته شود، برخی از برنامه‌نویسان در همان ابتدای کار و قبل از اینکه حتی عبارتی تایپ کرده باشد، براکت‌های شروع و پایین را تایپ می‌کنند و سپس عبارات مورد نظر را در درون آنها قرار می‌دهد.*

همانند یک بلوک که می‌تواند در هر کجای که یک عبارت منفرد وجود دارد جایگزین گردد، همچنین امکان داشتن *عبارت null* (یا عبارت تهی) وجود دارد. عبارت تهی با جایگزین کردن یک سیمکولن (;) بجای یک عبارت مشخص می‌شود.

خطای برنامه‌نویسی

*قرار دادن یک سیمکولن بلافاصله پس از شرط در یک عبارت **if** موجب رخ دادن خطای منطقی در عبارات **if** تک انتخابی و خطای نحوی در عبارات **if..else** دو انتخابی خواهد شد.*

۷-٤ عبارت تکرار while

یک عبارت تکرار به برنامه‌نویس امکان می‌دهد تا بر مبنای برقرار بودن یا نبودن مقداری در یک شرط، یک عمل را چندین بار و به تکرار انجام دهد. عبارت شبه‌کد زیر یک فرآیند تکرار شوند را در حین خرید نشان می‌دهد:

While there are more items on my shopping list
Purchase next item and cross it off my list

شرط "there are more items on my shopping list" ممکن است برقرار یا برقرار نباشد. اگر شرط برقرار باشد عمل "Purchase next item" و "Cross it off my list" به ترتیب اجرا خواهند شد. این عمل می‌تواند تا زمانی که شرط برقرار است انجام شود. عبارت موجود در ساختار تکرار **while** تشکیل دهنده

بدنه **while** است. سرانجام، زمانیکه شرط برقرار نباشد، تکرار پایان یافته و اولین دستور قرار گرفته پس از عبارت تکرار به اجرا در می‌آید.

مثالی که در زیر آورده شده از عبارت **while** استفاده کرده و اولین توان 3 بزرگتر از 100 را پیدا می‌کند. در ابتدای کار متغیر **product** با **3** مقدار دهی اولیه شده است:

```
int product = 3;

while (product <= 100)
   product = product * 3;
```

هنگامی که برنامه وارد عبارت **while** می‌شود، مقدار متغیر **product** برابر 3 است. متغیر **product** بصورت مکرر در 3 ضرب می‌شود و مقادیر 9، 27، 81 و 243 بدست می‌آیند. زمانیکه مقدار **product** برابر 243 شود، شرط **product<=100** در عبارت **while** برقرار نخواهد شد. در چنین حالتی تکرار با مقدار 243 برای **product** پایان می‌پذیرد. اجرای برنامه با عبارت بعد از **while** دنبال می‌شود. [نکته: اگر شرط یک عبارت **while** در همان ابتدا برقرار نباشد، عبارات قرار گرفته در بدنه عبارت تکرار اجرا نخواهند شد.]

خطای برنامه‌نویسی

شرط عبارت تکرار را به نوعی تنظیم نمائید که برنامه برای همیشه در داخل حلقه قرار نگیرد، در غیر اینصورت برنامه در حلقه بی‌نهایت "infinite loop" گرفتار می‌شود.

دیاگرام فعالیت UML به نمایش درآمده در شکل ۶-۴ نشاندهنده روند کنترلی است که متناظر با عبارت **while** مطرح شده در قسمت فوق می‌باشد. مجدداً نمادهای موجود در این دیاگرام نشاندهنده یک وضعیت عمل و یک تصمیم‌گیری هستند. همچنین این دیاگرام مبادرت به معرفی *نماد ادغام* UML کرده است، که دو روند یا جریان فعالیت را به یک فعالیت متصل می‌کند. UML نماد ادغام و تصمیم‌گیری را بصورت لوزی نشان می‌دهد. در این دیاگرام، نماد ادغام مبادرت به پیوند انتقال‌ها از وضعیت اولیه و از وضعیت عمل کرده است، از اینرو هر دو جریان وارد بخش تصمیم شده‌اند که تعیین می‌کند آیا حلقه مجدداً باید تکرار شود یا خیر. می‌توان نمادهای تصمیم و ادغام را توسط تعداد فلش‌های انتقال «واردشونده» و «خارج‌شونده» تشخیص داد. نماد تصمیم دارای یک فلش انتقال اشاره‌کننده به لوزی داشته و دارای دو یا چند فلش انتقال اشاره‌کننده به خارج از لوزی است که نشاندهنده انتقالات ممکنه از این نقطه هستند. علاوه بر این، هر فلش انتقال اشاره‌کننده به خارج از نماد تصمیم دارای یک نگهبان شرط در کنار خود است. در طرف دیگر، نماد ادغام قرار دارد که دارای دو یا چند فلش انتقال اشاره‌کننده به لوزی است و فقط یک فلش انتقال از آن خارج می‌شود و نشان می‌هد که چندین روند با یکدیگر برای انجام

فعالیت ادغام شده‌اند. توجه کنید، که برخلاف نماد تصمیم، نماد ادغام دارای یک رونوشت در کد C++ نیست.

دیاگرام شکل ۶-۴ بوضوح عملیات تکرار در عبارت **while** مطرح شده در ابتدای این بخش را نشان می‌دهد. فلش انتقال از وضعیت عمل به حالت ادغام اشاره می‌کند، که انتقال را به تصمیم باز می‌گرداند تا تستی دایر بر اینکه آیا حلقه دوباره باید صورت گیرد یا خیر انجام دهد، این حلقه زمانی شکسته می‌شود که شرط **product>100** برقرار گردد. پس از خاتمه عملیات **while**، کنترل به عبارت بعدی در برنامه انتقال می‌یابد (در این مورد وضعیت پایانی).

می‌توانید دیاگرام‌های فعالیت UML خالی عبارت تکرار **while** را تصور کنید که برنامه‌نویسان می‌توانند هر تعداد از آنها را به روش پشته‌ای یا تودرتو با سایر دیاگرام‌های فعالیت عبارات کنترلی بکار گیرند تا بخشی از الگوریتم را پیاده‌سازی کنند.

شکل ٦-٤ | دیاگرام فعالیت UML عبارت تکرار while.

٤-٨ فرموله کردن الگوریتم: شمارنده-کنترل تکرار

برای اینکه با نحوه توسعه الگوریتم‌ها آشنا شوید، مسئله بدست آوردن میانگین نمرات یک کلاس را با روش‌های مختلف بررسی می‌کنیم. صورت مسئله عبارت است از:

از کلاسی با ده دانش‌آموز آزمونی بعمل آمده است. نمرات این آزمون در اختیار شما قرار دارد (نمرات در محدودهٔ 0 تا 100 هستند). مجموع نمرات دانش‌آموزان و میانگین نمرات این کلاس را بدست آورید.

میانگین کلاس عبارت است از مجموع نمرات تقسیم بر تعداد دانش‌آموزان. الگوریتم بکار رفته بر روی کامپیوتر به منظور حل این مسئله بایستی تک تک نمرات را به عنوان ورودی دریافت کرده، محاسبه میانگین را انجام داده و نتیجه را به نمایش در آورد.

الگوریتم شبه‌کد با شمارنده-کنترل تکرار

اجازه دهید تا از شبه‌کد استفاده کرده و لیستی از فعالیت‌های اجرائی تهیه و ترتیب اجرا را مشخص سازیم. از روش *شمارنده-کنترل تکرار* برای دریافت تک تک نمرات بعنوان ورودی استفاده می‌کنیم. در این تکنیک از متغیری بنام *شمارنده* (*counter*) برای تعیین تعداد دفعات مجموعه‌ای از عبارات که اجرا خواهند شد، استفاده می‌شود. روش شمارنده-کنترل تکرار، روش *تکرار تعریف شده* نیز نامیده می‌شود چرا که تعداد تکرار قبل از اینکه حلقه آغاز شود، مشخص است. در این مثال، اجرای حلقه با رسیدن شمارنده به 10 خاتمه می‌یابد. در این بخش به معرفی الگوریتم شبه‌کد (شکل ۷-۴) و نسخه‌ای از کلاس

GradeBook (شکل‌های ۸-۴ و ۹-۴) می‌پردازیم که الگوریتم را توسط یک تابع عضو C++ پیاده‌سازی می‌کند. در بخش ۹-۴ با توسعه الگوریتم‌ها با استفاده از شبه‌کد آشنا خواهید شد.

به نقش *total* و *counter* در الگوریتم شبه‌کد دقت کنید (شکل ۷-۴). در واقع *total* متغیری است که از آن برای محاسبه مجموع مقادیر استفاده می‌شود و *counter* متغیری است که نقش شمارنده بر عهده دارد، در این برنامه، شمارنده تعداد نمرات وارد شدهٔ توسط کاربر را ثبت می‌کند.

Set total to zero
Set grade counter to one
While grade counter is less than or equal to 10
    Input the next grade
    Add the grade to the total
    Add one to the grade counter
Set the class average to the total divided by 10
Print the class average

شکل۷-٤ | الگوریتم شبه‌کد با استفاده از روش شمارنده-کنترل تکرار برای حل مسئله میانگین کلاس.

```
// Fig. 4.8: GradeBook.h
// Definition of class GradeBook that determines a class average.
// Member functions are defined in GradeBook.cpp
 include <string> // program uses C++ standard string class
using std::string;

// GradeBook class definition
class GradeBook
{
public:
   GradeBook( string ); // constructor initializes course name
   void setCourseName( string ); // function to set the course name
   string getCourseName(); // function to retrieve the course name
   void displayMessage(); // display a welcome message
   void determineClassAverage(); // averages grades entered by the user
private:
   string courseName; // course name for this GradeBook
}; // end class GradeBook
```

شکل۸-٤ | مسئله میانگین کلاس با استفاده از روش شمارنده-کنترل تکرار:سرآیند فایل GradeBook.

```
// Fig. 4.9: GradeBook.cpp
// Member-function definitions for class GradeBook that solves the
// class average program with counter-controlled repetition.
 include <iostream>
using std::cout;
using std::cin;
using std::endl;

 include "GradeBook.h" // include definition of class GradeBook

// constructor initializes courseName with string supplied as argument
GradeBook::GradeBook( string name )
{
   setCourseName( name ); // validate and store courseName
} // end GradeBook constructor

// function to set the course name;
// ensures that the course name has at most 25 characters
void GradeBook::setCourseName( string name )
{
   if ( name.length() <= 25 ) // if name has 25 or fewer characters
```

```
22       courseName = name; // store the course name in the object
23    else // if name is longer than 25 characters
24    { // set courseName to first 25 characters of parameter name
25       courseName = name.substr( 0, 25 ); // select first 25 characters
26      cout << "Name \"" << name << "\" exceeds maximum length (25).\n"
27          << "Limiting courseName to first 25 characters.\n" << endl;
28    } // end if...else
29 } // end function setCourseName
30
31 // function to retrieve the course name
32 string GradeBook::getCourseName()
33 {
34    return courseName;
35 } // end function getCourseName
36
37 // display a welcome message to the GradeBook user
38 void GradeBook::displayMessage()
39 {
40    cout << "Welcome to the grade book for\n" << getCourseName() << "!\n"
41       << endl;
42 } // end function displayMessage
43
44 // determine class average based on 10 grades entered by user
45 void GradeBook::determineClassAverage()
46 {
47    int total; // sum of grades entered by user
48    int gradeCounter; // number of the grade to be entered next
49    int grade; // grade value entered by user
50    int average; // average of grades
51
52    // initialization phase
53    total = 0; // initialize total
54    gradeCounter = 1; // initialize loop counter
55
56    // processing phase
57    while ( gradeCounter <= 10 ) // loop 10 times
58    {
59       cout << "Enter grade: "; // prompt for input
60       cin >> grade; // input next grade
61       total = total + grade; // add grade to total
62       gradeCounter = gradeCounter + 1; // increment counter by 1
63    } // end while
64
65    // termination phase
66    average = total / 10; // integer division yields integer result
67
68    // display total and average of grades
69    cout << "\nTotal of all 10 grades is " << total << endl;
70    cout << "Class average is " << average << endl;
71 } // end function determineClassAverage
```

شکل۹-٤ | مسئله میانگین کلاس با استفاده از روش شمارنده-کنترل تکرار:کد منبع فایل GradeBook.

افزایش قابلیت اعتبارسنجی GradeBook

قبل از اینکه به بحث پیاده‌سازی الگوریتم میانگین کلاس بپردازیم، اجازه دهید به بهبود کارائی انجام گرفته بر روی کلاس **GradeBook** توجه کنیم. در برنامه ۱۶-۳، تابع **setCourseName** مبادرت به اعتبارسنجی نام دوره با تست طول نام دور می‌کرد، که باید کمتر یا برابر 25 کاراکتر باشد (با استفاده از یک عبارت **if**). اگر شرط برقرار بود، نام دوره بکار گرفته می‌شد. سپس این کد با یک عبارت **if** دیگر دنبال می‌شد که مبادرت به تست طول نام دوره می‌کرد که آیا بزرگتر از 25 کاراکتر است یا خیر. دقت کنید که شرط عبارت **if** دوم کاملاً متضاد شرط **if** اول است. اگر شرطی با **true** ارزیابی گردد، بایستی

شرط‌های دیگر با **false** ارزیابی شوند. پیاده‌سازی چنین وضعیتی توسط عبارت **if..else** بهتر خواهد بود، از اینرو کد خود را با جایگزین کردن دو عبارت **if** با یک عبارت **if..else** اصلاح کرده‌ایم (خطوط 28-21 از برنامه شکل ۹-۴).

```
1  // Fig. 4.10: fig04_10.cpp
2  // Create GradeBook object and invoke its determineClassAverage function.
3   include "GradeBook.h" // include definition of class GradeBook
4
5  int main()
6  {
7     // create GradeBook object myGradeBook and
8     // pass course name to constructor
9     GradeBook myGradeBook( "CS101 C++ Programming" );
10
11    myGradeBook.displayMessage(); // display welcome message
12    myGradeBook.determineClassAverage(); // find average of 10 grades
13    return 0; // indicate successful termination
14 } // end main
```

```
Welcome to the grade book for
CS101 C++ Programming
  Enter grade: 67
  Enter grade: 78
  Enter grade: 89
  Enter grade: 67
  Enter grade: 87
  Enter grade: 98
  Enter grade: 93
  Enter grade: 85
  Enter grade: 82
  Enter grade: 100

  Total of all 10 grade is 846
  class average is 84
```

شکل ۱۰-٤| برنامه میانگین کلاس با شمارنده-کنترل تکرار:ایجاد یک شی از کلاس GradeBook (شکل ۸-٤ و ۹-٤) و فراخوانی تابع عضو determineClassAverage.

پیاده‌سازی شمارنده-کنترل تکرار در کلاس GradeBook

کلاس **GradeBook** (شکل ۸-۴ و ۹-۴) حاوی یک سازنده (اعلان شده در خط 11 از شکل ۸-۴ و تعریف شده در خطوط 15-12 از شکل ۹-۴) است که مبادرت به تخصیص مقداری به متغیر نمونه کلاس **courseName** می‌کند (اعلان شده در خط 17 از شکل ۸-۴). در خطوط 29-19، 35-32 و 42-38 از شکل ۹-۴ توابع عضو **setCourseName**، **getCourseName** و **displayMessage** تعریف شده‌اند. در خطوط 71-45 تابع عضو **determineClassAverage** تعریف شده است که پیاده سازی‌کننده الگوریتم میانگین کلاس توضیح داده شده در شبه کد شکل ۷-۴ است.

در خطوط 50-47 متغیرهای محلی **total**، **gradeCounter**، **grade** و **average** از نوع **int** اعلان شده‌اند. در متغیر **grade** ورودی کاربر ذخیره می‌شود. توجه کنید که اعلان‌های فوق در بدنه تابع عضو **determineClassAverage** قرار دارند.

در نسخه‌های کلاس **GradeBook** مطرح شده در این فصل، فرآیند خواندن و پردازش نمرات به روش ساده‌ای در نظر گرفته شده‌اند. محاسبه میانگین در تابع عضو **determineClassAverage** و با استفاده از متغیرهای محلی صورت می‌گیرد. در این بخش مبادرت به ذخیره‌سازی نمرات دانشجویان نمی‌کنیم. در فصل هفتم، با تغییری که در کلاس **GradeBook** انجام می‌دهیم قادر به نگهداری نمرات در حافظه خواهیم بود که توسط ساختمان داده *آرایه* صورت می‌گیرد. در اینحالت به یک شی **GradeBook** اجازه داده می‌شود تا محاسبات مختلف را بر روی همان مجموعه از نمرات انجام دهد بدون اینکه کاربر مجبور به وارد کردن همان نمرات به دفعات باشد.

برنامه‌نویسی ایده‌ال

همیشه یک خط خالی مابین بخش اعلان‌ها و عبارات اجرایی قرار دهید. در این صورت بخش اعلان بخوبی در برنامه مشخص شده و خوانایی برنامه افزایش می‌یابد.

در خطوط 54-53 متغیر **total** با **0** و **gradeCounter** با **1** مقداردهی اولیه شده‌اند.دقت کنید که متغیرهای **totoal** و **gradeCounter** قبل از اینکه در محاسبات بکار گرفته شوند، مقداردهی اولیه شده‌اند. معمولاً متغیرهای شمارنده را با یک یا صفر و بر اساس نیاز مقداردهی اولیه می‌کنند. یک متغیر مقداردهی نشده حاوی یک مقدار اشغال (یا مقدار تعریف نشده) است، آخرین مقداری که در مکان حافظه رزرو شده برای متغیر از قبل وجود داشته است. در این برنامه متغیرهای **grade** و **average** که مقدار خود را از طرف ورودی کاربر و محاسبه میانگین بدست می‌آورند، نیازی به مقداردهی اولیه ندارند.

خط 57 مشخص می‌کند که عبارت **while** تا زمانیکه مقدار **gradeCounter** کمتر یا معادل **10** باشد، تکرار خواهد شد. تا زمانیکه شرط برقرار باشد، ساختار **while** عبارات قرار گرفته مابین براکت‌های بدنه خود را تکرار خواهد کرد.

عبارت بکار رفته در خط 59، جمله **"Enter grade:"** را بنمایش در می‌آورد. این خط معادل عبارت شبه‌کد "*Prompt the user to enter the next grade.*" هستند. خط 60 مقدار وارد شده توسط کاربر را خوانده و آنرا در متغیر **grade** ذخیره می‌کند. این خط معادل شبه‌کد "*Input the next grade.*" است. بخاطر دارید که متغیر **grade** در ابتدای برنامه مقداردهی اولیه نشده است، به این دلیل که برنامه مقدار **grade** را از کاربر و در هر بار تکرار حلقه اخذ می‌کند. سپس، برنامه مقدار **total** را با مقدار جدید **grade** که

توسط کاربر وارد شده به روز می‌کند (خط **61**). مقدار **grade** با مقدار قبلی **total** جمع شده و نتیجه به **total** تخصیص می‌یابد.

در خط 62 متغیر **gradeCounter** یک واحد افزایش می‌یابد تا نشان دهد یک نمره مورد پردازش قرار گرفته است. اینکار تا زمانیکه شرط موجود در عبارت **while** برقرار نشود، ادامه می‌یابد. پس از اتمام حلقه، در خط 66 نتیجه محاسبه میانگین به متغیر **average** تخصیص می‌یابد. خط **69** پیغام **"Total of all 10 grades is"** و بدنبال آن مقدار متغیر **total** را بنمایش در می‌آورد. سپس در خط 70 پیغامی حاوی رشته **"Class average is"** که بدنبال آن مقدار متغیر **average** آورده شده، به نمایش در می‌آید. تابع عضو **determineClassAverage**، کنترل را به تابع فراخوان برگشت می‌دهد (تابع **main** در شکل ۱۰-۴).

توصیف کلاس GradeBook

شکل ۱۰-۴ حاوی تابع **main** این برنامه است، که یک شی از کلاس **GradeBook** ایجاد و به توصیف قابلیت‌های آن می‌پردازد. در خط 9 از شکل۱۰-۴ یک شی جدید از **GradeBook** بنام **myGradeBook** ایجاد می‌شود. رشته موجود در خط 9 به سازنده **GradeBook** ارسال می‌شود (خطوط ۱۵-۱۲ از شکل ۹-۴). خط 11 از شکل۱۰-۴ تابع عضو **displayMessage** را برای نمایش پیغام خوش‌آمدگویی به کاربر فراخوانی می‌کند. سپس خط 12 تابع عضو **determineClassAverage** را فراخوانی می‌کند تا کاربر بتواند 10 نمره را وارد کرده و سپس میانگین را محاسبه و چاپ می‌کند. تابع عضو، الگوریتم نشان داده شده در شبه کد شکل ۷-۴ را انجام می‌دهد.

نکاتی در ارتباط با تقسیم صحیح و قطع کردن

محاسبه میانگین توسط تابع عضو **determineClassAverage** صورت می‌گیرد که در واکنش به فراخوانی تابع در خط 12 از شکل۱۰-۴ فعال شده و یک عدد صحیح تولید می‌کند. خروجی برنامه نشان می‌دهد که مجموع نمرات در اجرای نمونه برنامه 846 است که به هنگام تقسیم بر 10، باید 84.6 بدست آید، عددی با نقطه اعشار. با این همه، در نتیجه محاسبه **total/10** عدد 84 بدست آمده است (خط 66 از شکل ۹-۴)، چرا که **total** و 10 هر دو مقادیر عددی صحیح هستند. نتیجه تقسیم دو عدد صحیح یک عدد صحیح است که در آن بخش اعشاری بدست آمده از تقسیم حذف می‌گردد (قطع می‌شود). در بخش بعد با نحوه بدست آوردن نتایج اعشاری از محاسبات آشنا خواهید شد.

خطای برنامه‌نویسی

فرض اینکه تقسیم صحیح مبادرت به گرد کردن (بجای قطع کردن) می‌کند می‌تواند نتایج اشتباهی

بدنبال داشته باشد. برای مثال، 4÷7 حاصل 1.75 را در ریاضی بدست می‌دهد، در حالیکه در یک تقسیم صحیح 1 کوتاه شده و 2 در حالت گرد شده تولید می‌کند.

در برنامه شکل ۹-۴، اگر در خط 66 از **gradeCounter** بجای 10 در محاسبه استفاده شود، خروجی این برنامه مقدار اشتباه 76 را نشان خواهد داد. دلیل اینکار در آخرین تکرار عبارت **while** نهفته است که **gradeCounter** به مقدار 11 در خط 62 افزایش یافته است.

خطای برنامه‌نویسی

استفاده از متغیر شمارنده حلقه، در یک عبارت محاسباتی پس از حلقه، معمولاً سبب تولید خطای منطقی بنام off-by-one-error می‌شود.

۹-٤ فرموله کردن الگوریتم‌ها: مراقبت-کنترل تکرار

اجازه دهید تا به مسئله میانگین کلاس بازگردیم و آنرا مجدداً و اینبار بصورت زیر و کلی‌تر تعریف کنیم:

"برنامه محاسبه میانگین کلاس را به نحوی توسعه دهید تا در هر بار اجرای برنامه، به تعداد اختیاری نمره دریافت کرده و محاسبه میانگین بر روی آنها اعمال شود."

در برنامه قبلی، تعداد نمرات از همان ابتدا مشخص بود (10 نمره). در این برنامه، تعداد نمراتی که بعنوان ورودی وارد خواهند شد مشخص نیستند. برنامه باید بر روی تعداد نمرات وارد شده کار کند. چگونه برنامه تشخیص می‌دهد که به گرفتن نمره پایان دهد؟ محاسبات به چه صورتی باید انجام گرفته و میانگین کلاس به نمایش درآید؟

یک راه‌حل برای رفع این مشکل، استفاده از یک مقدار ویژه بنام *مقدار مراقبتی* (*sentinel value*) است که پایان ورود داده‌ها را مشخص می‌کند (همچنین به این مقدار، *مقدار سیگنال*، *مقدار ساختگی* یا *پرچم* نیز می‌گویند). در این روش کاربر اقدام به وارد کردن نمره‌ها کرده و در پایان مقدار مراقبتی تعیین شده را به عنوان اینکه داده‌های ورودی به اتمام رسیده‌اند، وارد می‌سازد. روش مراقبت-کنترل تکرار، روش *تکرار-تعریف‌نشده* نیز نامیده می‌شود چرا که تعداد دفعات تکرار قبل از اجرای حلقه مشخص نیست.

واضح است که مقدار مراقبتی باید به نحوی انتخاب شود که به عنوان یک ورودی معتبر مورد قبول واقع نشود. بدلیل اینکه نمرات امتحان معمولاً منفی نیستند، می‌توانیم از مقدار 1- به عنوان مقدار مراقبتی در این برنامه استفاده کنیم. بنابراین به هنگام اجرای برنامه، نمرات کلاس، می‌توانند ترتیبی مانند 1- و 84، 74، 75، 96، 93 داشته باشند. برنامه باید نمره میانگین کلاس را با استفاده از مقادیر 93، 96، 75، 74 و 84 محاسبه کرده و به نمایش درآورد (1- یک مقدار مراقبتی است و نباید در محاسبه میانگین وارد شود).

خطای برنامه‌نویسی

انتخاب یک مقدار مراقبتی به عنوان یک داده معتبر، موجب رخ دادن خطای منطقی می‌شود.

الگوریتم شبه‌کد به روش مراقبت کنترل تکرار به روش از بالا به پایین، اصلاح گام‌به‌گام: اولین اصلاح

به هنگام بررسی مسائل پیچیده‌ای همانند این برنامه، عرضه الگوریتم شبه‌کد به آسانی امکان پذیر نمی‌باشد. از اینرو به برنامه میانگین کلاس با استفاده از تکنیکی بنام، از بالا به پایین، اصلاح گام‌به‌گام نزدیک می‌شویم که برای ساخت و توسعه برنامه‌های ساخت‌یافته مناسب و ضروری است. شبه‌کدی که در بالاترین سطح (*top*) ارائه می‌شود، عبارت است از:

Determine the class average for the quiz.

این عبارت تابع و هدف اصلی برنامه است که در واقع کاری که باید برنامه انجام دهد را در بردارد. عبارت top جزئیات ناکافی در مورد اینکه برنامه چگونه بایستی نوشته شود در خود دارد. بنابر این به طرف جزئیات برنامه و اصلاح گام به گام پیش می‌رویم. ابتدا عبارت top به قسمت‌های کوچکی تقسیم می‌شود که هر یک به ترتیب وظایفی در برنامه ایفا می‌کنند. نتیجه این تقسیمات در اولین گام می‌تواند چنین باشد:

Initialize Variables
Input, sum and count the quiz grades
Calculate and print the total of all student grades and the class average

در اینجا، با توجه به اینکه فقط از عبارت توالی استفاده شده، لیست انجام مراحل فقط شامل عبارت‌های اجرائی است که به ترتیب یکی پس از دیگری اجرا می‌شوند.

اصلاح گام‌به‌گام مرحله دوم

مرحله بعدی تجزیه برنامه به جزئیات بیشتر (مرحله دوم)، در ارتباط با متغیرها می‌باشد به یک متغیر بنام **total** نیاز است که مجموع اعداد را در خود نگهداری کند و به یک متغیر دیگر بنام **count** که نشان دهد، چه تعدادی از این اعداد مورد پردازش قرار گرفته‌اند. یک متغیر برای دریافت هر نمره از طریق ورودی و یک متغیر برای نگهداری میانگین محاسبه شده مورد نیاز است. عبارت شبه‌کد:

Initialize variable

را می‌توان به عبارات جزئی‌تر زیر تقسیم کرد:

Initialize total to zero
Initialize counter to zero

توجه کنید که فقط متغیرهای **total** و **counter** نیاز به مقداردهی اولیه قبل از بکارگیری دارند. متغیرهای **average** و **grade** (این متغیرها برای محاسبه میانگین و ورودی کاربر استفاده شده است)، نیازی به مقداردهی اولیه ندارند. عبارت شبه‌کد:

Input, sum and count the quiz grades

نیازمند یک عبارت تکرار (حلقه) است که نمرات را دریافت کند. چون بطور دقیق نمی‌دانیم که چه تعداد نمره به عنوان ورودی دریافت خواهیم کرد، از روش مراقبت-کنترل تکرار استفاده می‌کنیم. کاربر در هر زمان یک مقدار معتبر وارد می‌کند و پس از اینکه آخرین مقدار مورد نظر را وارد کرد، مقدار مراقبتی را وارد می‌کند تا از حلقه ورود نمرات خارج شود. برنامه در هر بار که داده وارد می‌شود مقدار آنرا با مقدار مراقبتی مقایسه می‌کند. دومین اصلاح بر روی عبارت شبه‌کد قبلی می‌تواند بصورت زیر باشد:

Prompt the user to enter the first grade
Input the first grade (possibly the sentinel)
While the user has not yet entered the sentinel
    Add this grade to the running total
    Add one to the grade counter
    Input the next grade (possibly the sentinel)

عبارت شبه‌کد زیر

Calculate and print the total of all student grades and the class average

می‌تواند به صورت عبارات جزئی‌تر زیر نوشته شود:

If the counter is not equal to zero
    Set the average to the total divided by the counter
    Print the total of all student grades in the class
    Print the average
Else
    Print "No grades were entered"

توجه کنید که در این قسمت برای جلوگیری از بروز خطای منطقی تقسیم بر صفر، یک تست بکار برده شده است که اگر در برنامه تشخیص داده نشود، می‌تواند مشکل ساز شود. شبه‌کد کامل برنامه میانگین در شکل ۱۱-۴ آورده شده است.

خطای برنامه‌نویسی

نتیجه تقسیم بر صفر خطای عظیم در زمان اجرا است.

اجتناب از خطا

به هنگام اجرای یک عمل تقسیم بر عبارتی که ممکن است مقدار آن صفر باشد، باید تستی به همین منظور و رسیدگی به آن در برنامه تدارک دیده شود. رسیدگی به این امر می‌تواند چاپ یک پیغام ساده خطا باشد. گاهی اوقات انجام عملیات پیچیده مورد نیاز است.

Initialize total to zero
Initialize counter to zero
Input the first grade (possibly the sentinel)
While the user has not as yet entered the sentinel
    Add this grade to the running total
    Add one to the grade counter
    Input the next grade (possibly the sentinel)

```
If the counter is not equal to zero
        Set the average to the total divided by the counter
        Print the average
Else
        Print "No grades were entered"
```

شکل ۱۱-٤ | الگوریتم شبه‌کد با استفاده از روش مراقبت-کنترل تکرار برای حل مسئله میانگین کلاس.

برنامه‌نویسی ایده‌ال

با قراردادن خطوط خالی در برنامه‌های شبه‌کد خوانائی آنها افزایش می‌یابد. خطوط خالی موجب می‌شوند تا عبارتهای کنترلی شبه‌کد و فازهای برنامه از هم متمایز شوند.

مهندسی نرم‌افزار

بسیاری از الگوریتم‌ها را می‌توان به صورت منطقی به سه فاز تقسیم کرد: فاز مقدار دهی که در آن متغیرهای برنامه مقداردهی اولیه می‌شوند، فاز پردازش که مقادیر داده‌ها وارد شده و متغیرها براساس آنها تنظیم می‌شوند و فاز پایان که مرحله انجام محاسبات و چاپ نتایج است.

الگوریتم شبه‌کد ۱۱-۴ مسئله میانگین کلاس را که در ابتدای این بخش بصورت کلی بیان شده بود، برطرف می‌کند. این الگوریتم فقط پس از طی دو مرحله اصلاح گام به گام توسعه یافت، در حالیکه گاهی اوقات به انجام مراحل بیشتر نیاز است.

مهندسی نرم‌افزار

برنامه‌نویسان زمانی به فرآیند از بالا به پایین و اصلاح گام به گام پایان می‌دهند که الگوریتم شبه‌کد بصورت مشخص جزئیات را بیان کرده باشد، به نحوی که بتوان آنها را به برنامه C++ تبدیل کرد. در اینحالت پیاده سازی برنامه C++ براحتی می‌تواند صورت گیرد.

پیاده‌سازی کلاس GradeBook به روش مراقبت-کنترل تکرار

شکل‌های ۱۲-۴ و ۱۳-۴ کلاس **GradeBook** را به نحوی نشان می‌دهند که حاوی تابع عضو **determineClassAverage** است که الگوریتم شبه کد شکل ۱۱-۴ را پیاده‌سازی می‌کند (این کلاس در شکل ۱۴-۴ توصیف شده است). اگر چه هر نمره وارد شده یک عدد صحیح است، امکان تولید یک عدد اعشاری به هنگام محاسبه میانگین وجود دارد، به عبارتی یک عدد حقیقی یا عدد با نقطه اعشار (همانند 7.33، 0.0975 یا 1000.12345). نوع داده **int** نمی‌تواند چنین اعدادی را عرضه کند، از اینرو این کلاس باید از نوع داده دیگری استفاده کند. زبان C++ دارای چندین نوع داده برای ذخیره‌سازی اعداد اعشاری در حافظه است، نوع‌های همانند **float** و **double**. تفاوت اصلی مابین این نوع در این است که در مقایسه با متغیرهای **float**، متغیرهای **double** قادر به نگهداری اعداد بزرگتر و دقیق‌تر در سمت نقطه اعشار هستند، در نتیجه دقت عدد بیشتر خواهد بود. این برنامه مبادرت به معرفی یک عملگر ویژه بنام عملگر *cast* است

که محاسبه میانگین را مجبور می‌کند تا نتیجه را بصورت عدد اعشاری تولید کند. این ویژگی به هنگام بررسی برنامه توضیح داده خواهد شد.

در این مثال مشاهده می‌کنید که عبارات کنترلی می‌توانند به صورت پشته یکی بر روی دیگری قرار داده شوند (بصورت متوالی). عبارت **while** در خطوط 75-67 از شکل۱۳-۴ بلافاصله پس از عبارات **if..else** قرار گرفته است و حالت توالی دارد. قسمت اعظم کد بکار رفته در این مثال با کد برنامه ۹-۴ یکسان است، از اینرو تمرکز خود را بر روی ویژگی‌ها و مباحث جدید متمرکز می‌کنیم.

```
1  // Fig. 4.12: GradeBook.h
2  // Definition of class GradeBook that determines a class average.
3  // Member functions are defined in GradeBook.cpp
4  #include <string> // program uses C++ standard string class
5  using std::string;
6
7  // GradeBook class definition
8  class GradeBook
9  {
10 public:
11    GradeBook( string ); // constructor initializes course name
12    void setCourseName( string ); // function to set the course name
13    string getCourseName(); // function to retrieve the course name
14    void displayMessage(); // display a welcome message
15    void determineClassAverage(); // averages grades entered by the user
16 private:
17    string courseName; // course name for this GradeBook
18 }; // end class GradeBook
```

شکل ۱۲-٤ | برنامه میانگین کلاس با روش مراقبت-کنترل تکرار: فایل سرآیند GradeBook

```
1  // Fig. 4.13: GradeBook.cpp
2  // Member-function definitions for class GradeBook that solves the
3  // class average program with sentinel-controlled repetition.
4  #include <iostream>
5  using std::cout;
6  using std::cin;
7  using std::endl;
8  using std::fixed; // ensures that decimal point is displayed
9
10 #include <iomanip> // parameterized stream manipulators
11 using std::setprecision; // sets numeric output precision
12
13 // include definition of class GradeBook from GradeBook.h
14 #include "GradeBook.h"
15
16 // constructor initializes courseName with string supplied as argument
17 GradeBook::GradeBook( string name )
18 {
19    setCourseName( name ); // validate and store courseName
20 } // end GradeBook constructor
21
22 // function to set the course name;
23 // ensures that the course name has at most 25 characters
24 void GradeBook::setCourseName( string name )
25 {
26    if ( name.length() <= 25 ) // if name has 25 or fewer characters
```

```
      courseName = name; // store the course name in the object
   else // if name is longer than 25 characters
   { // set courseName to first 25 characters of parameter name
      courseName = name.substr( 0, 25 ); // select first 25 characters
      cout << "Name \"" << name << "\" exceeds maximum length (25).\n"
         << "Limiting courseName to first 25 characters.\n" << endl;
   } // end if...else
} // end function setCourseName

// function to retrieve the course name
string GradeBook::getCourseName()
{
   return courseName;
} // end function getCourseName

// display a welcome message to the GradeBook user
void GradeBook::displayMessage()
{
   cout << "Welcome to the grade book for\n" << getCourseName() << "!\n"
      << endl;
} // end function displayMessage

// determine class average based on 10 grades entered by user
void GradeBook::determineClassAverage()
{
   int total; // sum of grades entered by user
   int gradeCounter; // number of grades entered
   int grade; // grade value
   double average; // number with decimal point for average

   // initialization phase
   total = 0; // initialize total
   gradeCounter = 0; // initialize loop counter

   // processing phase
   // prompt for input and read grade from user
   cout << "Enter grade or -1 to quit: ";
   cin >> grade; // input grade or sentinel value

   // loop until sentinel value read from user
   while ( grade != -1 ) // while grade is not -1
   {
      total = total + grade; // add grade to total
      gradeCounter = gradeCounter + 1; // increment counter

      // prompt for input and read next grade from user
      cout << "Enter grade or -1 to quit: ";
      cin >> grade; // input grade or sentinel value
   } // end while

   // termination phase
  if ( gradeCounter != 0 ) // if user entered at least one grade...
   {
      // calculate average of all grades entered
      average = static_cast< double >( total ) / gradeCounter;

      // display total and average (with two digits of precision)
      cout << "\nTotal of all " << gradeCounter << " grades entered is "
         << total << endl;
```

```
86       cout<< "Class average is" << setprecision( 2 ) << fixed << average
87          << endl;
88    } // end if
89    else // no grades were entered, so output appropriate message
90       cout << "No grades were entered" << endl;
91 } // end function determineClassAverage
```

شکل ۱۳-٤ | برنامه میانگین کلاس با روش مراقبت-کنترل تکرار: فایل کد منبع GradeBook

```
1  // Fig. 4.14: fig04_14.cpp
2  // Create GradeBook object and invoke its determineClassAverage function.
3
4  // include definition of class GradeBook from GradeBook.h
5  #include "GradeBook.h"
6
7  int main()
8  {
9     // create GradeBook object myGradeBook and
10    // pass course name to constructor
11    GradeBook myGradeBook( "CS101 C++ Programming" );
12
13    myGradeBook.displayMessage(); // display welcome message
14    myGradeBook.determineClassAverage(); // find average of 10 grades
15    return 0; // indicate successful termination
16 } // end main
```

```
Welcome to the grade book for
CS101 C++ Programming

Enter grade or -1 to quit: 97
Enter grade or -1 to quit: 88
Enter grade or -1 to quit: 72
Enter grade or -1 to quit: -1

Total of all 3 grades entered is 257
Class average is 85.67
```

شکل۱٤-٤|برنامه میانگین کلاس با روش مراقبت-کنترل تکرار: ایجاد یک شی از کلاس GradeBook(شکل۱۲-٤ و ۱۳-٤) و فراخوانی تابع عضو determineClassAverage.

در خط 55 متغیر **average** از نوع **double** اعلان شده است. این نوع به محاسبه میانگین امکان می‌دهد تا بصورت یک عدد اعشاری در متغیر ذخیره گردد. در خط 59 متغیر **gradeCounter** با صفر مقداردهی شده چرا که هنوز نمره‌ای وارد نشده است، به یاد داشته باشید که این برنامه از روش مراقبت-کنترل تکرار استفاده می‌کند. به منظور ثبت دقیق تعداد نمرات وارد شدهٔ، متغیر **gradeCounter** فقط به هنگام وارد شدن یک نمره معتبر بعنوان ورودی، افزایش می‌یابد.

تفاوت‌های موجود مابین روش‌های مراقبت-کنترل تکرار و شمارنده-کنترل تکرار

به تفاوت‌های موجود میان روش مراقبت-کنترل تکرار در این برنامه و شمارنده-کنترل تکرار در برنامه ۹-۴ توجه کنید. در روش شمارنده-کنترل تکرار، در هر بار تکرار عبارت **while** (خطوط 63-57 از شکل ۹-۴) یک مقدار از سوی کاربر دریافت می‌گردید. در روش مراقبت-کنترل تکرار، قبل از اینکه برنامه به عبارت **while** برسد، یک مقدار (خطوط 64-63 از شکل ۱۳-۴)) دریافت می‌شود. این مقدار تعیین می‌کند که آیا جریان کنترل برنامه وارد بدنه عبارت **while** شود یا خیر. اگر شرط عبارت **while** برقرار

نباشد (کاربر مقدار مراقبتی وارد کرده باشد)، بدنه عبارت **while** اجرا نخواهد شد (هیچ نمره‌ای وارد نمی‌شود). از سوی دیگر، اگر شرط برقرار شود، بدنه اجرا شده و مقدار وارد شدهٔ کاربر بکار گرفته می‌شود (به **total** افزوده می‌شود، خط 69). پس از پردازش مقدار، مقدار بعدی قبل از اینکه برنامه به انتهای بدنه عبارت **while** برسد توسط کاربر وارد می‌شود (خطوط 74-73). زمانیکه برنامه به **{** در خط 75 می‌رسد، اجرا با تست بعدی در شرط عبارت **while** ادامه می‌یابد (خط 67). مقدار جدید وارد شده تعیین می‌کند که آیا عبارت بدنه **while** مجدداً اجرا شود یا خیر. دقت کنید که مقدار بعدی همیشه قبل از اینکه شرط عبارت **while** ارزیابی شود، بلافاصله توسط کاربر وارد می‌شود. در اینحالت برنامه می‌تواند قبل از اینکه اقدام به پردازش مقداری نماید، تعیین کند که آیا آن مقدار، مقدار مراقبتی است یا خیر. اگر مقدار مراقبتی باشد، عبارت **while** خاتمه می‌یابد و مقدار به **total** افزوده نمی‌شود.

پس از خاتمه حلقه، عبارت if..else در خطوط 90-78 اجرا می‌شود. شرط موجود در خط 78 تعیین می‌کند که آیا نمره‌ای وارده شده است یا خیر. اگر نمره‌ای وارد نشده باشد، بخش else (خطوط 90-89) از عبارت if..else اجرا شده و پیغام "No grades were entered" را به نمایش درآورده و تابع عضو کنترل را به تابع فراخوان برگشت می‌دهد.

به بلوک موجود در حلقه while شکل ۱۳-۴ دقت کنید. بدون حضور براکت‌ها، سه عبارت آخر در بدنه حلقه در خارج از حلقه جای می‌گرفتند و این سبب می‌شد که کامپیوتر این کد را بصورت زیر و نادرست تفسیر کند:

```
// loop until sentinel value read from user
   while ( grade != -1 )
      total = total + grade; // add grade to total
   gradeCounter = gradeCounter + 1; // increment counter

   // prompt for input and read next grade from user
      cout << "Enter grade or -1 to quit: ";
      cin >> grade;
```

در این حالت برنامه دچار یک حلقه بی‌نهایت می‌شود در صورتیکه کاربر 1- را به عنوان اولین نمره وارد نکند (خط 64).

خطای برنامه‌نویسی

فراموش کردن براکت‌های تعیین‌کننده مرز یک بلوک می‌تواند، سبب‌ساز خطاهای منطقی همانند حلقه‌های بی‌نهایت شود.

برنامه‌نویسی ایده‌ال

در حلقه کنترل مقدار مراقبتی، که مقداری از کاربر تقاضا می‌کند، باید مقدار مراقبتی به کاربر نشان داده شود.

دقت اعداد اعشاری و نیاز حافظه

متغیرهای از نوع **float** عرضه‌کننده اعداد با دقت منفرد در نقطه اعشار هستند و دارای هفت رقم معنی‌دار در سیستم‌های 32 بیتی می‌باشند. متغیرهای از نوع **double** عرضه‌کننده دقت مضاعف در نقطه اعشار هستند. این دقت مستلزم دو برابر حافظه مورد نیاز برای یک متغیر **float** است و دارای 15 رقم معنی‌دار در سیستم‌های 32 بیتی است (تقریباً دو برابر دقیق‌تر از متغیرهای**float**). برای اکثر محاسبات صورت گرفته در برنامه‌ها نوع **float** می‌تواند کافی باشد، اما می‌توانید با استفاده از **double** دقت را تضمین کنید. در برخی از برنامه‌ها، حتی متغیرهای از نوع **double** هم کافی نیستند، برنامه‌هایی که خارج از قلمرو بحث این کتاب هستند. اکثر برنامه‌نویسان برای عرضه اعداد اعشاری از نوع **double** استفاده می‌کنند. در واقع C++ بطور پیش‌فرض با تمام اعداد اعشاری که در کد منبع برنامه تایپ می‌کنید (همانند 7.33 و 0.0975) همانند مقادیر **double** رفتار می‌کند. چنین مقادیری در کد منبع بعنوان ثابت‌های اعشاری شناخته می‌شوند.

غالبا اعداد اعشاری در انجام عملیات تقسیم گسترش زیادی پیدا می‌کنند. برای مثال با تقسیم 10 بر 3، نتیجه ...3.333333 با دنباله‌ای از 3های نامتناهی خواهد بود. کامپیوتر فضای ثابتی برای نگهداری چنین مقادیری در اختیار دارد، از اینرو ذخیره سازی مقادیر اعشاری فقط بصورت تخمینی صورت می‌گیرد.

علیرغم اینکه اعداد اعشاری همیشه 100 درصد دقیق نیستند، اما کاربردهای بسیاری دارند. برای مثال، هنگامی که در مورد حرارت عادی بدن یعنی 98.6 صحبت می‌کنیم، نیازی نیست تا دقت اعشاری آنرا بسیار دقیق بیان کنیم. زمانیکه به درجه حرارت در یک دماسنج نگاه می‌کنیم و آنرا 98.6 می‌خوانیم، ممکن است مقدار دقیق آن 98.5999473210643 باشد. اما استفاده از مقدار 98.6 به صورت تخمینی در بسیاری از موارد می‌تواند مناسب و کاربردی باشد.

خطای برنامه‌نویسی

استفاده از اعداد اعشاری با فرض اینکه این اعداد نشاندهنده مقدار کاملاً دقیق هستند (بویژه در عبارات مقایسه‌ای) می‌تواند نتایج اشتباهی بدنبال داشته باشد. اعداد اعشاری تقریبا در تمام کامپیوترها نشان‌دهنده یک مقدار تقریبی هستند.

تبدیل مابین نوع‌های بنیادین بصورت صریح و ضمنی

متغیر **average** بصورت **double** (خط 55 از شکل ۱۳-۴) اعلان شده تا نتیجه اعشاری محاسبه انجام گرفته را در خود ذخیره سازد. با این همه، متغیرهای **total** و **gradeCounter** هر دو از نوع صحیح می‌باشند. بخاطر دارید که نتیجه تقسیم دو عدد صحیح یک عدد صحیح است که در آن بخش اعشاری جواب از بین می‌رود (قطع می‌شود). در عبارت زیر

```
average = total / gradeCounter;
```

ابتدا تقسیم انجام می‌شود، از اینرو بخش اعشاری نتیجه قبل از تخصیص به **average** از بین می‌رود. برای انجام یک محاسبه اعشاری با مقادیر صحیح، بایستی مقادیر موقتی که اعداد اعشاری هستند برای محاسبه ایجاد کنیم. زبان C++ دارای *عملگر غیربا ینری cast* است که این وظیفه را انجام می‌دهد. در خط 81 از عملگر cast بصورت **static_cast<double>(total)** برای ایجاد یک کپی موقت اعشاری از عملوند موجود در درون پرانتزها یعنی **total** استفاده شده است. به استفاده از یک عملگر cast به این روش، *تبدیل صریح* می گویند. هنوز مقدار ذخیره شده در **total** یک عدد صحیح است.

اکنون محاسبه متشکل از یک مقدار اعشاری (نسخه **double** موقت از **total**) است که بر یک عدد صحیح در **gradeCounter** تقسیم می‌شود. کامپایلر C++ فقط از نحوه ارزیابی عباراتی که در آن نوع داده‌های عملوندها یکسان هستند، اطلاع دارد. برای اطمینان از اینکه عملوندها از نوع مشابه هستند، کامپایلر مبادرت به انجام عملی بنام *ترفیع* که *تبدیل ضمنی* نیز نامیده می‌شود بر روی عملوندهای انتخابی می کند. برای مثال، در یک عبارت که حاوی مقادیری از نوع داده **int** و **double** است، C++ مبادرت به ترفیع عملوندهای int به مقادیر **double** می کند. در این مثال با **total** همانند یک نوع داده **double** رفتار می کنیم (با استفاده از عملگر cast)، از اینرو کامپایلر مبادرت به ترفیع **gradeCounter** به **double** کرده و به محاسبه اجازه انجام می‌دهد و نتیجه تقسیم اعشاری به **average** تخصیص می‌یابد. در فصل ششم، در مورد نوع داده‌های بنیادین و نحوه ترفیع آنها توضیح خواهیم داد.

خطای برنامه‌نویسی

*می‌توان از عملگر **cast** برای تبدیل مابین نوع‌های بنیادین عددی همانند **int** و **double** و مابین نوع کلاس‌های مرتبط استفاده کرد (در فصل سیزدهم با این موضوع آشنا خواهید شد). تبدیل به یک نوع اشتباه می‌تواند خطای کامپایلر یا خطای زمان اجرا بوجود آورد.*

عملگرهای cast برای استفاده در هر نوع داده و همچنین نوع‌های کلاس در دسترس هستند. بدنبال عملگر **static_cast** یک جفت کاراکتر < و > که نوع داده را احاطه کرده‌اند آورده می‌شود. عملگر cast یک عملگر غیربا ینری است. عملگری که فقط یک عملوند اختیار می کند. در فصل دوم، با عملگرهای محاسباتی باینری آشنا شده‌اید. همچنین C++ از نسخه‌های عملگرهای غیربا ینری جمع (+) و منفی (-) پشتیبانی می کند، از اینرو برنامه‌نویس می‌تواند عبارتی مثل 7- یا 5+ بنویسد. عملگر cast از سایر عملگرهای غیربا ینری همانند + و – از تقدم بالاتری برخوردار است. این تقدم بالاتر از عملگرهای *، / و % و پایین‌تر از پرانتز است. در جدول شکل ۲۲-۴ این عملگر را با نماد **static_cast<*type*>()** عرضه کرده‌ایم.

قالب‌بندی اعداد اعشاری

قالب‌بندی بکار رفته در برنامه شکل۱۳-۴ را بطور خلاصه در این بخش و بطور دقیق‌تر در فصل پانزدهم توضیح خواهیم داد. فراخوانی تابع **setprecision** در خط 86 (با آرگومان 2) بر این نکته دلالت دارد که متغیر **average** از نوع **double** بایستی با دو رقم معنی‌دار در سمت راست نقطه اعشار چاپ شود (مثلاً 97.37) به اینحالت *کنترل‌کننده جریان پارامتری شده* (استریم) می‌گویند (بدلیل وجود 2 در درون پرانتز). برنامه‌هایی که از این فراخوانی استفاده می‌کنند باید حاوی رهنمود دستور دهنده زیر باشند (خط 10)

```
#include <iomanip>
```

خط 11 تصریح‌کننده نام فایل سرآیند <**iomanip**> است که در این برنامه بکار گرفته خواهد شد. دقت کنید که **endl** یک کنترل‌کننده جریان پارامتری نشده است (چرا که پس از آن مقدار یا عبارتی در درون پرانتزها وجود ندارد) و نیازمند فایل سرآیند <**iomanip**> نیست. اگر دقت تعیین نشود، معمولاً اعداد اعشاری با شش رقم معنی‌دار چاپ می‌شوند (دقت پیش‌فرض در اکثر سیستم‌های 32 بیتی). کنترل‌کننده جریان **fixed** بر این نکته دلالت دارد که مقادیر اعشاری بایستی با خروجی که فرمت نقطه ثابت نامیده می‌شوند چاپ شوند، که متضاد نماد علمی می‌باشد. نماد علمی روشی برای نمایش یک عدد بصورت، عدد اعشاری مابین مقدار1 الی10 است که در توانی از 10 ضرب می‌شود. برای مثال، مقدار3100 را می‌توان در نماد علمی بصورت3.1×10^3 به نمایش در آورد. به هنگام نمایش مقادیری که بسیار بزرگ یا بسیار کوچک هستند، نماد علمی می‌تواند ابزار مناسبی برای اینکار باشد در فصل پانزدهم با قالب‌بندی نماد علمی آشنا خواهید شد. در طرف مقابل، قالب‌بندی نقطه ثابت قرار دارد که یک عدد اعشاری را مجبور می‌کند تا به تعداد مشخص شده مبادرت به نمایش ارقام کند. همچنین این فرمت نقطه اعشار و دنباله صفرها در چاپ را کنترل می‌کند، حتی اگر عدد یک عدد صحیح باشد، همانند 88.00، بدون قالب‌بندی نقطه ثابت چنین عددی در C++ بصورت 88 چاپ می‌شود، بدون دنباله صفرها و نقطه اعشار. زمانیکه از کنترل‌کننده‌های جریان **fixed** و **setprecision** در برنامه‌ای استفاده می‌شود، مقادیر چاپ شده به تعداد نقاط دیسمال که توسط مقدار ارسالی به **setprecision** مشخص می‌شود، گرد می‌شوند (همانند مقدار 2 در خط 86)، اگرچه مقدار موجود در حافظه بدون تغییر باقی می‌ماند. برای مثال، مقادیر 87.946 و 67.543 بصورت 87.95 و 67.54 چاپ می‌شوند. توجه کنید که می‌توان نقطه اعشار را با استفاده از کنترل‌کننده جریان **showpoint** به نمایش درآورد. اگر **showpoint** بدون **fixed** بکار گرفته شود، دنباله صفحه‌ها چاپ نخواهد شد. همانند **endl**، کنترل‌کننده‌های جریان**fixed** و**showpoint**

پارامتری شده نبوده و نیازی به سرآیند فایل <**iomanip**> ندارند. هر دو آنها را می‌توان در سرآیند <**iostream**> پیدا کرد.

خط 86 و 87 از شکل ۱۳-۴ خروجی میانگین کلاس هستند. در این مثال میانگین کلاس گرد شده به نزدیکترین صدم و دقیقاً با دو رقم در سمت راست نقطه اعشار به نمایش درآمده‌اند. کنترل‌کننده جریان پارامتری شده (خط 86) نشان می‌دهد که مقدار متغیر **average** بایستی با دقت دو رقم در سمت راست نقطه اعشار به نمایش درآید (**setprecision(2)**). در اجرای نمونه‌ای برنامه سه نمره وارد برنامه ۱۴-۴ شده که مجموع آنها 257 شده است و میانگین حاصل از این رقم عدد 85.666666 است. کنترل‌کننده جریان پارامتری شده **setprecision** سبب می‌شود تا مقدار به تعداد رقم مشخص گرد شود. در این برنامه، میانگین به 85.67 گرد شده است.

۱۰-٤ فرموله کردن الگوریتم‌ها: عبارات کنترلی تودرتو

اجازه دهید تا به بررسی مسئله دیگری بپردازیم. مجدداً الگوریتم را با استفاده از شبه‌کد و از بالا به پایین، اصلاح گام به گام فرموله کرده و سپس برنامه C++ مربوط به آنرا خواهیم نوشت. در مثال‌های قبلی مشاهده کردید که عبارتهای کنترلی همانند یک پشته یکی بر روی دیگری و به ترتیب قرار داده می‌شدند. در این مرحله، به معرفی روشی خواهیم پرداخت که عبارتهای کنترلی در آن را می‌توان با یکدیگر ترکیب کرد، بطوریکه عبارتی در درون عبارت دیگر جای می‌گیرد.

به صورت مسئله توجه نمائید:

یک کالج با برگزاری دوره‌ای دانشجویان را آماده امتحان پایان ترم می‌کند. سال گذشته، ۱۰ تن از دانشجویان که این دوره را گذرانده بودند در امتحان پایان ترم شرکت کردند. مدیریت کالج می‌خواهد از وضعیت دانشجویان شرکت کرده در امتحان مطلع شود. از شما خواسته شده تا برنامه‌ای بنویسید تا خلاصه‌ای از نتایج آزمون ارائه دهد. لیستی از ۱۰ دانشجو دریافت کرده و سپس در کنار نام کسانی که در آزمون قبول شده‌اند 1 و کسانی که در آزمون مردود شده‌اند 2 چاپ شود.

این برنامه باید بصورت زیر نتایج آزمون را تحلیل نماید:

۱- وارد کردن نتیجه هر آزمون (برای مثال 1 یا 2). نمایش پیغام "Enter result" در هر بار که برنامه درخواست نتیجه آزمون می‌کند.

۲- شمارش تعداد قبولی‌ها و مردودی‌ها.

۳- نمایش خلاصه‌ای از نتایج آزمون، شامل تعداد دانشجویان که موفق به گذراندن آزمون شده‌اند و تعدادی که مردود شده‌اند.

۴-اگر بیش از ۸ دانشجو از آزمون با موفقیت عبور کرده‌اند. پیغام "Raise tuition" به نمایش درآید.

پس از مطالعه صورت مسئله، تصمیمات زیر را برای حل آن اتخاذ می‌کنیم:

۱- برنامه باید بر روی نتایج آزمون 10 دانشجو کار کند، از اینرو حلقه شمارنده - کنترل می‌تواند بکار گرفته شود.

۲- نتیجه هر آزمون عدد 1 یا 2 است. هر بار که برنامه اقدام به خواندن نتیجه یک آزمون می‌کند، برنامه باید یک 1 یا 2 دریافت نماید.

۳- دو شمارنده به ذخیره‌سازی نتایج آزمون می‌پردازند. یکی برای شمارش تعداد دانشجویان که از آزمون با موفقیت عبور کرده‌اند و دیگری برای شمارش تعدادی که در آزمون مردود شده‌اند.

۴- پس از اینکه برنامه تمام نتایج را مورد پردازش قرار داد، باید تعیین کند که آیا تعداد قبولی‌ها بیش از هشت نفر است یا خیر.

اجازه دهید تا با روش از بالا به پایین، اصلاح گام‌به‌گام کار را دنبال کنیم. عبارت شبه‌کد زیر در بالاترین سطح (top) قرار دارد:

Analyze exam result and decide if tuition should be raised

مجدداً یادآوری می‌کنیم که عبارت top توصیف کلی در مورد برنامه است و قبل از اینکه بتوان شبه‌کد را به فرم یک برنامه C++ نوشت انجام چندین مرحله اصلاح گام‌به‌گام ضروری است. اولین اصلاح عبارت است از:

Initialize variables
Input the 10 exam grades and count passes and failures
Print a summary of the exam result and decide if tuition should be raised

حتی زمانیکه یک تصور کامل از کل برنامه بدست آورده باشیم، انجام اصلاحات بعدی مورد نیاز است. باید به دقت به بررسی و مشخص کردن متغیرها پرداخت. شمارنده‌ها به منظور ثبت قبولی‌ها و مردودی‌ها مورد نیاز هستند. یک شمارنده، کنترل کننده حلقه بوده و یک متغیر، ورودی کاربر را ذخیره می‌کند. عبارت شبه‌کد زیر

Initialize variables

می‌تواند بصورت زیر اصلاح شود:

Initialize passes to zero
Initialize failures to zero
Initialize student counter to one

فقط شمارنده‌های، تعداد قبولی‌ها و مردودی‌ها و تعداد دانش‌آموزان مقداردهی اولیه می‌شود. عبارت شبه‌کد

Input the 10 quiz grades and count passes and failures

مستلزم یک حلقه برای وارد کردن نتیجه هر آزمون است. در این برنامه بدلیل اینکه از همان ابتدا تعداد نتایج آزمون مشخص است (۱۰)، از اینرو می‌توان از روش شمارنده-کنترل تکرار استفاده کرد. در درون

حلقه یک عبارت انتخابی دو گانه تعیین می‌کند که نتیجه آزمون قبولی است یا مردودی و شمارنده مربوطه یک واحد افزایش می‌یابد. اصلاح عبارت شبه‌کد قبلی می‌تواند بصورت زیر انجام شود:

```
While student counter is less than or equal to 10
    Prompt the user to enter the next exam result
    Input the next exam result
    If the student passed
            Add one to passes
    Else
            Add one to failures
    Add one to student counter
```

به نحوه استفاده از خطوط خالی در میان مجموعه ***if..else*** دقت کنید که باعث افزایش خوانایی برنامه شده است. عبارت شبه‌کد

```
Print a summary of the exam results and decide if tuition should be raised
```

می‌تواند بصورت زیر اصلاح شود:

```
Print the number of passes
Print the number of failures
If more than eight student passed
    Print "Raise tuition"
```

در شکل ۱۵-۴ دومین مرحله اصلاح بصورت کامل نشان داده شده است. به کاربرد خطوط خالی در میان ساختار **while** توجه کنید، که باعث افزایش خوانائی برنامه می‌شوند. اکنون این شبه‌کد بقدر کافی برای تبدیل به یک برنامه C++ آماده شده است.

```
Initialize passes to zero
Initialize failures to zero
Initialize student to zero
While student counter is less than or equal to ten
        Input the next exam result
        If the student passed
                Add one to passes
        Else
                Add one to failures
                Add one to student counter
Print the number of passes
Print the number of failures
If more than eight students passed
        Print "Raise tuition"
```

شکل ۱۵-٤ | شبه‌کد برنامه نتیجه آزمون.

تبدیل به کلاس Analysis

کلاس C++ که مبادرت به پیاده‌سازی الگوریتم شبه‌کد کرده است در شکل‌های ۱۶-۴ و ۱۷-۴ و دو اجرای نمونه در شکل ۱۸-۴ نشان داده است.

```
1  // Fig. 4.16: Analysis.h
2  // Definition of class Analysis that analyzes examination results.
3  // Member function is defined in Analysis.cpp
4
5  // Analysis class definition
6  class Analysis
7  {
8  public:
9     void processExamResults(); // process 10 students' examination results
10 }; // end class Analysis
```

شکل ١٦-٤ | برنامه بررسی نتیجه آزمون: فایل سرآیند Analysis.

```
1  // Fig. 4.17: Analysis.cpp
2  // Member-function definitions for class Analysis that
3  // analyzes examination results.
4  #include <iostream>
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  // include definition of class Analysis from Analysis.h
10 #include "Analysis.h"
11
12 // process the examination results of 10 students
13 void Analysis::processExamResults()
14 {
15    // initializing variables in declarations
16    int passes = 0; // number of passes
17    int failures = 0; // number of failures
18    int studentCounter = 1; // student counter
19    int result; // one exam result (1 = pass, 2 = fail)
20
21    // process 10 students using counter-controlled loop
22    while ( studentCounter <= 10 )
23    {
24       // prompt user for input and obtain value from user
25       cout << "Enter result (1 = pass, 2 = fail): ";
26       cin >> result; // input result
27
28       // if...else nested in while
29       if ( result == 1 )          // if result is 1,
30          passes = passes + 1;     // increment passes;
31       else                        // else result is not 1, so
32          failures = failures + 1; // increment failures
33
34       // increment studentCounter so loop eventually terminates
35       studentCounter = studentCounter + 1;
36    } // end while
37
38    // termination phase; display number of passes and failures
39    cout << "Passed " << passes << "\nFailed " << failures << endl;
40
41    // determine whether more than eight students passed
```

```
42    if ( passes > 8 )
43       cout << "Raise tuition " << endl;
44 } // end function processExamResults
```

شکل ۱۷-٤ |برنامه بررسی نتیجه آزمون: عبارات کنترلی تودرتو در فایل کد منبع Analysis

```
1  // Fig. 4.18: fig04_18.cpp
2  // Test program for class Analysis.
3  #include "Analysis.h" // include definition of class Analysis
4
5  int main()
6  {
7     Analysis application; // create Analysis object
8     application.processExamResults(); // call function to process results
9     return 0; // indicate successful termination
10 } // end main
```

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 9
Failed: 1
Raise Tuition
```

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Passed: 6
Failed: 4
```

شکل ۱۸-٤ | برنامه تست کننده کلاس Analysis

خطوط 16-18 از برنامه۱۷-۴ متغیرهای اعلان کرده‌اند که تابع عضو **processExamResults** از کلاس **Analysis** از آنها برای پردازش نتایج آزمون استفاده می‌کند. توجه کنید که از یکی از ویژگی‌های زبان C++ استفاده کرده‌ایم که به مقداردهی اولیه متغیر امکان می‌دهد تا با بخش اعلان یکی شود (**passes** با صفر، **failures** با صفر و **studentCounter** با 1 مقداردهی اولیه شده‌اند). امکان مقداردهی در ابتدای تکرار هر حلقه وجود دارد، معمولاً چنین مقداردهی‌های مجددی توسط عبارات تخصیصی بجای اعلان‌ها یا انتقال اعلان‌ها بدرون بدنه حلقه صورت می‌گیرند.

حلقه while ده بار تکرار می‌شود (خطوط 36-22). در هر تکرار، حلقه یک نتیجه آزمون دریافت و آن را پردازش می‌کند. دقت کنید که عبارت **if..else** در خطوط 32-29 برای پردازش هر نتیجه در عبارت **while** بصورت تودرتو قرار گرفته است. اگر **result** برابر **1** باشد، عبارت **if..else** یک واحد به **passes** اضافه می‌کند و در غیر اینصورت فرض می‌کند که **result** برابر **2** بوده و یک واحد به **failures** اضافه می‌نماید. خط 35 مبادرت به افزایش **studentCouunter** قبل از اینکه شرط تست حلقه در خط 22 صورت گیرد می‌کند. پس دریافت 10 مقدار، حلقه پایان یافته و خط 39 تعداد قبولی‌ها (**passes**) و مردودی‌ها (**failures**) را به نمایش در می‌آورد. عبارت **if** در خطوط 43-42 تعیین می‌کند که آیا تعداد دانشجویان قبول شده بیش از هشت نفر است یا خیر. اگر چنین باشد پیغام "Raise Tuition" به نمایش در می‌آید.

بررسی کلاس Analysis

در برنامه ۱۸-۴ یک شی **Analysis** ایجاد (خط 7) و تابع عضو **processExamResults** فراخوانی می‌شود (خط 8) تا مجموع نتایج آزمون وارد شده توسط کاربر پردازش شود. دو اجرای نمونه از برنامه ۱۸-۴ در خروجی نشان داده شده است. در انتهای اولین اجرا، شرط موجود در خط 42 تابع عضو **procesExamResults** در شکل ۱۷-۴ برقرار شده (بیش از هشت دانشجو قبول شده‌اند) و از اینرو پیغام "Raise Tuition" به نمایش درآمده است.

۱۱-٤ عملگرهای تخصیص‌دهنده

++C دارای چندین عملگر تخصیص‌دهنده برای کاستن از طول عبارات تخصیصی است. برای مثال، عبارت

```
c = c + 3;
```

می‌تواند بصورت زیر و با استفاده از عملگر += نوشته شود

```
c += 3;
```

عملگر += مقدار عملوند قرار گرفته در سمت راست را به مقدار عملوند سمت چپ اضافه کرده و نتیجه آنرا در متغیر عملوند سمت چپ ذخیره می‌کند. هر عبارتی به فرم زیر را

متغیر = عبارت عملگر متغیر;

می‌توان به فرم زیر نوشت:

متغیر عملگر = عبارت;

عملگر یکی از عملگرهای باینری +، -، *، / یا % و متغیر یک مقدار سمت چپ (lvalue) است. مقدار سمت چپ، متغیری است که در سمت چپ یک عبارت تخصیصی جای می‌گیرد. جدول شکل ۱۹-۴ حاوی عملگرهای تخصیص دهنده محاسباتی و عبارات نمونه‌ای است که از این عملگرها استفاده می‌کنند.

عملگر تخصیص دهنده	عبارت نمونه	معادل	تخصیص
با فرض c = 3، d=5، e=4، f=6 و g=12			

+=	c += 7	c = c + 7	c به 10
-=	d -= 4	d = d - 4	d به 1
*=	e *= 5	e = e * 5	e به 20
/=	f /= 3	f = f / 2	f به 2
%=	g %= 9	g = g % 9	g به 3

شکل ۱۹-٤ | عملگرهای تخصیص‌دهنده.

۱۲-٤ عملگرهای افزاینده و کاهنده

علاوه بر عملگرهای محاسباتی تخصیص دهنده، زبان C++ دارای *عملگر افزاینده* غیربایری ++ و *عملگر کاهنده* غیربایری -- است. این عملگرها در جدول شکل ۲۰-۴ توضیح داده شده‌اند. اگر متغیر **c** بخواهد یک واحد افزایش پیدا کند، عملگر افزاینده ++ را می‌توان بجای استفاده از عبارت **c = c + 1** یا **c += 1** بکار گرفت. اگر عملگر افزاینده یا کاهنده قبل از یک متغیر قرار داده شود، به مفهوم *پیش‌افزایش یافته* یا *پیش‌کاهش یافته* خواهد بود. اگر عملگر افزاینده یا کاهنده پس از یک متغیر بکار گرفته شود، به مفهوم *پس‌افزایشی* یا *پس‌کاهشی* خواهد بود.در هر دو حالت افزایشی یا کاهشی مقدار متغیر یک واحد افزایش یا کاهش پیدا می‌کند. پس از انجام کار، مقدار جدید متغیر در عبارتی که حاوی آن است بکار گرفته می‌شود.

عملگر	عبارت نمونه	عنوان	توضیح
++	**++a**	پیش‌افزایشی	**a** یک واحد افزایش می‌یابد، سپس مقدار جدید **a** در عبارتی که حاوی **a** است بکار گرفته می‌شود.
++	**a++**	پس‌افزایشی	از مقدار جاری **a** در عبارتی که حاوی آن است استفاده شده، سپس مقدار **a** یک واحد افزایش میابد.
--	**--b**	پیش‌کاهشی	**b** یک واحد کاهش می‌یابد، سپس مقدار جدید **b** در عبارتی که حاوی **b** است بکار گرفته می‌شود.
--	**b--**	پس‌کاهشی	از مقدار جاری **b** در عبارتی که حاوی آن است استفاده شده، سپس مقدار **b** یک واحد کاهش میابد.

شکل ۲۰-٤ | عملگرهای افزاینده و کاهنده.

برنامه شکل ۲۱-۴ به توصیف تفاوت موجود مابین نسخه‌های پیش‌افزایش و پس‌افزایش عملگر افزاینده ++ می‌پردازد. عملگر کاهنده -- نیز به طریق مشابهی کار می‌کند. توجه کنید که این مثال حاوی کلاس نمی‌باشد، اما فایل کد منبع با **main** تمام برنامه‌ها کار می‌کند. در این فصل و فصل سوم شاهد مثال‌های بوده‌اید که حاوی یک کلاس (شامل سرآیند و فایلهای کد منبع برای این کلاس بوده‌اند) به همراه فایل کد منبع دیگری برای تست کلاس بودند. این فایل کد منبع حاوی تابع **main** است که یک شی از کلاس ایجاد و توابع عضو خود را فراخوانی می‌کند. در این مثال، فقط خواسته‌ایم تا مکانیزم عملگر++ را به نمایش در آوریم، از اینرو فقط از یک فایل کد منبع با تابع **main** استفاده کرده‌ایم. خط 12 مبادرت به

مقداردهی اولیه متغیر c با 5 و خط 13 مقدار اولیه c را به نمایش در می‌آورد. خط 14 مقدار عبارت c++ را چاپ می‌کند. این عبارت سبب پس‌افزایش متغیر c شده و در نتیجه مقدار اولیه c یعنی 5 چاپ می‌گردد، سپس مقدار c افزایش می‌یابد. از اینرو، خط 14 مقدار اولیه c یعنی 5 را مجدداً چاپ می‌کند. خط 15 مقدار جدید c یعنی 6 را برای تاکید بر این نکته که مقدار متغیر براستی در خط 14 افزایش یافته است چاپ می‌نماید.

```
1  // Fig. 4.21: fig04_21.cpp
2  // Preincrementing and postincrementing.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  int main()
8  {
9     int c;
10
11    // demonstrate postincrement
12    c = 5; // assign 5 to c
13    cout << c << endl; // print 5
14    cout << c++ << endl; // print 5 then postincrement
15    cout << c << endl; // print 6
16
17    cout << endl; // skip a line
18
19    // demonstrate preincrement
20    c = 5; // assign 5 to c
21    cout << c << endl; // print 5
22    cout << ++c << endl; // preincrement then print 6
23    cout << c << endl; // print 6
24    return 0; // indicate successful termination
25 } // end main
```

```
5
5
6

5
6
6
```

شکل ۲۱-٤ |تفاوت مابین عملگرهای پیش‌افزایشی و پس‌افزایشی.

خط 20 مقدار متغیر c را به 5 باز می‌گرداند و خط 21 مقدار c را چاپ می‌کند. خط 22 مبادرت به چاپ مقدار عبارت c++ را می‌کند. این عبارت سبب پیش‌افزایش c شده است، از اینرو مقدار آن افزایش یافته و سپس مقدار جدید یعنی 6 چاپ می‌شود. خط 23 مجدداً مقدار c را به نمایش در می‌آورد تا نشان دهد که مقدار c هنوز پس از اجرای خط 22 برابر 6 است.

عملگرهای تخصیص ریاضی و عملگرهای افزاینده و کاهنده می‌توانند عبارات برنامه‌نویسی را ساده‌تر کنند. سه عبارت تخصیصی در برنامه ۱۷-۴

```
passes = passes + 1;
failures = failures + 1;
studentCounter = studentCounter + 1;
```

را می‌توان با استفاده از عملگرهای تخصیصی بصورت زیر هم نوشت

```
passes += 1;
failures += 1;
student += 1;
```

با عملگرهای پیش‌افزایشی بصورت زیر

```
++passes;
++failures;
++studentCounter;
```

با عملگرهای پس‌افزایشی بصورت زیر نوشت

```
passes++
failures++;
studentCounter++;
```

خطای برنامه‌نویسی

مبادرت به استفاده از عملگر افزاینده یا کاهنده بر روی عبارتی بجز نام یک متغیر، همانند ++(x + 1) خطای نحوی خواهد بود.

جدول شکل ۲۲-۴ نمایشی از تقدم و ارتباط عملگرهای مطرح شده تا بدین‌جا را عرضه کرده است. نمایش عملگرها با تقدم آنها از بالا به پایین است. ستون دوم توصیف کننده ارتباط عملگرها در هر سطح تقدم است. دقت کنید که عملگر شرطی (:?) ، عملگر غیربااینری پس‌افزایشی (++) ، پس‌کاهشی (- -) ، جمع (+) ، تفریق (-) ، و عملگرهای = ، =+ ، =* ، =/ و =% از چپ به راست ارزشیابی می‌شوند. مابقی عملگرهای جدول شکل ۲۲-۴ از راست به چپ می‌باشند. ستون سوم اسامی عملگرها را نشان می‌دهد.

عملگر	ارتباط	نوع
( )	چپ به راست	پرانتز
++ -- static_cast< *type* >()	چپ به راست	غیربااینری
++ -- + -	راست به چپ	غیربااینری
* / %	چپ به راست	تعددی
+ -	چپ به راست	افزاینده کاهنده
<< >>	چپ به راست	درج/استخراج
< <= > >=	چپ به راست	رابطه‌ای
== !=	چپ به راست	برابری
?:	راست به چپ	شرطی
= += -= *= /= %=	راست به چپ	تخصیصی

شکل ۲۲-٤ |تقدم و رابطه عملگرهای مطرح شده تا این مرحله.

۱۳-٤ مبحث آموزشی مهندسی نرم‌افزار: شناسایی صفات کلاس در سیستم ATM

در بخش ۱۱-۳ اولین مرحله از طراحی شی‌گرا (OOD) را برای سی‌ستم ATM انجام دادیم. تحلیل مستند نیازها و شناسایی کلاس‌های مورد نیاز در پیاده‌سازی سیستم. همچنین اسامی و کلمات کلیدی موجود در مستند نیازها را مشخص و کلاس‌ها را مجزا کرده و نقشی که هر یک در سیستم ATM بازی می‌کنند را شناسایی کردیم. سپس کلاس‌ها و روابط آنها را توسط دیاگرام کلاس UML مدل‌سازی کردیم (شکل ۲۳-۳). کلاس‌ها دارای صفات (داده) و عملیات (رفتار) هستند. صفات کلاس در برنامه‌های C++ بعنوان عضوهای داده و عملیات کلاس توسط توابع عضو پیاده‌سازی می‌شوند. در این بخش، به تعیین برخی از صفات مورد نیاز در سیستم ATM می‌پردازیم. در فصل پنجم، بررسی می‌کنیم چگونه این صفات در تعیین وضعیت یک شی نقش دارند. در فصل ششم، به تعیین عملیات کلاس‌ها خواهیم پرداخت.

شناسایی صفات

به صفات برخی از شی‌ها در جهان واقعی توجه کنید: هر فردی دارای قد، وزن بوده و می‌تواند چپ‌دست، راست‌دست یا قادر به نوشتن با هر دو دست باشد. صفات یک رادیو شامل تنظیم ایستگاه، تنظیم صدا و تنظیمات AM یا FM است. صفات یک اتومبیل شامل دور موتور، حجم مخزن سوخت و نوع جعبه‌دنده است. صفات یک کامپیوتر شخصی شامل سازنده آن (همانند Dell، Sun، Apple یا IBM)، نوع صفحه نمایش (مثلاً LCD یا CRT)، میزان حافظه اصلی و سایز دیسک سخت است.

می‌توانیم صفات بسیار زیادی برای کلاس‌ها را در سیستم با دقت در کلمات توصیف‌کننده و عبارات موجود در مستند نیازها پیدا کنیم. برای هر صفتی که نقشی در سیستم بازی می‌کند یک صفت ایجاد کرده و آن را به یک یا چند کلاس شناسایی شده در بخش ۱۱-۳ تخصیص می‌دهیم. همچنین صفات را برای نمایش داده‌های اضافی که ممکن است کلاس نیاز داشته باشد یا داده‌های که می‌توانند فرآیند طراحی را مشخص‌تر سازند، ایجاد می‌کنیم.

در جدول شکل ۲۳-۴ کلمات و عبارت بدست آمده از مستند نیازها را که توصیف‌کننده کلاس هستند، لیست شده‌اند. ترتیب دستیابی به این کلاس بر اساس ظاهر شدن آنها در مستند نیازها است.

کلاس	کلمات و جملات توصیفی
ATM	تایید هویت کاربر
BalanceInquiry	شماره حساب
Withdrawl	شماره حساب

	موجودی
Deposit	شماره حساب
	موجودی
BankDatabase	[کلمه یا جمله توصیف‌کننده وجود ندارد]
Account	شماره حساب
	PIN
	مانده حساب
Screen	[کلمه یا جمله توصیف‌کننده وجود ندارد.]
keypad	[کلمه یا جمله توصیف‌کننده وجود ندارد.]
CashDispenser	هر روز با 500 عدد اسکناس 20 دلاری شروع بکار می‌کند.
DepositeSlot	[کلمه یا جمله توصیف‌کننده وجود ندارد.]

شکل ۲۳-٤ | کلمات و عبارات توصیف‌کننده از مستند نیازهای ATM.

جدول ۲۳-۴ ما را به سمت ایجاد یک صفت کلاس ATM سوق می‌دهد. کلاس ATM مسئول نگهداری اطلاعاتی در ارتباط با وضعیت ATM است. عبارت «تایید هویت کاربر» توصیف‌کننده وضعیتی از ATM است (در بخش۱۱-۵ به معرفی وضعیت‌ها خواهیم پرداخت)، از اینرو **userAuthenticated** را بعنوان یک صفت بولی در نظر گرفته‌ایم (صفتی که دارای یک مقدار true یا **fasle** است). نوع **Boolean** در UML معادل نوع **bool** در زبان C++ است. این صفت بر این نکته دلالت دارد که آیا ATM با موفقیت هویت کاربر جاری را تایید کرده است یا خیر، برای اینکه سیستم به کاربر اجازه انجام تراکنش و دسترسی به اطلاعات حساب را فراهم آورد بایستی ابتدا **userAuthenticated** برابر **true** باشد. این صفت در حفظ امنیت داده‌ها در سیستم نقش مهمی ایفا می‌کند.

کلاس‌های **BalanceInquiry**، **Withdrawal** و **Deposit** یک صفت را به اشتراک می‌گذارند. هر تراکنشی مستلزم یک «شماره‌حساب» است که متناظر با حساب کاربری است که تراکنش را انجام می‌دهد. یک صفت صحیح **accountNumber** به هر کلاس تراکنش برای شناسایی حساب اهدا می‌کنیم.

کلمات و جملات توصیفی در مستند نیازها پیشنهاد برخی از صفات متفاوت و مورد نیاز برای هر کلاس ترکنش را می‌کنند. مستند نیازها بر این نکته دلالت دارد که برای برداشت پول نقد یا سپرده‌گذاری، باید کاربر یک «مقدار» مشخص از پول را برای برداشت یا سپرده‌گذاری مشخص کند. از اینرو، به کلاس‌های **Withedrawal** و **Deposit** یک صفت بنام **amount** تخصیص می‌دهیم تا مقدار مشخص شده از سوی کاربر را در خود ذخیره سازد. میزان پول مرتبط با برداشت پول و سپرده‌گذاری، تعریف‌کننده مشخصه این تراکنش‌ها است که سیستم نیازمند آنها می‌باشد. کلاس **BalanceInquiry** نیازی به داده‌های اضافی برای انجام وظیفه خود ندارد. تنها نیاز این کلاس یک شماره‌حساب است تا براساس آن موجودی حساب را بازیابی کند.

کلاس **Account** دارای چندین صفت است. مستند نیازها مشخص می‌کند که هر حساب بانکی دارای یک «شماره‌حساب» و "PIN" است، که سیستم با استفاده از آن مبادرت به شناسایی حساب و هویت کاربران می‌کند. به کلاس **Account** دو صفت صحیح تخصیص داده‌ایم: **PIN** و **accoutnNumber**. همچنین مستند نیازها تصریح می‌کند که هر حسابی مبادرت به نگهداری «موجودی» از میزان پولی که در حساب وجود دارد و مقدار پولی که کاربر بعنوان سپرده در پاکت وارد سیستم ATM وارد کرده ولی هنوز توسط مامور بانک تایید نشده و چک‌هایی که وارد حساب نشده‌اند، می‌کند. با این همه، باید حساب، میزان موجودی که کاربر سپرده‌گذاری کرده است را ثبت کند. بنابر این، تصمیم گرفته‌ایم که یک حساب باید قادر به نمایش میزان موجودی با استفاده از دو صفت UML از نوع **Double** باشد: **availableBalance** و **totalBalance**. صفت **availableBalance** تعیین‌کننده میزان پولی است که کاربر می‌تواند بصورت نقد از حساب خود برداشت کند. صفت **totalBalance** به کل موجودی اشاره دارد که شامل سپرده‌گذاری نیز می‌شود. برای مثال، فرض کنید یک کاربر ATM مبلغ 50.00 دلار در یک حساب خالی سپرده‌گذاری کرده است. در اینحالت صفت **totalBalance** به 50.00 دلار افزایش می‌یابد تا میزان سپرده ثبت گردد، اما مقدار صفت **availableBalance** هنوز در صفر دلار باقی می‌ماند.

کلاس **CashDispenser** دارای یک صفت است. مستند نیازها مشخص می‌کند که تحویل‌دار خودکار «هر روز کار خود را با 500 قطعه اسکناس 20 دلاری شروع می‌کند.» این تحویل‌دار باید مراقب تعداد اسکناس‌ها موجود باشد تا بتواند تعیین کند که آیا به میزان کافی اسکناس برای پرداخت به تقاضای صورت گرفته در اختیار دارد یا خیر. به کلاس **CashDispenser** یک صفت صحیح به نام **count** تخصیص می‌دهیم. که در ابتدای کار با 500 تنظیم شده است.

در برنامه‌های واقعی هیچ تضمینی وجود ندارد که مستند نیازها به قدر کافی غنی، دقیق و گویا برای طراحان سیستم‌های شی‌گرا باشد تا آنها هم بتوانند تمام صفات یا حتی تمام کلاس‌ها را تعیین کنند. نیاز به کلاس‌ها، صفات و رفتارهای اضافی در فرآیند طراحی خود را آشکار می‌کنند. همانطوری که در این مبحث آموزشی پیش می‌رویم، مبادرت به افزودن، اصلاح و حذف اطلاعاتی در ارتباط با کلاس‌ها در سیستم خود خواهیم کرد.

مدل کردن صفات

در دیاگرام کلاس شکل ۲۴-۴ برخی از صفات کلاس‌های موجود در سیستم ATM به نمایش درآمده‌اند. جدول شکل ۲۳-۴ در شناسایی این صفات به ما کمک کرده است. برای سادگی کار، شکل ۲۴-۴ نمایشگر وابستگی موجود مابین کلاس‌ها نیست که آنها را در شکل ۲۳-۳ قبلاً عرضه کرده بودیم. از بخش‌های قبل بخاطر دارید که در UML، صفات کلاس در بخش میانی دیاگرام کلاس قرار داده می‌شوند. نام هر صفت و نوع آن توسط یک کولن از هم جدا شده و سپس در برخی از موارد یک علامت تساوی و مقدار اولیه هم بعد از آنها آورده می‌شود.

به صفت **userAuthentiated** از کلاس ATM توجه کنید:

```
userAuthenticated : Boolean = false
```

در اعلان این صفت سه نوع داده در ارتباط با آن وجود دارد. نام صفت **userAuthenticated** است. نوع صفت **Boolean** است. در C++، یک صفت را می‌توان توسط یک نوع بنیادین همانند **bool**، **int** یا **double** یا یک نوع کلاس عرضه کرد. در شکل ۲۴-۴ از نوع‌های بنیادین برای صفات استفاده کرده‌ایم. همچنین مقدار اولیه صفت را هم مشخص کرده‌ایم. صفت **userAuthenticated** در کلاس **ATM** دارای مقدار اولیه **false** است. به این معنی که سیستم در ابتدای کار کاربر را تایید نمی‌کند. اگر صفتی دارای مقدار اولیه مشخص شده‌ای نباشد، فقط نام و نوع صفت به نمایش در می‌آیند. برای مثال صفت **accountNumber** از کلاس **BalanceInquiry** از نوع **Integer** است که مقدار اولیه هم ندارد چرا که مقدار این صفت عددی است که هنوز از آن اطلاعی نداریم. این عدد در زمان اجرای برنامه و براساس شماره حساب وارد شده توسط کاربر جاری ATM تعیین می‌شود.

شکل ۲٤-٤ | کلاس‌ها همراه با صفات.

شکل ۲۴-۴ حاوی صفاتی برای کلاس‌های **Screen**، **Keypad** و **DepositSlot** نیست. این اجزاء جزء کامپونت‌های مهم در سیستم هستند که هنوز طراحی ما قادر به تعیین صفات آنها نشده است. با این

وجود در ادامه روند طراحی یا به هنگام پیاده‌سازی این کلاس‌ها در C++ می‌توان به بررسی آنها پرداخت. چنین حالتی در فرآیند مهندسی نرم‌افزار کاملاً طبیعی است.

مهندسی نرم‌افزار

در مراحل اولیه فرآیند طراحی، برخی از کلاس‌ها فاقد صفات (و عملیات) هستند. با این وجود، چنین کلاس‌هایی نباید از نظر دور نگه داشته شوند، چرا که این صفات (و عملیات) می‌توانند خود را در مراحل بعدی طراحی و پیاده‌سازی نشان دهند.

همچنین توجه کنید که در شکل ۲۴-۴ صفتی برای کلاس **BankDatabase** در نظر گرفته نشده است. از فصل سوم بخاطر دارید که در C++، صفات را می‌توان با نوع‌های بنیادین یا نوع‌های کلاس عرضه کرد. چون در مدل کردن دیاگرام‌های کلاس این شکل تصمیم بر استفاده از نوع‌های بنیادین برای صفات گرفته‌ایم، این کلاس فعلاً صفتی ندارد. مدل کردن صفت از نوع کلاس بسیار واضح بوده و همانند یک رابطه (در عمل یک ترکیب) مابین کلاس با صفت است. برای مثال، دیاگرام کلاس در شکل ۲۳-۳ نشان می‌دهد که کلاس **BankDatabase** در یک رابطه ترکیبی با صفر یا چند شی **Account** شرکت دارد. از این ترکیب، می‌توانیم تعیین کنیم که به هنگام پیاده‌سازی سیستم ATM در C++، ملزم به ایجاد صفتی از کلاس **BankDatabase** هستیم که صفر یا بیشتر شی Acount در خود نگهداری کند. به همین ترتیب با کلاس‌های **Screen**، **Keypad**، **CashDispenser** و **DepositSlot** رفتار می‌کنیم. مدل کردن این صفات مبتنی بر ترکیب می‌تواند سبب افزونگی در شکل۲۴-۴ شود، چرا که ترکیب‌های مدل شده در شکل ۲۳-۳ بر این واقعیت تاکید دارند که پایگاه داده حاوی اطلاعاتی در ارتباط با صفر یا بیشتر حساب بوده و ATM مرکب از صفحه‌نمایش،صفحه‌کلید، پرداخت‌کننده‌پول و شکاف‌سپرده‌گذاری است. معمولاً طراحان نرم‌افزاری چنین روابط کامل/ بخش را بصورت ترکیبی بجای صفات مدل‌سازی می‌کنند.

دیاگرام کلاس در شکل ۲۴-۴ ساختار پایه‌ای مدل ما را نشان می‌دهد، اما کامل نیست. در بخش ۱۱-۵ مبادرت به شناسایی وضعیت و فعالیت شی‌ها در مدل کرده و در بخش ۲۲-۶ به بررسی عملیاتی که شی‌ها انجام می‌دهند می‌پردازیم.

تمرینات خودآزمایی مبحث مهندسی نرم‌افزار

۱-۴ عموماً شناسایی صفات کلاس‌ها در سیستم با تحلیل _____در مستند نیازها صورت می‌گیرد.

a) اسامی و جملات

b) کلمات و جملات توصیفی

c) افعال

d) همه گزینه‌های فوق

۴-۲ کدامیک از موارد زیر نشاندهنده صفتی از یک هواپیما نیستند؟

a) طول

b) طول بال هواپیما

c) پرواز

d) تعداد صندلی‌ها

۴-۳ به توضیح مفهوم اعلان صفت کلاس **CashDispenser** در دیاگرام کلاس شکل ۲۴-۴ بپردازید:

```
count : Integer = 500
```

پاسخ خودآزمایی مبحث آموزشی مهندسی نرم‌افزار

۴-۱ b.

۴-۲ c. پرواز یک عمل یا رفتار در هواپیما است و نشاندهنده صفت نمی‌باشد.

۴-۳ به این معنی است که **count** از نوع **Integer** بوده و مقدار اولیه آن 500 می‌باشد. این صفت تعداد اسکناس‌های که هر روز در **CashDispenser** قرار داده می‌شود را کنترل می‌کند.

خودآزمایی

۴-۱ جاهای خالی را در عبارات زیر با کلمات مناسب پر کنید.

a) تمام برنامه‌ها در سه نوع عبارت کنترلی: .....، ..... و ..... نوشته شوند.

b) عبارت انتخاب ..... در صورت برقرار بودن شرط عملی را به اجرا درآورده و در صورت برقرار نبودن عمل دیگری را به اجرا در می‌آورد.

c) تکرار مجموعه‌ای از دستورالعمل‌ها به دفعات مشخص، تکرار ..... نامیده می‌شود.

d) زمانیکه از همان ابتدا تعداد تکرار عبارت مشخص نباشد، یک مقدار ..... می‌تواند به کار گرفته شده و به تکرار خاتمه دهد.

e) مشخص کردن ترتیب اجرای عبارت در یک برنامه کامپیوتری، ..... برنامه نامیده می‌شود.

f) ..... یک زبان مصنوعی و فرمال است که به برنامه‌نویسان در ایجاد الگوریتم‌ها کمک می‌کند.

g) ..... توسط زبان C++ به منظور پیاده سازی ویژگی‌های متفاوتی، نظیر عبارتهای کنترل، رزرو شده‌اند.

h) عبارت انتخابی ..... عبارت چند انتخابی نامیده می‌شود چرا که از میان موارد متفاوت انتخاب خود را انجام می‌دهد.

۴-۲ چهار عبارت متفاوت C++ بنویسید که 1 را به متغیر **x** از نوع صحیح اضافه کند.

۴-۳ عبارت یا مجموعه‌ای از عبارات را برای انجام موارد خواسته شده زیر بنویسید:

a) تخصیص مقدار **x** و **y** به **z** و سپس افزایش **x** به میزان یک واحد پس از انجام محاسبات. فقط با استفاده از یک عبارت.

b) بزرگتر بودن مقدار متغیر **count** را از 10 بررسی کنید. اگر چنین باشد، عبارت "Count is greater than 10" چاپ گردد.

c) متغیر **x** را از 1 کم کنید. سپس آنرا از متغیر **total** تفریق نمائید. فقط از یک عبارت استفاده کنید.

۴-۴ یک عبارت C++ برای انجام موارد خواسته شده زیر بنویسید:

a) اعلان متغیرهای **sum** و **x** از نوع **int**.

b) تخصیص 1 به متغیر **x**.

c) تخصیص 0 به متغیر **sum**.

d) محاسبه مجموع متغیرهای **x** و **sum** و تخصیص نتیجه به متغیر **sum**.

e) چاپ عبارت "**The sum is**:" که بدنبال آن مقدار متغیر **sum** آمده باشد.

۴-۵ با ترکیب عبارات نوشته شده در تمرین ۴-۴ آنرا به فرم برنامه‌ای در آورید که مجموع اعداد از 10 تا 1 را محاسبه و چاپ کند. از یک عبارت **while** برای ایجاد حلقه استفاده کنید. زمانیکه مقدار متغیر **x** به 11 رسید، حلقه پایان پذیرد.

۴-۶ مقدار هر متغیر را پس از هر محاسبه تعیین کنید. فرض کنید تمام متغیرها در ابتدای کار مقدار 5 دارند.

a) `product *= x++;`

b) `quotient /= ++x;`

۴-۷ عباراتی در C++ بنویسید که موارد خواسته شده زیر را انجام دهند.

a) وارد کردن متغیر صحیح **x** با **cin** و <<.

b) وارد کردن متغیر صحیح **y** با **cin** و <<.

c) مقداردهی متغیر صحیح **i** با **1**.

d) مقداردهی متغیر صحیح **power** با **1**.

e) ضرب متغیر **power** در **x** و تخصیص نتیجه به **power**.

f) پس افزایشی کردن متغیر **i** به میزان **1** واحد.

g) تعیین اینکه آیا **i** کوچکتر یا مساوی **y** است یا خیر.

h) چاپ متغیر صحیح **power** با **cin** و <<.

۴-۸ برنامه‌ای در C++ بنویسید که از عبارت تمرین ۴-۷ استفاده کرده و **x** را به توان **y** برساند. این برنامه باید از حلقه تکرار **while** استفاده کند.

۴-۹ خطا یا خطاهای موجود در عبارات زیر را تشخیص داده و اصلاح کنید.

a)

```
while (c <= 5)
      {
```

```
product *= c;
c++;
```

(b

```
cin<< value
```

(c

```
if (gender == 1)
    cout<<"Woman"<<endl;
else;
    cout<<"Man"<<endl;
```

۴-۱۰ در حلقه تکرار **while** عبارت زیر چه اشتباهی وجود دارد؟

```
while( z>= 0)
     sum +=z;
```

پاسخ خودآزمایی

۴-۱ a) توالی، انتخاب، تکرار. b) **if..else**. c) مشخص شده یا تعریف شده. d) مراقبتی، سیگنال یا پرچم. e) کنترل. f) شبه‌کد. g) کلمات کلیدی. h) **switch**.

۴-۲

```
x = x + 1;
x += 1;
++x;
x++;
```

۴-۳

(a

```
z = x++ + y;
```

(b

```
if (count > 10)
        cout<<"Count is greater than 10"<<endl;
```

(c

```
total -= --x;
```

۴-۴

(a

```
int sum;int x;
```

(b

```
x = 1;
```

(c

```
sum = 0;
```

(d

```
sum += x
```
یا
```
sum = sum + x;
```

(e

```
cout<<"The sum is:" <<Sum << endl;
```

۴-۵

```
// Exercise 4.5 Solution: ex04_05.cpp
// Calculate the sum of the integers from 1 to 10.
#include <iostream>
using std::cout;
using std::endl;
```

```
int main()
{
   int sum; // stores sum of integers 1 to 10
   int x; // counter

   x = 1; // count from 1
   sum = 0; // initialize sum

   while ( x <= 10 ) // loop 10 times
   {
      sum += x; // add x to sum
      x++; // increment x
   } // end while

   cout << "The sum is: " << sum << endl;
   return 0; // indicate successful termination
} // end main
```

۴-۶

a) `product = 25, x = 6;`

b) `quotient = 0, x = 6;`

```
// Exercise 4.6 Solution: ex04_06.cpp
// Calculate the value of product and quotient.
#include <iostream>
using std::cout;
using std::endl;

int main()
{
   int x = 5;
   int product = 5;
   int quotient = 5;

   // part a
   product *= x++; // part a statement
   cout << "Value of product after calculation: " << product << endl;
   cout << "Value of x after calculation: " << x << endl << endl;

   // part b
   x = 5; // reset value of x
   quotient /= ++x; // part b statement
   cout << "Value of quotient after calculation: " << quotient << endl;
   cout << "Value of x after calculation: " << x << endl << endl;
   return 0; // indicate successful termination
} // end main
```

۴-۷

a) cin >>x;
b) cin >>y;
c) i = 1;
d) power = 1;
e) power *=x ; or power = power * x ;
f) i++;
g) if (i<=y)
h) count << power << endl;

۴-۸

```
// Exercise 4.8 Solution: ex04_08.cpp
// Raise x to the y power.
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

int main()
{
   int x; // base
   int y; // exponent
   int i; // counts from 1 to y
   int power; // used to calculate x raised to power y

   i = 1; // initialize i to begin counting from 1
   power = 1; // initialize power

   cout << "Enter base as an integer: ";  // prompt for base
   cin >> x; // input base

   cout << "Enter exponent as an integer: "; // prompt for exponent
   cin >> y; // input exponent

   // count from 1 to y and multiply power by x each time
   while ( i <= y )
   {
      power *= x;
      i++;
   } // end while

   cout << power << endl; // display result
   return 0; // indicate successful termination
} // end main
```

٤-۹

a) خطا: فاقد براکت راست (}) در بدنه **while** است.

اصلاح : افزودن براکت راست پس از عبارت **C++;**

b) خطا: استفاده از عملگر درج بجای عملگر استخراج.

اصلاح: تغییر >> به <<.

c) خطا: سیمکولن پس از **else** یک خطای منطقی است. عبارت خروجی دوم همیشه اجرا خواهد شد.

اصلاح: حذف سیمکولن پس از **else**.

۴-۱۰ مقدار متغیر **z** هرگز در حلقه **while** تغییر نمی‌کند. از اینرو اگر شرط تکرار حلقه (**z >= 0**) در بدو امر برقرار باشد (**true**), یک حلقه بی‌نهایت بوجود خواهد آمد. برای اجتناب ازحلقه بی‌نهایت, بایستی **z** کاهش یابد تا سرانجام از صفر کمتر شود.

تمرینات

۴-۱۱ خطا یا خطاهای موجود در عبارت زیر را تشخیص داده و اصلاح کنید:

```
a) if (age >= 65);
       cout <<  "Age is greater than or equal to 65" << endl;
   else
       cout << "Age is less than to 65 << endl;"

b) if ( age >= 65);
      cout << "Age is greater than or equal to 65" << endl;
   else;
      cout << "Age is less than to 65" << endl";

c) int x = 1, total;

   while ( x <= 10)
  {
    total +=x;
    x++;
  }

d) while ( x <= 100 )
     total += x;
     x++;

e) while ( y > 0 )
   {
      cout << y << endl;
      y++;
   }
```

۴-۱۲ برنامه زیر چه عبارتی را چاپ می‌کند؟

```
// Exercise 4.12: ex04_12.cpp
// What does this program print?
#include <iostream>
using std::cout;
using std::endl;

int main()
{
   int y; // declare y
   int x = 1; // initialize x
   int total = 0; // initialize total

   while ( x <= 10 ) // loop 10 times
   {
      y = x * x; // perform calculation
      cout << y << endl; // output result
      total += y; // add y to total
      x++; // increment counter x
   } // end while

   cout << "Total is " << total << endl; // display result
   return 0; // indicate successful termination
} // end main
```

برای تمرینات ۱۳-٤ تا ۱٦-٤ هر یک از مراحل زیر را انجام دهید:

a) خواندن صورت مسئله.

b) فرموله کردن الگوریتم با شبه کد و مرحله اصلاح گام به گام از بالا به پایین.

c) نوشتن برنامه به زبان++C .

d) تست خطا یابی و اجرای برنامه.

۱۳-۴ راننده‌گان علاقه‌مند به دانستن مسافت طی شده توسط اتومبیل خود هستند. راننده‌ای مبادرت به ثبت تعداد دفعات سوخت‌گیری, میزان سوخت و مسافت پیموده شده می‌کند. برنامه‌ای در C++ بنویسد که با استفاده از یک حلقه while مبادرت به دریافت مسافت طی شده و تعداد گالون‌های زده شده در هر بار سوخت گیری کند. برنامه باید مسافت طی شده (مایل طی شده) بر حسب هر گالون را محاسبه کرده و بنمایش در آورد. خروجی برنامه می‌تواند شبیه عبارات زیر باشد.

```
Enter the miles used(-1 to quit):287
Enter gallons:13
MPG this tankful:22.076923
Total MPG:2.0769923

Enter the miles used(-1 to quit):200
Enter gallons:10
MPG this tankful:20.000000
Total MPG:21.173913

Enter the miles used(-1 to quit):-1
(MPG: Miles Per Gallon)
```

۱۴-۴ برنامه‌ای بنویسید که اگر مشتری یک فروشگاه بیش از موجودی خود در کارت اعتباری سفارش دهد، مشخص گردد. برای هر مشتری اطلاعات زیر موجود هستند:

۱- شماره حساب

۲- میزان موجودی در ابتدای ماه

۳- مجموع سفارشات از طرف مشتری در این ماه

۴- مجموع اعتبارات بکار برده شده توسط مشتری در این ماه.

۵- حد اعتبار

برنامه باید هر کدامیک از موارد فوق را دریافت و اعتبار جدید مشتری را محاسبه کند (= میزان موجودی + سفارشات - اعتبار) و مشخص نماید که آیا سفارش مشتری بیش از اعتبارش است یا خیر. در صورت سفارش بیش از حد پیغام "**Credit limit exceeded**" چاپ شود. خروجی برنامه می‌تواند همانند عبارات زیر باشد.

```
Enter account number(-1 to end):100
Enter beginning balance:5394.78
Enter total credits:500.00
Enter credit limit:5500.00
New balance is 5894.78
Account:100
Credit limit:5500.00
Balance:5894.78
```

```
Credit Limit Exceeded

Enter account number(-1 to end):200
Enter beginning balance:1000.00
Enter total charges:123.45
Enter credit limit:1500.00
New balance is 802.45

Enter account number(-1 to end):300
Enter beginning balance:500.00
Enter total charges:274.73
Enter total credits:100.00
Enter credit limit:800.00
New balance is 674.73
```

۴-۱۵ یک شرکت بزرگ شیمیایی براساس کمیسیون، حقوق فروشندگان خود را پرداخت می‌کند. هر فروشنده‌ای، در هفته 200 دلار به همراه 9 درصد از فروش هفته دریافت می‌کند. برای مثال، فروشنده‌ای که 5000 دلار در هفته فروش داشته، 200 دلار به همراه 9 در صد از 5000 دلار یا مجموع 650 دلار دریافت می‌کند. برنامه‌ای بنویسید که از یک عبارت **while** برای دریافت فروش هفتگی هر فروشنده استفاده کرده و دریافتی وی را به نمایش در آورد. در هر بار، حقوق یک فروشنده را محاسبه نماید.

```
Enter sales in dollars(-1 to end):5000.00
Salary is:$650.00
Enter sales in dollars(-1 to end):6000.00
Salary is:$740.00
Enter sales in dollars(-1 to end):-1
```

۴-۱۶ برنامه‌ای در C++ بنوسید که با استفاده از یک عبارت while حقوق دریافتی چند کارمند را محاسبه کند. شرکت بر اساس 40 ساعت کار در هفته حقوق مستقیم پرداخت می‌کند و اگر بیش از 40 ساعت کار صورت گرفته باشد، نصف آن زمان به حقوق 40 ساعتی افزوده می‌شود. برنامه اطلاعات هر کارمند را که شامل ساعت کار کرده در هفته بوده به همراه نرخ دستمزد ساعتی را دریافت کرده و حقوق وی را محاسبه می‌کند. خروجی برنامه می‌تواند همانند عبارات زیر باشد.

```
Enter hours worked(-1 to end):39
Enter hourly rate of the worker($00.00):10.00
Salary is $390.00

Enter hours worked(-1 to end):40
Enter hourly rate of the worker($00.00):10.00
Salary is $400.00

Enter hours worked(-1 to end):41
Enter hourly rate of the worker($00.00):10.00
Salary is $415.00

Enter hours worked(-1 to end):-1
```

۴-۱۷ فرآیند یافتن بزرگترین عدد از جمله برنامه‌های پرکاربرد است. برای مثال، برنامه‌ای می‌تواند بهترین فروشنده را بر اساس میزان فروش تعیین کند. کسی که بیشترین فروش را داشته است، به عنوان برنده انتخاب می‌شود.

شبه‌کدی نوشته، سپس برنامه C++ آنرا ایجاد کنید که با استفاده از عبارت **while** مبادرت به تعیین و چاپ بزرگترین عدد از بین **10** عدد وارد شده توسط کاربر کند. برنامه باید از سه متغیر به شرح زیر استفاده نماید:

counter: یک شمارنده برای شمارش تا **10**

number: عدد جاری وارد شده به برنامه.

largest: بزرگترین عدد دریافت شده تا بدین جا.

۴-۱۸ برنامه‌ای بنویسید که با استفاده از عبارت **while** اقدام به چاپ مقادیر جدول زیر کند:

N	10*N	100*N	1000*N
1	10	100	1000
2	20	200	2000
3	20	300	3000
4	40	400	4000
5	50	500	5000

۴-۱۹ با استفاده از روش بکار رفته در تمرین ۴-۱۷، دو عدد بزرگ را در میان 10 عدد پیدا کنید (نکته: باید هر عدد را یک بار وارد کنید.)

۴-۲۰ برنامه بررسی نتایج آزمون در شکل‌های ۴-۱۶ الی ۴-۱۸ فرض می‌کند که هر مقدار ورودی که توسط کاربر وارد می‌شود اگر 1 نباشد پس 2 است. برنامه را برای اعتبارسنجی ورودی‌ها اصلاح کنید. اگر مقدار وارد شده 1 یا 2 نباشد، حلقه تا دریافت مقدار صحیح تکرار شود.

۴-۲۱ برنامه زیر چه عبارتی را چاپ می‌کند؟

```
// Exercise 4.21: ex04_21.cpp
// What does this program print?
#include <iostream>
using std::cout;
using std::endl;

int main()
{
   int count = 1; // initialize count

   while ( count <= 10 ) // loop 10 times
   {
      // output line of text
      cout << ( count % 2 ? "****" : "++++++++" ) << endl;
      count++; // increment count
   } // end while

   return 0; // indicate successful termination
} // end main
```

۴-۲۲ برنامه زیر چه عبارتی را چاپ می‌کند؟

```
// Exercise 4.22: ex04_22.cpp
// What does this program print?
#include <iostream>
using std::cout;
using std::endl;

int main()
{
   int row = 10; // initialize row
   int column; // declare column

   while ( row >= 1 ) // loop until row < 1
   {
      column = 1; // set column to 1 as iteration begins

      while ( column <= 10 ) // loop 10 times
      {
         cout << ( row % 2 ? "<" : ">" ); // output
         column++; // increment column
      } // end inner while

      row--; // decrement row
      cout << endl; // begin new output line
   } // end outer while

   return 0; // indicate successful termination
} // end main
```

۴-۲۳ مشکل Danglin-Else. خروجی هر یک از عبارات زیر را با فرض زمانیکه x برابر 9، y برابر 11 و زمانیکه x برابر 11 و y برابر 9 است را تعیین کنید. توجه کنید که کامپایلر دندانه‌گذاری موجود در یک برنامه C++ را نادیده می‌گیرد. کامپایلر C++ همیشه یک else را با if قبلی در نظر می‌گیرد مگر اینکه با قرار دادن براکت‌ها {} غیر این را حکم کنید. در نگاه اول، برنامه‌نویس نمی‌تواند مطمئن باشد که کدام if و else با هم هستند، از اینرو این مشکل Dangling-else شناخته می‌شود. برای اینکه حل مسئله کمی مشکل‌ت شود، دندانه‌گذاری را در این عبارات اعمال نکرده‌ایم.

```
a)
if (x<10)
if (y>10)
cout << "*****"<<endl;
else
cout <<"#####"<<endl;
cout <<"$$$$$"<<endl;

b)
if (x<10)
{
if (y>10)
cout <<"*****"<<endl;
}
else
{
```

```
cout <<"#####"<<endl;
cout <<"$$$$$"<<endl;
}
```

۴-۲۴ (مشکل Dangling-Else) کد زیر را برای تولید خروجی به نمایش در آمده اصلاح کنید. از دندانه‌گذاری مناسب استفاده کنید. هیچ تغییری بجز اعمال براکت نباید در کد بوجود آورید.

```
if (y == 8)
if (x == 5)
cout << "@@@@@" << endl;
else
cout << "#####" << endl;
cout << "$$$$$" << endl;
cout << "&&&&&" << endl;
```

a) با فرض x=5 و y=8 ،خروجی زیر تولید شود:

```
@@@@@
$$$$$
&&&&&
```

b) با فرض x=5 و y=8، خروجی زیر تولید شود:

```
@@@@@
```

c) با فرض x=5 و y=8، خروجی زیر تولید شود:

```
@@@@@
&&&&&
```

d) با فرض x=5 و y=7، خروجی زیر تولید شود:

```
#####
$$$$$
&&&&&
```

۴-۲۵ برنامه‌ای بنویسید که سایز یک ضلع چهارگوش را دریافت و یک چهارگوش توخالی براساس آن سایز از ستاره‌ها (*) و فاصله‌ها چاپ کند. برنامه باید برای ترسیم چهارگوش‌های با سایز 1 تا 20 عمل کند. برای مثال، اگر سایز 5 وارد برنامه شود، بایستی خروجی زیر چاپ شود.

```
*****
*   *
*   *
*   *
*****
```

۴-۲۶ پالندروم، عدد یا عبارتی است که خواندن آن از هر دو جهت یکسان است. برای مثال، اعداد پنج رقمی و از نوع صحیح زیر همگی پالندروم هستند: 12321، 55555، 45554، 11611. برنامه‌ای بنویسید که پنج رقم از نوع صحیح دریافت کرده و تعیین کند که آیا پالندروم است یا خیر.

۴-۲۷ یک عدد صحیح فقط حاوی صفرها و یک‌ها (یعنی باینری) دریافت کرده و معادل دیسمال آنرا چاپ کنید. از عملگر باقیمانده و تقسیم برای انتخاب ارقام باینری از سمت راست به چپ استفاده کنید (در هر بار یک رقم). با توجه به اینکه در سیستم عددی دیسمال، سمت راست‌ترین رقم دارای ارزش مکانی 1، رقم بعدی دارای ارزش مکانی 10، سپس 100، سپس 1000 و الی آخر است. در سیستم عددی باینری، سمت راست‌ترین رقم دارای ارزش مکانی 1، رقم بعدی دارای ارزش مکانی 2، سپس 4، سپس 8 و الی آخر است. از اینرو عدد دیسمال 234 می‌تواند

بصورت 1*4 + 3*10 + 2*100 تفسیر شود. معادل دیسمال عدد باینری 1101 است که بصورت 1*4 + 0*2 + 1*1 + 1*8 یا 8 + 4 + 0 + 1 یا 13 است.

۴-۲۸ برنامه بنویسید که الگوی زیر را به نمایش در آورد. برنامه باید از سه عبارت خروجی استفاده کرده باشد، یکی از عبارات می‌تواند بفرم زیر باشد:

```
   cout << "*" ;
   cout << '  ' ;
   cout << endl ;
********
 ********
********
 ********
********
 ********
********
 ********
```

۴-۲۹ برنامه‌ای بنویسید که توالی‌هایی از 2 را بصورت 2,4,8,16,32,64 الی آخر تولید کرده و چاپ نماید. حلقه **while** نباید خاتمه یابد (یعنی یک حلقه بی‌نهایت ایجاد کنید). برای انجام اینکار، کافیست در شرط عبارت **while** کلمه کلیدی **true** را قرار دهید. با انجام اینکار چه اتفاقی رخ خواهد داد؟

۴-۳۰ برنامه‌ای بنویسید که شعاع یک دایره را دریافت (از نوع **double**) و قطر، مساحت و محیط آن را محاسبه کنید. از مقدار 3.14159 برای π استفاده کنید.

۴-۳۱ در عبارت زیر چه اشتباهی وجود دارد؟ عبارت زیر را به نحوی اصلاح کنید که خواسته برنامه‌نویس را برآورده سازد.

```
   cout << ++(x+y);
```

۴-۳۲ برنامه‌ای بنویسید که مقدار double غیرصفرخوانده و تعیین کند که این مقادیر می‌توانند نشاندهنده اضلاع یک مثلث باشند یا خیر.

۴-۳۳ برنامه‌ای بنویسید که مقدار صحیح غیرصفر خوانده و تعیین کند که این مقادیر می‌توانند نشاندهنده یک مثل راست گوشه باشند یا خیر.

۴-۳۴ شرکتی می‌خواهد داده‌های خود را از طریق خط تلفن منتقل نماید. تمام داده‌های انتقالی از چهار رقم صحیح تشکیل شده‌اند. برنامه‌ای بنویسید که داده‌های انتقالی این شرکت را بصورت کد درآورد. ابتدا برنامه یک عدد چهار رقمی صحیح را خوانده و سپس بطریق زیر آنرا کدگذاری نماید: هر رقم را با باقیمانده تقسیم بر 10 جایگزین سازد. سپس مکان رقم اول را با رقم سوم و رقم چهارم را با رقم دوم عوض کند. در پایان عدد کدگذاری شده را چاپ کند. در ادامه برنامه‌ای بنویسید که عدد کدگذاری شده را دریافت و آنرا کدگشایی نماید.

۴-۳۵ فاکتوریل n عدد صحیح غیرمنفی بصورت !n نوشته می‌شود و بصورت زیر تعریف می‌گردد:

`n! = n.(n - 1).(n - 2).....1` (n = مقادیر بزرگتر یا مساوی 1)

و

n!= 1 (برای n = 0)

برای مثال 5.4.3.2.1 = !5 است که حاصل آن 120 است.

a) برنامه‌ای بنویسید که یک مقدار صحیح غیرمنفی را دریافت (از طریق کادر تبادلی) و فاکتوریل آنرا محاسبه و چاپ کند.

b) برنامه‌ای بنویسید که مقدار ثابت ریاضی e را با استفاده از فرمول زیر تخمین بزند:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots$$

c) برنامه‌ای بنویسید که مقدار e^x را با استفاده از فرمول زیر محاسبه کند:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$