فصل پنجم

عبارات کنترلی: بخش ۲

اهداف

- **آشنایی با عبارات تکرار for، do...while و اجرای عبارات تکرار شونده.**
- **اصول شمارنده کنترل تکرار.**
- **استفاده از عبارت چند انتخابی switch.**
- **استفاده از عبارات کنترل برنامه break و continue.**
- **استفاده از عملگرهای منطقی.**
- **اجتناب از پی‌آمد اشتباه گرفتن عملگر تخصیص با تساوی.**

رئوس مطالب

۵-۱ مقدمه

۵-۲ نکاتی در مورد شمارنده-کنترل تکرار

۵-۳ عبارت تکرار for

۵-٤ مثال‌های با استفاده از عبارت for

۵-۵ عبارت تکرار do...while

۵-٦ عبارت چند انتخابی switch

۵-۷ عبارات break و continue

۵-۸ عملگرهای منطقی

۵-۹ اشتباه گرفتن عملگر تساوی (==) و عملگر تخصیص (=)

۵-۱۰ چکیده برنامه‌نویسی ساخت‌یافته

۵-۱۱ مبحث آموزشی مهندسی نرم‌افزار: شناسایی وضعیت و فعالیت شی‌ها در سیستم ATM

۵-۱ مقدمه

فصل چهارم را با معرفی انواع بلوک‌های سازنده که در حل مسئله نقش دارند، آغاز کردیم. با استفاده از این بلوک‌های سازنده تکنیک‌های ایجاد برنامه را بهبود بخشیدیم. در این فصل، مبحث تئوری و قواعد علمی برنامه‌نویسی ساخت‌یافته را با معرفی مابقی عبارات کنترلی C++ ادامه می‌دهیم. با عبارات کنترلی که در این فصل مطرح می‌کنیم و عبارات کنترلی معرفی شده در فصل قبلی قادر به ایجاد و کنترل شی‌ها خواهیم بود. همچنین به مبحث برنامه‌نویسی شی‌گرا که آن را از فصل اول آغاز کرده‌ایم ادامه می‌دهیم.

در این فصل به توصیف عبارات **do...while**, **for** و **switch** می‌پردازیم. در کنار مثال‌های کوچکی که در آنها از **while** و **for** استفاده شده، به بررسی اصول و نیازهای شمارنده-کنترل‌تکرار خواهیم پرداخت. بخشی از این فصل را اختصاص به گسترش کلاس **GradeBook** عرضه شده در فصل‌های سوم و چهارم داده‌ایم. در واقع، نسخه‌ای از کلاس **GradeBook** را ایجاد می‌کنیم که از عبارت **switch** برای شمارش تعداد نمرات A، B، C، D و F وارد شده از سوی کاربر استفاده می‌کند. به معرفی عبارات **break** و**continue** خواهیم پرداخت که کنترل‌کننده برنامه هستند. در مورد عملگرهای منطقی، که به برنامه‌نویسان اجازه می‌دهند تا از شرط‌های پیچیده و قدرتمندتر در عبارات کنترلی استفاده کنند، صحبت خواهیم کرد.

همچنین در ارتباط با خطای رایجی که در ارتباط با عدم درک صحیح تفاوت مابین عملگر برابری (==) و تخصیص (=) رخ می‌دهد توضیحاتی ارائه می‌کنیم. در پایان، بطور خلاصه شده به توصیف عبارات کنترلی C++ و تکنیک‌های حل مسئله مطرح شده در این فصل و فصل چهارم می‌پردازیم.

۵-۲ نکاتی در مورد شمارنده-کنترل تکرار

در فصل قبل، با مفهوم روش شمارنده-کنترل تکرار آشنا شدید. در این بخش با استفاده از عبارت تکرار while، اقدام به فرموله کردن عناصر مورد نیاز در روش شمارنده-کنترل تکرار می‌کنیم:

۱- نام ***متغیر کنترل*** (یا شمارنده حلقه) که برای تعیین تکرار حلقه بکار گرفته می‌شود.

۲- ***مقدار اولیه*** متغیر کنترل.

۳- ***شرط تکرار حلقه*** برای تست مقدار نهایی متغیر کنترل (آیا حلقه ادامه یابد یا خیر).

۴- ***افزایش*** (یا ***کاهش***) متغیر کنترل در هر بار تکرار حلقه.

در برنامه شکل ۱-۵ از چهار عنصر شمارنده-کنترل تکرار برای نمایش ارقام 1-10 استفاده شده است. در خط 9، نام *متغیر کنترلی* **(counter)** از نوع صحیح اعلان شده و فضای در حافظه برای آن رزرو می‌شود و با *مقدار اولیه* 1 تنظیم شده است. این اعلان یک مقداردهی اولیه است. بخش مقداردهی این عبارت یک جزء اجرائی است و از اینرو، خود عبارت هم اجرائی می‌باشد.

اعلان و مقداردهی اولیه **counter** در خط 9 را می‌توان توسط عبارات زیر هم انجام داد:

```
int counter; // declare control variable
counter = 1; // initialize control variable to 1
```

ما از هر دو روش استفاده می‌کنیم.

به عبارت **while** (خطوط 15-11) توجه کنید. خط 14 مقدار جاری **counter** را به میزان 1 واحد در هر بار تکرار حلقه افزایش می‌دهد. شرط تکرار حلقه (خط 11) در عبارت **while** تست می‌کند که آیا مقدار متغیر کنترل کمتر یا برابر 10 است یا خیر، به این معنی که 10، *مقدار نهایی* برای برقرار بودن شرط است. بدنه عبارت **while** تا زمانیکه متغیر کنترل به 5 برسد اجرا می‌شود. زمانیکه مقدار متغیر کنترل بیش از 10 شود (هنگامی که **counter** به مقدار 11 برسد)، حلقه پایان می‌یابد.

```
1  // Fig. 5.1: fig05_01.cpp
2  // Counter-controlled repetition.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  int main()
8  {
9     int counter = 1; // declare and initialize control variable
10
```

```
11    while ( counter <= 10 ) // loop-continuation condition
12    {
13       cout << counter << " ";
14       counter++; // increment control variable by 1
15    } // end while
16
17    cout << endl; // output a newline
18    return 0; // successful termination
19 } // end main
```

```
 1 2 3 4 5
```

شکل ۱-۵ | شمارنده-کنترل تکرار.

برنامه‌نویسی ایده‌ال

قبل و بعد از هر عبارت یک خط خالی قرار دهید تا قسمت‌های متفاوت برنامه از یکدیگر تمیز داده شوند.

برنامه‌نویسی ایده‌ال

قراردادن فضاهای خالی در ابتدا و انتهای عبارتهای کنترلی و دندانه‌دار کردن این عبارتها به برنامه یک ظاهر دوبعدی می‌دهد و باعث افزایش خوانایی برنامه می‌شود.

با مقداردهی **counter** با 0 و جایگزین کردن عبارت **while** با

```
while( ++counter <= 10 ) //loop-continuation condition
   cout << counter << " ";
```

می‌توان برنامه شکل ۱-۵ را مختصرتر کرد. با این کد از یک عبارت صرفه‌جویی شده است، چرا که عملیات افزایش بصورت مستقیم در شرط **while** قبل از بررسی شرط صورت می‌گیرد. همچنین این کد نیاز به استفاده از براکت‌ها در اطراف بدنه **while** را از بین برده است، چرا که **while** فقط حاوی یک عبارت است. گاهی اوقات فشرده‌سازی کد به این روش می‌تواند خوانایی، نگهداری، اصلاح و خطایابی برنامه را مشکل کند.

برنامه‌نویسی ایده‌ال

با افزایش تعداد سطح‌های تودرتو، درک عملکرد برنامه مشکل‌تر می‌شود. بعنوان یک قانون، سعی کنید از سه سطح تودرتو فراتر نروید.

خطای برنامه‌نویسی

مقادیر اعشاری تقریبی هستند، از اینرو کنترل شمارش حلقه‌ها با متغیرهای اعشاری می‌تواند در شمارش مقادیر، غیردقیق بوده و شرط خاتمه را بدقت انجام ندهند.

اجتناب از خطا

شمارش حلقه‌ها را با مقادیر صحیح کنترل کنید.

۳-۵ عبارت تکرار for

در بخش ۲-۵ به بررسی اصول شمارنده-کنترل تکرار پرداخته شد. می‌توان از عبارت**while** در پیاده‌سازی هر حلقه کنترل شده با شمارنده استفاده کرد. زبان C++ دارای عبارت تکرار **for** است که از تمام جزئیات

شمارنده-کنترل‌تکرار استفاده می‌کند. برای نشان دادن قدرت **for** برنامه ۱-۵ را مجدداً بازنویسی می‌کنیم. نتیجه اینکار در برنامه شکل ۲-۵ دیده می‌شود.

```
1  // Fig. 5.2: fig05_02.cpp
2  // Counter-controlled repetition with the for statement.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  int main()
8  {
9     // for statement header includes initialization,
10    // loop-continuation condition and increment.
11    for ( int counter = 1; counter <= 10; counter++ )
12       cout << counter << " ";
13
14    cout << endl; // output a newline
15    return 0; // indicate successful termination
16 } // end main
```

```
1 2 3 4 5 6 7 8 9 10
```

شکل ۲-۵ | شمارنده-کنترل‌تکرار با عبارت for.

هنگامی که عبارت **for** شروع بکار می‌کند (خطوط 12-11)، متغیر کنترل **counter** با 1 مقداردهی می‌شود، از اینرو تا بدین جا دو عنصر اولیه شمارنده-کنترل تکرار یعنی *نام متغیر* و *مقدار اولیه* تعیین شده‌اند. سپس، شرط ادامه حلقه **counter <= 10** بررسی می‌شود. بدلیل اینکه مقدار اولیه متغیر **counter**، برابر 1 است، شرط برقرار بوده، و مقدار1 با اجرای عبارت خط 12 چاپ می‌شود. سپس مقدار متغیر **counter** توسط عبارت **counter++** افزایش یافته و مجددا حلقه با تست شرط تکرار آغاز می‌شود. در این لحظه، متغیر کنترل حاوی مقدار 2 است. این مقدار متجاوز از مقدار پایانی نیست و از اینرو برنامه عبارت موجود در بدنه را به اجرا در می‌آورد. این فرآیند تا زمانیکه **counter** به مقدار 10 برسد و آنرا چاپ کند و متغیر کنترل **counter** به 11 افزایش یابد، ادامه پیدا می‌کند و موجب می‌شود تا تست شرط حلقه برقرار نشده و تکرار خاتمه یابد. برنامه با اجرای اولین عبارت پس از عبارت **for** ادامه می‌یابد (در این مثال، برنامه به عبارت خروجی در خط 14 می‌رسد و آنرا اجرا می‌کند).

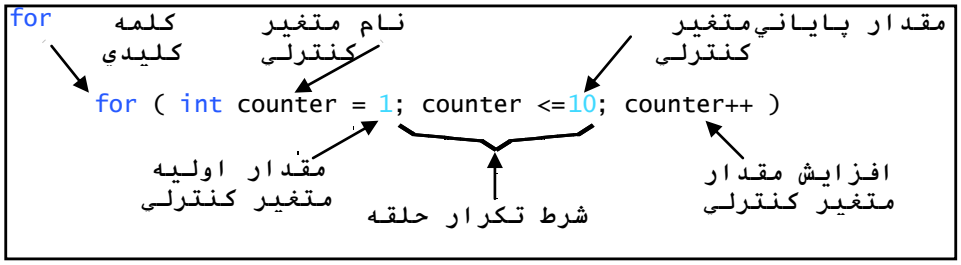
کامپونت‌های تشکیل دهنده سرآیند for

در شکل۳-۵ نگاهی دقیق‌تر بر عبارت **for** برنامه ۲-۵ داشته‌ایم (خط 11). گاهاً به خط اول عبارت **for** *سرآیند for* می‌گویند. این سرآیند حاوی ایتم‌های مورد نیاز در ایجاد شمارنده-کنترل‌تکرار به همراه یک متغیر کنترلی است.

دقت کنید که در برنامه۲-۵ از شرط تکرار حلقه **counter <= 10** استفاده شده است. اگر برنامه‌نویس اشتباهاً بنویسد **counter < 10**، حلقه فقط 9 بار اجرا خواهد شد. این خطا معروف به خطای *off-by-one* است.

خطای برنامه‌نویسی

استفاده از عملگر رابطه‌ای اشتباه یا استفاده از یک مقدار نهایی اشتباه در شرط شمارنده یک حلقه ***while*** *یا* ***for*** *می‌تواند خطای off-by-one بدنبال داشته باشد.*



شکل ۳-۵ | کامپونت‌های سرآیند for.

برنامه‌نویسی ایده‌ال

استفاده از مقدار نهایی در شرط یک عبارت ***while*** *یا* ***for*** *و استفاده از عملگر رابطه‌ای => می‌تواند جلوی خطاهای off-by-one را بگیرید. برای مثال، در حلقه‌ای که برای چاپ مقادیر 1 تا 10 کاربرد دارد، شرط تکرار حلقه بایستی* ***counter <= 10*** *بجای* ***counter < 10*** *یا* ***counter < 11*** *باشد. برخی از برنامه‌نویسان ترجیح می‌دهند شمارش را از صفر آغاز کنند، در اینحالت برای شمارش 10 بار تکرار حلقه، بایستی متغیر* ***counter*** *با صفر مقداردهی شده و شرط تست حلقه به صورت* ***counter < 10*** *نوشته شود.*

فرمت کلی عبارت **for** بشکل زیر است

for (*مقداردهی اولیه; شرط تکرار حلقه; افزایش***)**
عبارت

در این عبارت، جمله مقداردهی اولیه نشاندهنده نام متغیر کنترلی حلقه بوده و مقدار اولیه را فراهم می‌آورد، شرط تکرار حلقه حاوی مقدار پایانی متغیر کنترلی بوده و در صورت برقرار بودن حلقه به اجرا در می‌آید، و جمله افزایش مبادرت به افزودن مقدار متغیر کنترلی می‌کند. می‌توان بجای عبارت **for** از معادل عبارت **while** و بصورت زیر استفاده کرد:

مقداردهی اولیه;
while (*شرط تکرار حلقه***) {**
عبارت
افزایش;
}

در اینجا فقط یک استثناء وجود دارد که در بخش ۷-۵ به بررسی آن خواهیم پرداخت.

اگر جمله مقداردهی اولیه در سرآیند عبارت **for** اعلان‌کننده متغیرکنترلی باشد (نوع متغیرکنترلی قبل از نام متغیر آمده باشد)، می‌توان از آن متغیر کنترلی فقط در بدنه همان **for** استفاده کرد. این متغیر کنترلی در خارج از عبارت **for** شناخته شده نخواهد بود. این محدودیت استفاده از نام متغیر کنترلی بعنوان *قلمرو* متغیر شناخته می‌شود. قلمرو یک متغیر تعیین‌کننده مکانی است که متغیر می‌تواند در برنامه بکار گرفته شود. در فصل ششم با جزئیات قلمرو آشنا خواهید شد.

خطای برنامه‌نویسی

*زمانیکه متغیر کنترلی در یک عبارت **for** در بخش مقداردهی اولیه سرآیند **for** اعلان شده باشد، استفاده از آن متغیر کنترلی پس از عبارت، یک خطای کامپایل بدنبال خواهد داشت.*

قابلیت حمل

*در C++ استاندارد، قلمرو متغیر کنترلی اعلان شده در بخش مقداردهی اولیه یک عبارت **for** با قلمرو مطرح شده در کامپایلرهای قدیمی C++ متفاوت است. در کامپایلرهای قدیمی، قلمرو متغیر کنترلی با پایان یافتن بلوک تعیین کننده عبارت **for** خاتمه نمی‌پذیرد. کد C++ ایجاد شده با کامپایلرهای قدیمی C++ می‌تواند به هنگام کامپایل با کامپایلرهای استاندارد با مشکل مواجه شود. اگر در حال کار با کامپایلرهای قدیمی هستید و می‌خواهید از عملکرد دقیق کدهای خود بر روی کامپایلرهای استاندارد مطمئن شوید، دو استراتژی برنامه‌نویسی تدافعی وجود دارد که می‌توانید از آنها استفاده کنید: اعلان متغیرهای کنترلی با اسامی مختلف در هر عبارت **for** یا اگر ترجیح می‌دهید از نام یکسانی برای متغیر کنترلی در چندین عبارت **for** استفاده کنید، اعلان متغیر کنترلی قبل از اولین عبارت **for** است.*

همانطوری که مشاهده خواهید کرد، جملات مقداردهی اولیه و افزایش را می‌توان با کاما از یکدیگر متمایز کرد. کامای (ویرگول) بکار رفته در این جملات، از نوع عملگرهای کاما هستند و تضمین می‌کنند که لیست جملات از سمت چپ به راست ارزیابی خواهند شد. عملگر کاما از تمام عملگرهای C++ از تقدم پایین‌تری برخوردار است. مقدار و نوع یک لیست جدا شده با کاما، معادل مقدار و نوع سمت راست‌ترین جمله در لیست خواهد بود. در اکثر مواقع از کاما در عبارات **for** استفاده می‌شود. اصلی‌ترین کاربرد کاما این است که برنامه‌نویس امکان استفاده از چندین جمله مقداردهی اولیه و یا چندین جمله افزایش‌دهنده را فراهم می‌آورد. برای مثال، امکان دارد چندین متغیر کنترلی در یک عبارت **for** وجود داشته باشند که بایستی مقداردهی اولیه شده و افزایش داده شوند.

مهندسی نرم‌افزار

با قرار دادن یک سیمکولن بلافاصله پس از سرآیند for، یک حلقه تاخیردهنده بوجود می‌آید. چنین حلقه‌ای با یک بدنه تهی فقط به تعداد دفعات مشخص شده تکرار شده و کاری بجز شمارش انجام نمی‌دهد. برای مثال، امکان دارد از یک حلقه تاخیری برای کاستن از سرعت قراردادن خروجی در صفحه نمایش استفاده کنید تا

کاربر بتواند براحتی آن را بخواند. اما باید مراقب باشید، چرا که چنین زمان تاخیری در میان انواع سیستم‌ها با پردازنده‌های مختلف متفاوت است.

سه عبارت در عبارت **for** حالت اختیاری دارند. اگر شرط تکرار حلقه فراموش شود، C++ فرض خواهد کرد که شرط تکرار حلقه برقرار است، و از اینرو یک حلقه بی‌نهایت بوجود خواهد آمد. می‌توان عبارت مقداردهی اولیه متغیر کنترلی را از عبارت **for** خارج کرد، اگر این متغیر در جای دیگری از برنامه و قبل از حلقه مقداردهی شده باشد. عبارت سوم مربوط به بخش افزایش است و در صورتیکه عملیات افزایش مقدار متغیر کنترلی توسط عبارتی در بدنه **for** صورت گیرد یا نیازی به افزایش وجود نداشته باشد، می‌توان آنرا از سرآیند حذف کرد. عبارت افزایشی در عبارت **for** همانند یک عبارت منفرد در انتهای بدنه **for** عمل می‌کند. از اینرو، عبارات

```
counter = counter + 1
counter += 1
++counter
counter++
```

همگی معادل بخش افزایشی در سرآیند عبارت **for** هستند. بسیاری از برنامه‌نویسان ترجیح می‌دهند تا از **counter++** استفاده کنند. به این دلیل که عملیات افزایش مقدار متغیر کنترلی را پس از اجرای بدنه حلقه به انجام می‌رساند، و از اینرو این رفتار در افزایش بسیار طبیعی بنظر می‌رسد.

خطای برنامه‌نویسی

*استفاده از کاما بجای سیمکولن در سرآیند عبارت **for** خطای نحوی خواهد بود.*

خطای برنامه‌نویسی

*قرار دادن یک سیمکولن بلافاصله در سمت راست پرانتز یک سرآیند **for** یک بدنه تهی برای **for** تولید خواهد کرد. اینکار می‌تواند خطای منطقی بدنبال داشته باشد.*

بخش‌های مقداردهی اولیه، شرط تکرار حلقه و افزایش در عبارت **for** می‌توانند حاوی عبارات محاسباتی باشند. برای مثال، فرض کنید که $x = 2$ و $y = 10$ باشد. اگر x و y در داخل بدنه حلقه تغییری نیابد، پس عبارت

```
for ( int j = x; j <= 4 * x * y; j += y / x)
```

معادل عبارت زیر خواهد بود

```
for ( int j = 2; j <= 80; j += 5)
```

مقدار بخش افزایش می‌تواند منفی باشد، در اینحالت گام پیمایش حلقه معکوس خواهد بود. اگر شرط تکرار حلقه در همان ابتدای کار برقرار نباشد، بدنه عبارت **for** اجرا نخواهد شد، و اجرا با عبارت پس از

عبارت **for** ادامه می‌یابد. مقدار متغیر کنترلی به دفعات در محاسبات درون بدنه عبارت **for** بکار گرفته می‌شود یا به نمایش در می‌آید، اما اینکار الزامی ندارد. در بسیاری از موارد از متغیر کنترلی فقط برای کنترل تکرار استفاده می‌شود.

اجتناب از خطا

*اگر چه مقدار، متغیر کنترل‌ی می‌تواند در بدنه حلقه **for** تغییر یابد، اما از انجام چنین کاری اجتناب کنید چرا که می‌تواند برنامه را بسمت خطاهای ناخواسته‌ای هدایت کند.*

دیاگرام فعالیت UML عبارت for

دیاگرام فعالیت UML عبارت **for** شبیه به دیاگرام فعالیت **while** است (شکل ۶-۴). در شکل ۴-۵ دیاگرام فعالیت عبارت **for** بکار رفته در برنامه ۲-۵ آورده شده است. در این دیاگرام مشخص است که فرآیند مقداردهی اولیه فقط یکبار و قبل از ارزیابی تست شرط تکرار حلقه برای بار اول صورت می‌گیرد و اینکه عمل افزایش در هر بار و پس از اجرای عبارات بدنه انجام می‌شود. دقت کنید (در کنار وضعیت اولیه، خطوط انتقال، ادغام، وضعیت پایانی) دیاگرام فقط حاوی یک وضعیت عمل و یک تصمیم‌گیری است.

شکل ٤-٥ | دیاگرام فعالیت UML عبارت تکرار for در برنامه ۲-۵.

٤-٥ مثال‌های با استفاده از عبارت for

مثال‌های مطرح شده در این بخش، متدهای متنوعی از کاربرد متغیر کنترلی در یک عبارت **for** هستند. در هر مورد، سرآیند **for** نوشته شده است.

a) متغیر کنترلی که از 1 تا 100 با گام 1 افزایش می‌یابد.

```
for ( int i = 1; i <= 100; i++)
```

b) متغیر کنترلی که از 100 تا 1 با گام 1- افزایش می‌یابد (با کاهش 1).

```
for ( int i = 100; i >= 1; i--)
```

c) متغیر کنترلی که از 7 تا 77، با گام 7 افزایش می‌یابد.

```
for ( int i = 7; i <= 77; i += 7)
```

d) متغیر کنترلی که از 20 تا 2 با گام 2- افزایش می‌یابد.

```
for ( int i = 20; i >= 2; i -=2)
```

e) متغیر کنترلی، که توالی از مقادیر 20، 17، 14، 11، 8، 5 و 2 به خود می‌گیرد.

```
for ( var j = 2; j <= 20; j += 3)
```

f) متغیر کنترلی که توالی از مقادیر 0، 11، 22، 33، 44، 55، 66، 77، 88 و 99 به خود می‌گیرد.

```
for ( var j = 99; j >= 20; j -=11)
```

خطای برنامه‌نویسی

نتیجه عدم استفاده صحیح از عملگر رابطه‌ای در شرط تکرار حلقه که بصورت معکوس شمارش می‌کند (همانند استفاده اشتباه ***i <= 1*** *بجای* ***i >= 1*** *در شمارش معکوس حلقه به سمت 1) معمولاً یک خطای منطقی است که نتایج اشتباهی به هنگام اجرا برنامه تولید می‌کند.*

برنامه: مجموع اعداد زوج از 2 تا 20

دو مثال بعدی برنامه‌های ساده‌ای هستند که به توضیح عبارت **for** می‌پردازند. برنامه ۵-۵ از عبارت **for** برای بدست آوردن مجموع اعداد زوج 100 تا 2 استفاده می‌کند. در هر بار تکرار حلقه (خطوط 13-12) مقدار جاری متغیر کنترل **number** به متغیر **total** افزوده می‌شود.

```
1  // Fig. 5.5: fig05_05.cpp
2  // Summing integers with the for statement.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  int main()
8  {
9     int total = 0; // initialize total
10
11    // total even integers from 2 through 20
12    for ( int number = 2; number <= 20; number += 2 )
13       total += number;
14
15    cout << "Sum is " << total << endl; // display results
16    return 0; // successful termination
17 } // end main
```

```
Sum is 110
```

شکل ۵-۵ | عبارت for بکار رفته برای محاسبه مجموع اعداد زوج.

به بدنه عبارت **for** در برنامه ۵-۵ توجه نمائید. در واقع می‌توان این بدنه را با سمت راستترین بخش سرآیند **for** و با استفاده از *کاما* ادغام کرد، بصورت زیر:

```
for ( int number = 2; // initialization
      number <= 20; // loop continuation condition
      total += number, number += 2 ) // total and increment
;// empty body
```

برنامه‌نویسی ایده‌ال

گرچه ادغام عبارات بدنه **for** *با بخش سرآیند آن وجود دارد اما از انجام اینکار اجتناب کنید تا درک برنامه مشکل نشود.*

برنامه‌نویسی ایده‌ال

در صورت امکان، سایز عبارات سرآیند کنترل را محدود کنید.

برنامه:محاسبه سود

مثال بعدی در ارتباط با محاسبه یک مسئله ترکیبی با استفاده از عبارت **for** است. صورت مسئله به شرح زیر است:

شخصی 1000.00 دلار در یک حساب پس انداز با سود 5٪ سرمایه‌گذاری کرده است. با فرض اینکه کل سود نیز ذخیره می‌شود، مقدار پول موجود در حساب را در پایان هر سال در یک مدت 10 ساله محاسبه و چاپ کنید. برای بدست آوردن این مقادیر، از فرمول زیر کمک بگیرید:

$$a = p(1 + r)^n$$

در این فرمول

p میزان سرمایه اولیه

r نرخ سود

n تعداد سال‌ها

a مقدار سرمایه در پایان سال nام است.

این مسئله مستلزم حلقه‌ای است که دلالت بر انجام محاسبه‌ای برای هر سال در مدت ده سال پول باقیمانده در حساب پس‌انداز است. این راه حل در برنامه شکل ۶-۵ آورده شده است.

عبارت **for** (خطوط 35-28) مبادرت به اجرای بدنه خود به میزان 10 بار می‌کند. مقدار متغیر کنترلی از 1 تا 10 به میزان یک واحد در هر بار افزایش می‌یابد. زبان C++ حاوی عملگر توان نیست، از اینرو از تابع کتابخانه‌ای استاندارد **pow** به همین منظور استفاده کرده‌ایم (خط 31). تابع **pow(x,y)** مبادرت به محاسبه مقدار **x** به توان **y** می‌کند. در این مثال، عبارت جبری $(1+r)^n$ بصورت pow(1.0+rate,year) نوشته شده است، که متغیر **rate** نشاندهنده **r** و متغیر **year** نشاندهنده **n** است. تابع **pow** دو آرگومان از نوع **double** دریافت و یک مقدار **double** برگشت می‌دهد.

این برنامه بدون سرآیند فایل <**cmath**> کامپایل نمی‌شود. تابع **pow** نیازمند دو آرگومان از نوع **double** است. دقت کنید که متغیر **year** از نوع صحیح است. سرآیند <**cmath**> حاوی اطلاعاتی است که به کامپایلر می‌گوید تا مقدار **year** را بطور موقت تبدیل به نوع **double** کند، قبل از اینکه تابع فراخوانی شود. این اطلاعات در نمونه اولیه تابع **pow** وجود دارند. در فصل ششم با چندین تابع کتابخانه **math**آشنا خواهید شد.

```
1  // Fig. 5.6: fig05_06.cpp
2  // Compound interest calculations with for.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::fixed;
7
8  #include <iomanip>
9  using std::setw; // enables program to set a field width
10 using std::setprecision;
11
12 #include <cmath> // standard C++ math library
13 using std::pow; // enables program to use function pow
```

```
14
15 int main()
16 {
17    double amount; // amount on deposit at end of each year
18    double principal = 1000.0; // initial amount before interest
19    double rate = .05; // interest rate
20
21    // display headers
22    cout << "Year" << setw( 21 ) << "Amount on deposit" << endl;
23
24    // set floating-point number format
25    cout << fixed << setprecision( 2 );
26
27    // calculate amount on deposit for each of ten years
28    for ( int year = 1; year <= 10; year++ )
29    {
30       // calculate new amount for specified year
31       amount = principal * pow( 1.0 + rate, year );
32
33       // display the year and the amount
34       cout << setw( 4 ) << year << setw( 21 ) << amount << endl;
35    } // end for
36
37    return 0; // indicate successful termination
38 } // end main
```

```
Year     Amount on deposit
1                  1050.00
2                  1102.50
3                  1157.63
4                  1215.51
5                  1276.28
6                  1340.10
7                  1407.10
8                  1477.46
9                  1551.33
10                 1628.89
```

شکل ٦-٥ | عبارت for برای محاسبه سود سرمایه‌گذاری.

خطای برنامه‌نویسی

*بطور کلی، فراموش کردن الحاق فرآیند سرآیند مورد نیاز به هنگام استفاده از توابع کتابخانه استاندارد (همانند <**cmatch**>) خطای کامپایل بدنبال خواهد داشت.*

دقت به هنگام استفاده از نوع double در محاسبات مالی

دقت کنید که متغیرهای **amount** و **principal** و **rate** از نوع ***double*** هستند. به این دلیل از این نوع استفاده کرده‌ایم که می‌خواهیم به بخش‌های کسری رسیدگی کرده و به نوعی نیاز داریم که امکان انجام محاسبات دقیق در زمینه مسائل مالی را فراهم آوریم. متاسفانه این نوع می‌تواند مشکل‌ساز شود. در این بخش با یک توضیح ساده شاهد خواهید بود که چگونه به هنگام استفاده از نوع **float** یا **double** در نمایش‌های مالی می‌توانیم دچار اشتباه شویم (فرض کنید از **setprecision(2)** برای مشخص کردن دو رقم دقت به هنگام چاپ استفاده کرده‌ایم): دو مبلغ دلاری ذخیره شده در ماشین می‌تواند 14.234 (که 14.23 چاپ می‌شود) و 18.673 (که 18.67 چاپ خواهد شد) را تولید کند. زمانیکه این مبالغ با هم جمع

شوند، مجموع داخلی 32.907 تولید می‌شود که بصورت 32.91 چاپ می‌گردد. از اینرو نتیجه چاپی به صورت زیر ظاهر خواهد شد.

```
  14.23
+ 18.67
-------
  32.91
```

اما در صورتیکه خودمان این اعداد را با هم جمع کنیم مجموع 32.90 را بدست خواهیم آورد. پس مشکلی در کار است.

برنامه‌نویسی ایده‌ال

*از متغیرهای نوع **float** یا **double** برای انجام محاسبات مالی استفاده نکنید. عدم دقت کافی در چنین اعدادی می‌تواند در نتیجه تولیدی از محاسبات مالی، شما را دچار مشکل کند.*

کنترل کننده‌های جریان برای قالب‌بندی کردن خروجی عددی

عبارت خروجی در خط 25 قبل از حلقه **for** و عبارت خروجی در خط 34 در حلقه **for** ترکیب شده‌اند تا مقادیر متغیرهای **year** و **amount** با قالب‌بندی (فرمت) تصریح شده توسط کنترل‌کننده‌های جریان پارامتری شده **setprecision** و **setw** و کنترل‌کننده استریم غیرپارامتری **fixed** چاپ شوند. کنترل‌کننده استریم **setw(4)** مشخص می‌کند که باید مقدار خروجی بعدی به طول فیلد مشخص شده یعنی 4 به نمایش درآید، یعنی، **cout** مقدار را با حداقل 4 موقعیت کاراکتری چاپ خواهد کرد. اگر مقداری که چاپ می‌شود، کمتر از 4 کاراکتر طول داشته باشد، مقدار از سمت راست تراز می‌شود (بطور پیش‌فرض). اگر مقدار خروجی بیش از 4 کاراکتر طول داشته باشد، طول فیلد گسترش می‌یابد تا به کل مقدار جا دهد. برای تاکید بر این نکته که مقادیر در خروجی باید از سمت چپ تراز شوند، کافیست از کنترل‌کننده استریم غیرپارامتری **left** استفاده شود (در سرآیند <**iostream**> قرار دارد). ترازبندی از سمت راست را می‌توان با استفاده از کنترل‌کننده استریم غیرپارامتری **right** بدست آورد.

قالب‌بندی‌های دیگر، در عبارات خروجی بر این نکته دلالت می‌کنند که متغیر **amount** بصورت یک مقدار با نقطه ثابت با یک نقطه دیسمال (تصریح شده در خط 25 با کنترل‌کننده استریم **fixed**) تراز از سمت راست در فیلدی به میزان 21 کاراکتر (تصریح شده در خط 34 با **setw(21)**) و با دقت دو رقم در سمت راست نقطه دیسمال (تصریح شده در خط 25 با کنترل‌کننده **setprecision(2)**) چاپ خواهد شد. کنترل‌کننده‌های استریم **fixed** و **setprecision** را بر روی استریم خروجی (یعنی **cout**) و قبل از حلقه **for** اعمال کرده‌ایم چرا که این تنظیمات قالب‌بندی تا زمانیکه تغییر نیافته‌اند ثابت باقی می‌مانند، به چنین تنظیماتی، تنظیمات *چسبنده* می‌گویند. از اینرو، نیازی ندارند در هر بار تکرار حلقه بکار گرفته شوند. با

این همه، طول فیلد (میدان) مشخص شده با **setw** فقط بر مقدار بعدی در خروجی بکار گرفته می‌شود. در فصل پانزدهم در ارتباط با قابلیت‌های قالبندی ورودی/خروجی زبان C++ صحبت خواهیم کرد.

به عبارت **rate + 1.0** که بصورت یک آرگومان در تابع **pow** و در بدنه عبارت **for** قرار دارد، توجه کنید. در واقع، این محاسبه همان نتیجه را در زمان هر تکرار حلقه تولید می‌کند، از اینرو تکرار بی‌موردی صورت گرفته و باید این عبارت یکبار و قبل از حلقه محاسبه شود.

کارایی

برخی از کامپایلرها حاوی ویژگی‌های بهینه‌سازی هستند که سبب افزایش کارایی کد نوشته شده توسط شما می‌شوند، اما بهتر است از همان مرحله شروع کار کدنویسی، کدهای خود را کارآمد بنویسیم.

کارایی

از قراردادن عبارات محاسباتی در داخل حلقه که مقدار آنها در هر بار اجرای حلقه تغییری نمی‌یابد خودداری کنید. چنین عباراتی فقط یکبار و قبل از حلقه باید ارزیابی شوند.

۵-۵ عبارت تکرار do...while

عملکرد عبارت تکرار **do...while** همانند عبارت **while** است. در عبارت **while**، شرط تکرار حلقه در ابتدای حلقه تست می‌شود، قبل از اینکه بدنه حلقه به اجرا درآید. عبارت **do...while** شرط تکرار حلقه را پس از اجرای حلقه تست می‌کند. از اینرو، در یک عبارت **do...while**، همیشه بدنه حلقه حداقل یکبار به اجرا در می‌آید. اگر فقط یک عبارت در بدنه وجود داشته باشد، استفاده از براکت‌ها در **do...while** ضرورتی ندارد. با این همه، معمولا برای اجتناب از سردرگمی مابین عبارات **while** و **do...while** از براکت‌ها استفاده می‌شود. برای مثال،

```
while (شرط)
```

نشان‌دهنده سرآیند عبارت **while** است. یک عبارت **do...while** بدون براکت‌ها در اطراف بدنه خود (با یک عبارت) بصورت زیر خواهد بود

```
do
   عبارت
while ( شرط );
```

که می‌تواند باعث اشتباه شود. خط آخر، می‌تواند توسط خواننده به غلط بعنوان یک عبارت while با عبارت تهی تفسیر شود (وجود سیمکولن در انتهای شرط). از اینرو، برای اجتناب از اشتباه، بهتر است در یک عبارت **do...while** با یک عبارت از براکت‌ها استفاده شود:

```
do {
```

عبارت;

} **while** (*شرط*);

در برنامه شکل ۷-۵ از یک عبارت **do...while** برای چاپ مقادیر از 10 تا 1 استفاده شده است. عبارت **do...while** در خطوط 15-11 اعمال شده است. در اولین برخورد برنامه با این عبارت، خط 13 اجرا شده و مقدار متغیر **counter** (با مقدار 1) چاپ شده، سپس **counter** یک واحد افزایش می‌یابد (خط 14). سپس شرط در خط 15 ارزیابی می‌شود. اکنون متغیر **counter** حاوی مقدار **2** است که کمتر یا مساوی با **10** می‌باشد، چون شرط برقرار است، عبارت **do...while** مجدداً اجرا می‌شود. در بار دهم که عبارت اجرا شد، عبارت خط 13 مقدار **10** را چاپ کرده و در خط 14، مقدار **counter** به **11** افزایش می‌یابد. در این لحظه، شرط موجود در خط 15، نادرست ارزیابی شده و برنامه از عبارت **do...while** خارج می‌شود و برنامه به سراغ عبارت پس از حلقه می‌رود (خط 17).

```
1  // Fig. 5.7: fig05_07.cpp
2  // do...while repetition statement.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  int main()
8  {
9     int counter = 1; // initialize counter
10
11    do
12    {
13       cout << counter << " "; // display counter
14       counter++; // increment counter
15    } while ( counter <= 10 ); // end do...while
16
17    cout << endl; // output a newline
18    return 0; // indicate successful termination
19 } // end main
```

```
1 2 3 4 5 6 7 8 9 10
```

شکل ۷-۵ | عبارت تکرار do...while

دیاگرام فعالیت UML عبارت do...while

شکل ۸-۵ حاوی دیاگرام فعالیت UML برای عبارت **do...while** است. این دیاگرام به وضوح نشان می‌دهد که شرط تکرار حلقه حداقل پس از یک بار اجرای عبارات موجود در بدنه حلقه، ارزیابی نخواهد شد. این دیاگرام فعالیت را با عبارت **while** بکار رفته در شکل ۶-۴ مقایسه کنید. مجدداً، توجه کنید که این دیاگرام حاوی یک نماد وضعیت عمل و تصمیم‌گیری است.

شکل ۸-۵ | دیاگرام فعالیت UML عبارت تکرار do...while

۵-۶ عبارت چند انتخابی switch

در فصل گذشته، در ارتباط با عبارت تک انتخابی **if** و دو انتخابی **if...else** صحبت کردیم. زبان C++ دارای عبارت چند انتخابی **switch** برای رسیدگی به چنین شرایطی است.

کلاس GradeBook با عبارت switch

در مثال بعدی، مبادرت به عرضه یک نسخه بهبود یافته از کلاس **GradeBook** معرفی شده در فصل سوم و فصل چهارم می‌کنیم. نسخه جدید از کاربر می‌خواهد تا مجموعه‌ای از امتیازات حرفی را وارد کرده، سپس تعداد دانشجویانی که هر کدام امتیازی دریافت کرده‌اند، به نمایش در می‌آورد. این کلاس از یک عبارت **switch** برای تعیین اینکه امتیاز وارد شده یک A، B، C، D یا F می‌باشد و افزایش مقدار شمارنده امتیاز وارد شده، استفاده می‌کند. کلاس **GradeBook** در برنامه شکل ۹-۵ تعریف شده و تعریف تابع عضو آن در برنامه شکل ۱۰-۵ قرار دارد. شکل ۱۱-۵ نمایشی از اجرای نمونه برنامه همراه ورودی‌ها و خروجی برنامه **main** است که از کلاس **GradeBook** برای پردازش امتیازات استفاده می‌کند. همانند نسخه‌های قبلی تعریف کلاس، تعریف کلاس **GradeBook** در شکل ۹-۵ حاوی نمونه اولیه تابع برای توابع عضو **setCourseName** در خط 13، **getCourseName** در خط 14 و **displayMessage** در خط 15 به همراه سازنده کلاس در خط 12 است. همچنین در تعریف کلاس عضو داده **courseName** بصورت ***private*** اعلان شده است (خط 19).

کلاس **GradeBook** در شکل۹-۵ دارای پنج عضو داده **private** است (خطوط 14-20)، متغیرهای شمارنده برای هر امتیاز (یعنی برای A، B، C، D و F). همچنین کلاس دارای دو تابع عضو ***public*** بنام‌های **inputGrades** و **displayGradeReport** است. تابع عضو **inputGrade** (اعلان شده در خط 16) مبادرت خواندن حروف امتیازی به تعداد اختیاری از طرف کاربر با روش مقدار مراقبتی می‌کند و شمارنده امتیاز مقتضی را برای هر امتیاز وارد شده به روز می‌نماید. تابع عضو **displayGradeReport** (اعلان شده در خط 17) گزارشی از تعداد دانشجویان دریافت‌کننده هر امتیاز به نمایش در می‌آورد.

```
1  // Fig. 5.9: GradeBook.h
2  // Definition of class GradeBook that counts A, B, C, D and F grades.
3  // Member functions are defined in GradeBook.cpp
4
5  #include <string> // program uses C++ standard string class
6  using std::string;
7
8  // GradeBook class definition
9  class GradeBook
10 {
11 public:
12    GradeBook( string ); // constructor initializes course name
13    void setCourseName( string ); // function to set the course name
14    string getCourseName(); // function to retrieve the course name
15    void displayMessage(); // display a welcome message
16    void inputGrades(); // input arbitrary number of grades from user
17    void displayGradeReport(); // display a report based on the grades
18 private:
19    string courseName; // course name for this GradeBook
20    int aCount; // count of A grades
21    int bCount; // count of B grades
22    int cCount; // count of C grades
23    int dCount; // count of D grades
24    int fCount; // count of F grades
```

```
25 }; // end class GradeBook
```

شکل ۹-۵ | تعریف کلاس GradeBook.

فایل کد منبع **GradeBook.cpp** در شکل ۱۰-۵ حاوی تعریف تابع عضو کلاس **GradeBook** است. توجه کنید زمانیکه یک شی **GradeBook** برای اولین بار ایجاد می‌شود و هنوز هیچ امتیازی وارد نشده است، خطوط 20-16 در سازنده مبادرت به مقداردهی اولیه پنج شمارنده امتیاز با صفر می‌کنند. همانطوری که بزودی خواهید دید، این شمارنده‌ها در تابع عضو **inputGrade** و بعنوان امتیازهای ورودی کاربر افزایش می‌یابند. تعاریف توابع عضو **setCourseName**، **getCourseName** و **displayMessage** با نسخه‌های قبلی موجود در کلاس **GradeBook** یکسان هستند. اجازه دهید تا نگاهی به توابع عضو جدید در **GradeBook** داشته باشیم.

```
1  // Fig. 5.10: GradeBook.cpp
2  // Member-function definitions for class GradeBook that
3  // uses a switch statement to count A, B, C, D and F grades.
4  #include <iostream>
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  #include "GradeBook.h" // include definition of class GradeBook
10
11 // constructor initializes courseName with string supplied as argument;
12 // initializes counter data members to 0
13 GradeBook::GradeBook( string name )
14 {
15    setCourseName( name ); // validate and store courseName
16    aCount = 0; // initialize count of A grades to 0
17    bCount = 0; // initialize count of B grades to 0
18    cCount = 0; // initialize count of C grades to 0
19    dCount = 0; // initialize count of D grades to 0
20    fCount = 0; // initialize count of F grades to 0
21 } // end GradeBook constructor
22
23 // function to set the course name; limits name to 25 or fewer characters
24 void GradeBook::setCourseName( string name )
25 {
26    if ( name.length() <= 25 ) // if name has 25 or fewer characters
27       courseName = name; // store the course name in the object
28    else // if name is longer than 25 characters
29   { // set courseName to first 25 characters of parameter name
30       courseName = name.substr( 0, 25 ); // select first 25 characters
31       cout << "Name \"" << name << "\" exceeds maximum length (25).\n"
32          << "Limiting courseName to first 25 characters.\n" << endl;
33    } // end if...else
34 } // end function setCourseName
35
36 // function to retrieve the course name
37 string GradeBook::getCourseName()
38 {
39    return courseName;
40 } // end function getCourseName
41
42 // display a welcome message to the GradeBook user
43 void GradeBook::displayMessage()
44 {
45    // this statement calls getCourseName to get the
46    // name of the course this GradeBook represents
47    cout << "Welcome to the grade book for\n" << getCourseName() << "!\n"
48       << endl;
```

```
49 } // end function displayMessage
50
51 // input arbitrary number of grades from user; update grade counter
52 void GradeBook::inputGrades()
53 {
54    int grade; // grade entered by user
55
56    cout << "Enter the letter grades." << endl
57       << "Enter the EOF character to end input." << endl;
58
59    // loop until user types end-of-file key sequence
60    while ( ( grade = cin.get() ) != EOF )
61    {
62      // determine which grade was entered
63       switch ( grade ) // switch statement nested in while
64       {
65          case 'A': // grade was uppercase A
66          case 'a': // or lowercase a
67             aCount++; // increment aCount
68             break; // necessary to exit switch
69
70          case 'B': // grade was uppercase B
71          case 'b': // or lowercase b
72             bCount++; // increment bCount
73             break; // exit switch
74
75          case 'C': // grade was uppercase C
76         case 'c': // or lowercase c
77             cCount++; // increment cCount
78             break; // exit switch
79
80          case 'D': // grade was uppercase D
81          case 'd': // or lowercase d
82             dCount++; // increment dCount
83             break; // exit switch
84
85          case 'F': // grade was uppercase F
86          case 'f': // or lowercase f
87             fCount++; // increment fCount
88             break; // exit switch
89
90          case '\n': // ignore newlines,
91          case '\t': // tabs,
92          case ' ': // and spaces in input
93             break; // exit switch
94
95          default: // catch all other characters
96             cout << "Incorrect letter grade entered."
97                << " Enter a new grade." << endl;
98             break; // optional; will exit switch anyway
99       } // end switch
100    } // end while
101 } // end function inputGrades
102
103 // display a report based on the grades entered by user
104 void GradeBook::displayGradeReport()
105 {
106    // output summary of results
107    cout << "\n\nNumber of students who received each letter grade:"
108       << "\nA: " << aCount // display number of A grades
109       << "\nB: " << bCount // display number of B grades
110       << "\nC: " << cCount // display number of C grades
111       << "\nD: " << dCount // display number of D grades
112       << "\nF: " << fCount // display number of F grades
113       << endl;
114 } // end function displayGradeReport
```

شکل ۱۰-۵ | کلاس GradeBook از عبارت switch برای شمارش امتیازات A, B, C, D و F استفاده می‌کند.

خواندن کاراکترهای ورودی

کاربر مبادرت به وارد کردن امتیازات حرفی برای یک واحد درسی در تابع **inputGrades** می‌کند (خطوط 101-52). در سرآیند حلقه **while** در خط 60، ابتدا عبارت تخصیصی قرار گرفته در درون پرانتز (**grade = cin.get()**) اجرا می‌شود. تابع **cin.get()** یک کاراکتر از صفحه کلید خوانده و آن را در متغیر **grade** از نوع صحیح ذخیره می‌سازد (اعلان شده در خط 54). معمولاً کاراکترها در متغیرهای از نوع **char** ذخیره می‌شوند، با این همه، می‌توان کاراکترها را در هر نوع داده صحیحی ذخیره کرد، چرا که نشاندهنده 1 بایت صحیح در کامپیوتر هستند. از اینرو، می‌توانیم براساس استفاده، با یک کاراکتر همانند یک مقدار صحیح یا بعنوان یک کاراکتر رفتار کنیم. برای مثال، عبارت

```
cout << "The character (" << 'a' << ") has the value"
     << static_cast< int > ( 'a' ) << endl;
```

مبادرت به چاپ کاراکتر **a** و مقدار صحیح آن بصورت زیر می‌کند:

```
The character (a) has the value 97
```

عدد صحیح **97** نشاندهنده شماره عددی کاراکتر **a** در کامپیوتر است. اکثر کامپیوترها از مجموعه کاراکتری (*American Standard Code for Information Interchange*) ASCII استفاده می‌کنند، که در این مورد 97 نشاندهنده حرف کوچک 'a' است.

بطور کلی عبارات تخصیص‌دهنده مبادرت به تخصیص مقدار قرار گرفته در سمت راست به متغیر قرار گرفته در سمت چپ علامت = می‌کنند. بنابر این مقدار عبارت تخصیصی **grade=cin.get()** همان مقدار برگشتی از سوی **cin.get()** بوده و به متغیر **grade** تخصیص می‌یابد. می‌توان از عبارات تخصیص‌دهنده برای تخصیص یک مقدار به چندین متغیر استفاده کرد. برای مثال در عبارت زیر

```
a=b=c=0;
```

ابتدا تخصیص **c=0** صورت می‌گیرد چرا که عملگر = دارای شرکت‌پذیری از سمت راست به چپ است. سپس به متغیر **b** مقدار تخصیصی **c=0**، تخصیص می‌یابد (که صفر است). سپس متغیر **a** حاوی مقدار تخصیصی **b=(c=0)** خواهد شد که آن هم صفر است. در برنامه، مقدار عبارت تخصیصی **grade=cin.get()** با مقدار **EOF** مقایسه می‌شود (نمادی که کوتاه شده عبارت "*end-of-file*" است). ما از EOF (که معمولاً دارای مقدار **1-** است) بعنوان مقدار مراقبتی استفاده کرده‌ایم. با این همه، نیازی به تایپ مقدار 1- یا تایپ حروف EOF بعنوان مقدار مراقبتی ندارید. بجای آن از یک ترکیب کلیدی استفاده کرده‌ایم، که نشاندهنده EOF در سیستم است. **EOF** یک ثابت سمبولیک سیستم است که در سرآیند فایل <**iostream**> تعریف شده است. اگر مقداری به **grade** تخصیص دهید که معادل با EOF

باشد، حلقه **while** در خطوط 100-60 بکار خود خاتمه می‌دهد. نمایش کاراکترهای وارد شده به این برنامه را بصورت مقادیر صحیح انتخاب کرده‌ایم، چرا که EOF دارای یک مقدار صحیح است.

در سیستم‌های UNIX/Linux و برخی از سیستم‌های دیگر، EOF با تایپ

```
<ctrl> d
```

در یک خط وارد می‌شود. این عبارت به این معنی است که کلید *ctrl* را فشار و پایین نگه داشته و سپس کلید d فشار داده شود. در سیستم‌های دیگر همانند Microsoft Windows، می‌توان EOF را با تایپ

```
<ctrl> z
```

وارد کرد.

قابلیت حمل

کلیدهای ترکیبی برای وارد کردن EOF به سیستم وابسته هستند.

در این برنامه، کاربر امتیازها را توسط صفحه کلید وارد می‌کند. زمانیکه کاربر کلید Enter را فشار دهد، کاراکترها توسط تابع **cin.get()** خوانده می‌شوند، یک کاراکتر در یک زمان. اگر کاراکتر وارد شده EOF نباشد، برنامه وارد عبارت **switch** می‌شود (خط 99-63) که شمارنده امتیاز را متناسب با حرف وارد شده، یک واحد افزایش می‌دهد.

جزئیات عبارت switch

عبارت **switch** متشکل از تعدادی برچسب **case** و یک حالت **default** اختیاری است. از این عبارت در این مثال برای تعیین اینکه کدام شمارنده باید براساس امتیاز وارد شده افزایش یابد، استفاده شده است. زمانیکه کنترل برنامه به **switch** می‌رسد، برنامه مبادرت به ارزیابی عبارت موجود در درون پرانتزها می‌کند (یعنی **grade**) که بدنبال کلمه کلیدی **switch** قرار گرفته است (خط 63). به این عبارت، *عبارت کنترلی* گفته می‌شود. عبارت **switch** شروع به مقایسه مقدار عبارت کنترلی با هر برچسب **case** می‌کند. فرض کنید که کاربر حرف C را بعنوان کد امتیاز وارد کرده باشد. برنامه شروع به مقایسه C با هر **case** موجود در بدنه **switch** می‌کند. اگر مطابقتی یافت شود (در خط 75، **case 'C'**)، برنامه عبارات موجود در آن **case** را به اجرا در می‌آورد. در ارتباط با حرف C، خط 77 مبادرت به افزایش **cCount** به می‌زان یک واحد می‌کند. عبارت **break** (خط 78) سبب می‌شود که کنترل برنامه به اولین عبارت پس از **switch** منتقل شود که در این برنامه کنترل به خط 100 انتقال می‌یابد. این خط، انتهای بدنه حلقه **while** را نشان می‌دهد (خطوط 100-60)، از اینرو جریان کنترل به سمت شرط **while** در خط 60 می‌رود تا تعیین نماید که آیا حلقه بایستی ادامه یابد یا خیر.

caseهای موجود در عبارت **switch** بطور صریح مبادرت به تست حروف کوچک و بزرگ حرف‌های A، B، C، D و F می‌کنند. توجه کنید که caseهای موجود در خطوط 66-65 در تست مقادیر 'A' و 'a' کاربرد دارند (هر دو نشاندهنده امتیاز A می‌باشند). لیست caseها در این روش بصورت متوالی بوده و عبارتی مابین آنها وجود ندارد و به هر دو **case** اجازه می‌دهد تا یک مجموعه از عبارات را به اجرا درآورند، زمانیکه عبارت کنترلی با 'A' یا 'a' ارزیابی شود، عبارات قرار گرفته در خطوط 68-67 اجرا خواهند شد. توجه کنید که هر **case** می‌تواند چندین عبارت داشته باشد. عبارت انتخابی **switch** متفاوت از دیگر عبارات کنترلی بوده و نیازی به استفاده از براکت‌ها در اطراف چندین عبارت در هر **case** ندارد.

بدون عبارات **break**، هر زمان که مطابقتی در **switch** تشخیص داده شود، عبارات آن **case** و caseهای متعاقب آن اجرا خواهند شد تا اینکه به یک عبارت **break** یا به انتهای **switch** برسد. به این حالت "*falling through*" یا عدم دستیابی به نتیجه در caseهای بعدی گفته می‌شود.

خطای برنامه‌نویسی

*نتیجه فراموش کردن عبارت **break** در مکانی که به آن در **switch** نیاز است، یک خطای منطقی است.*

خطای برنامه‌نویسی

*حذف فاصله مابین کلمه **case** و مقدار ارزش در یک عبارت **switch** خطای منطقی است. برای مثال، نوشتن **case3:** بجای **case 3:** یک برچسب بلااستفاده بوجود می‌آورد.*

تدارک دیدن حالت *default*

اگر هیچ مطابقتی مابین مقدار عبارت کنترلی و یک برچسب **case** پیدا نشود، حالت **default** اجرا خواهد شد (خطوط 98-95). در این مثال از حالت **default** برای پردازش تمام مقادیر عبارت کنترلی که امتیازهای معتبر نیستند، کاراکترهای خط جدید، تب یا فاصله استفاده کرده‌ایم. اگر هیچ مطابقتی رخ ندهد، حالت **default** اجرا می‌شود و خطوط 97-96 یک پیغام خطا به نمایش در می‌آورند تا بر این نکته دلالت کنند که یک حرف امتیازی اشتباه وارد شده است. اگر هیچ مطابقتی در یک عبارت **switch** رخ ندهد و این عبارت فاقد حالت **default** باشد، کنترل برنامه بسادگی به اجرای اولین عبارت پس از **switch** ادامه خواهد داد.

برنامه‌نویسی ایده‌ال

*در عبارات **switch** حالت **default** را در نظر بگیرید. در حالت **default** برنامه‌نویس می‌تواند مواردی که برای پردازش موارد استثناء پیش می‌آیند در نظر بگیرد. با اینکه می‌توان به هر ترتیبی شرط‌های **case** و حالت **default** را در یک عبارت **switch** قرار داد، اما بهتر است ضابطه **default** در آخر قرار داده شود.*

برنامه‌نویسی ایده‌ال

*در یک عبارت **switch** که **default** در انتهای آن قرار دارد، نیازی نیست که ضابطه **default** حاوی دستور*

__break__ باشد. برخی از برنامه‌نویسان به منظور حفظ تقارن و وضوح بیشتر با سایر caseها ترجیح می‌دهند از __break__ در __default__ استفاده کنند.

نادیده گرفتن کارکترهای خط جدید، تب و فاصله در ورودی

خطوط 93-90 در عبارت **switch** شکل ۱۰–۵ سبب می‌شوند تا برنامه کاراکترهای خط جدید، تب و فاصله‌ها را در نظر نگیرد. خواندن یک کاراکتر در هر زمان می‌تواند مشکل‌ساز شود. برای داشتن برنامه‌ای که کاراکترها را بخواند، بایستی آنها را به کامپیوتر با فشردن کلید Enter صفحه‌کلید ارسال کنیم. با اینکار یک کاراکتر خط جدید (*newline*) در ورودی پس از کاراکترهایی که مایل به پردازش آنها هستیم، وارد می‌شود. غالباً برای اینکه برنامه بدرستی کار کند نیاز است تا به این کاراکتر رسیدگی شود. با قرار دادن **case** سابق‌الذکر در عبارت **switch**، مانع از نمایش پیغام خطا در حالت **default** در هر بار مواجه شدن با کاراکترهای خط جدید، تب (tab) یا فاصله در ورودی شده‌ایم.

خطای برنامه‌نویسی

عدم پردازش کاراکترهای خط جدید و سایر کاراکترهای white-space در ورودی، زمانیکه در هر بار یک کاراکتر خوانده می‌شود، می‌تواند شما را با خطاهای منطقی مواجه سازد.

تست کلاس GradeBook

برنامه شکل ۱۱–۵ یک شی **GradeBook** ایجاد می‌کند (خط 9). خط 11 تابع عضو **displayMessage** شی را برای نمایش پیغام خوش‌آمدگویی به کاربر فراخوانی می‌کند. خط 12 تابع عضو **inputGrades** را برای خواندن مجموعه‌ای از امتیازها از سوی کاربر و شمارش تعداد دانشجویان دریافت‌کننده امتیاز فراخوانی می‌کند. به پنجره خروجی/ورودی به نمایش درآمده در شکل ۱۱–۵ توجه کنید که یک پیغام خطا در واکنش به وارد کردن یک امتیاز اشتباه (یعنی E) به نمایش درآورده است. خط 13 مبادرت به فراخوانی تابع عضو **displayGradeReport** تعریف شده در خطوط 114-104 از شکل ۱۰–۵ می‌کند و این تابع نتیجه برنامه را براساس امتیازهای وارد شده به نمایش در می‌آورد.

```
1  // Fig. 5.11: fig05_11.cpp
2  // Create GradeBook object, input grades and display grade report.
3
4  #include "GradeBook.h" // include definition of class GradeBook
5
6  int main()
7  {
8     // create GradeBook object
9     GradeBook myGradeBook( "CS101 C++ Programming" );
10
11    myGradeBook.displayMessage(); // display welcome message
12    myGradeBook.inputGrades(); // read grades from user
13    myGradeBook.displayGradeReport(); // display report based on grades
14    return 0; // indicate successful termination
15 } // end main
```

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

Enter a letter grades.
Enter the EOF character to end input.
a
B
c
C
A
d
f
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
b
^z

Number of student who received each letter grade:
A:3
B:2
C:3
D:2
F:1
```

شکل ۱۱-۵ | ایجاد یک شی GradeBook و فراخوانی توابع عضو آن.

دیاگرام فعالیت UML عبارت switch

شکل ۱۲-۵ نشاندهنده دیاگرام فعالیت UML یک عبارت چند انتخابی **switch** است. اکثر عبارات **switch** از یک **break** در هر **case** استفاده می‌کنند تا به عبارت **switch** پس از پردازش آن **case** خاتمه دهند. شکل۱۲-۵ بر استفاده از عبارت **break** در دیاگرام فعالیت تاکید دارد. بدون عبارت **break**کنترل به اولین عبارت پس از ساختار **switch** پس از اینکه یک **case** پردازش شد، منتقل نمی‌شود. بجای آن کنترل به سراغ اجرای **case** بعدی می‌رود.

این دیاگرام به وضوح نشان می‌دهد که عبارت **break** در انتهای یک **case** سبب انتقال بلافاصله کنترل به خارج از عبارت **switch** می‌شود. مجدداً توجه کنید که این دیاگرام حاوی نمادهای وضعیت عمل و تصمیم‌گیری است. همچنین توجه کنید که در این دیاگرام از نمادهای ادغام برای انتقال از عبارات **break** به وضعیت پایانی استفاده شده است.

به هنگام استفاده از عبارت **switch**، بخاطر داشته باشید که می‌توان از آن فقط برای تست یک مقدار ارزشی ثابت استفاده کرد. هر ترکیبی از ثابت‌های کاراکتری و ثابت‌های عددی صحیح، بصورت یک ثابت عددی صحیح ارزیابی می‌شوند. یک ثابت کاراکتری بصورت یک کاراکتر خاص در میان علامت نقل قول همانند **'A'** است. یک ثابت عددی، یک مقدار عددی صحیح است. همچنین هر برچسب **case** می‌تواند تعیین‌کننده یک عبارت ارزشی ثابت باشد.

خطای برنامه‌نویسی

مشخص کردن یک عبارت همراه با متغیرها همانند ***a+b*** در برچسب ***case*** یک ***switch***، خطای نحوی

است.

خطای برنامه‌نویسی

*تدارک دیدن برچسب‌های یکسان در یک عبارت **switch** خطای کامپایل بدنبال خواهد داشت. همچنین قرار دادن عبارات مختلف در caseها که مقدار آنها یکسان ارزیابی گردند نیز خطای کامپایل بوجود می‌آورد. برای مثال قرار دادن **case4+1:** و **case3+2:** در یک عبارت **switch** خطای کامپایل است چرا که هر دو آنها برابر **case 5:** هستند.*

شکل ۱۲-۵ | دیاگرام فعالیت UML ساختار switch با عبارت break.

تکاتی در ارتباط با نوع داده

زبان C++ دارای نوع داده با سایزهای مختلف است. برای مثال، امکان دارد برنامه‌های مختلف، به اعداد صحیح با سایزهای متفاوت نیاز داشته باشند. C++ دارای چندین نوع داده برای عرضه مقادیر صحیح است. طول مقادیر صحیح برای هر نوع بستگی به سخت‌افزار کامپیوتر دارد. علاوه بر نوع‌های **int** و **char**، زبان C++ نوع‌های **short** (کوتاه شده short int) و **long** (کوتاه شده long int) را در نظر گرفته است. حداقل محدودهٔ مقادیر صحیح از نوع **short** از 32.768- تا 32.767 می‌باشد. برای اکثر محاسبات صحیح، مقادیر صحیح از نوع **long** کافی هستند. حداقل محدودهٔ مقادیر از نوع **long** از 2.147.483.648- تا 2.147.483.647 است. بر روی اکثر کامپیوترها، مقادیر **int** معادل **short** یا **long** هستند. محدودهٔ مقادیر برای یک **int** حداقل برابر با مقادیر **short** بوده و بیشتر از مقادیر **long** نمی‌باشند. از نوع داده **char** می‌توان برای عرضه هر کاراکتری در مجموعه کاراکتری کامپیوتر استفاده کرد. همچنین می‌توان از آن برای نمایش مقادیر صحیح کوچک استفاده کرد.

قابلیت حمل

*بدلیل اینکه intها می‌توانند مابین سیستم‌ها سایز متفاوتی داشته باشند، اگر انتظار دارید محاسبه‌ای سبب تولید مقداری بیش از محدودهٔ 32.767 تا 32.763- نماید از نوع **long** استفاده کرده و در صورت امکان برنامه را بر روی سیستم‌های مختلف اجرا کنید.*

کارایی

اگر استفاده بهینه از حافظه مطرح باشد، می‌توانید از مقادیر صحیح با سایز کوچک استفاده کنید.

۵-۷ عبارات break و continue

علاوه بر عبارات کنترلی و انتخابی، زبان C++ عبارات **break** و **continue** را برای ایجاد تغییر در جریان کنترل در نظر گرفته است. بخش بعدی شما را با نحوه استفاده از **break** در خاتمه دادن به اجرای یک عبارت **switch** آشنا خواهد کرد. همچنین این بخش در مورد نحوه استفاده از **break** در یک عبارت تکرار مطالبی عرضه خواهد کرد.

عبارت break

دستور **break** با اجرا در عبارات **while ,for ,do..while** یا **switch** سبب می‌شود تا کنترل بلافاصله از عبارت خارج شده، و برنامه با اولین عبارت پس از عبارت ادامه می‌یابد. معمولا از دستور **break** برای خارج شدن زود هنگام از حلقه یا پرش از مابقی عبارت **switch** (همانند برنامه ۱۰-۵) استفاده می‌شود. برنامه شکل ۱۳-۵ به توضیح عملکرد دستور **break** در عبارت تکرار **for** پرداخته است (خط 14).

```
1  // Fig. 5.13: fig05_13.cpp
2  // break statement exiting a for statement.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  int main()
8  {
9     int count; // control variable also used after loop terminates
10
11    for ( count = 1; count <= 10; count++ ) // loop 10 times
12    {
13       if ( count == 5 ) // if count is 5,
14          break;         // terminate loop
15
16       cout << count << " ";
17    } // end for
18
19    cout << "\nBroke out of loop at count = " << count << endl;
20    return 0; // indicate successful termination
21 } // end main
```

```
1 2 3 4
Broke out of loop at count = 5
```

۱۳-۵ | عبارت break در یک عبارت for.

زمانیکه عبارت **if** تشخیص می‌دهد که **count** برابر 5 است، دستور **break** اجرا می‌شود. با اینکار عبارت **for** خاتمه می‌یابد و برنامه به خط 19 منتقل می‌شود (بلافاصله پس از عبارت **for**)، و پیغامی مبنی بر اینکه متغیر کنترلی به هنگام خاتمه حلقه چه مقداری داشته به نمایش در می‌آید. عبارت **for** بطور کامل و فقط چهار بار بجای ده بار اجرا می‌شود. توجه کنید که متغیر کنترلی **count** در خارج از سرآیند عبارت **for** تعریف شده است، از اینروست که می‌توانیم از متغیر کنترلی هم در بدنه حلقه و هم پس از آن استفاده کنیم.

***عبارت* countinue**

دستور **continue**، با اجرا شدن در عبارات **for, while** یا **do..while** از مابقی عبارات موجود در درون بدنه عبارت پرش کرده و کار را با حلقه بعد دنبال می‌کند. در عبارات **while** و **do..while**، شرط تکرار حلقه بلافاصله پس از اجرای **continue** ارزیابی می‌شود. در عبارات **for**، بخش افزایش دهنده، پس از ارزیابی شرط تکرار حلقه صورت می‌گیرد. این مورد تنها اختلاف مابین **for** و **while** است. قرار دادن اشتباه **continue** قبل از بخش افزایش در **while** می‌تواند سبب بوجود آمدن یک حلقه بی‌نهایت شود.

در برنامه ۱۴–۵ از دستور **continue** در یک عبارت **for** استفاده شده (خط 12) تا از عبارت خروجی خط 14 پرش شود، زمانیکه عبارت **if** در خطوط 12-11 تشخیص دهد که مقدار **count** برابر 5 شده است. زمانیکه عبارت **continue** اجرا شود، برنامه از مابقی بدنه **for** پرش خواهد کرد. کنترل برنامه با افزایش متغیر کنترلی عبارت **for** ادامه می‌یابد و بدنبال آن شرط تکرار حلقه صورت می‌گیرد تا تعیین کند آیا بایستی حلقه ادامه یابد یا خیر.

```
1  // Fig. 5.14: fig05_14.cpp
2  // continue statement terminating an iteration of a for statement.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  int main()
8  {
9     for ( int count = 1; count <= 10; count++ ) // loop 10 times
10    {
11       if ( count == 5 ) // if count is 5,
12          continue;      // skip remaining code in loop
13
14       cout << count << " ";
15    } // end for
16
17    cout << "\nUsed continue to skip printing 5" << endl;
18    return 0; // indicate successful termination
19 } // end main
```

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing 5
```

شکل ١٤–٥ | عبارت continue در یک عبارت for.

در بخش ۳–۵ مشخص کردیم که عبارت **while** می‌توانند در بسیاری از موارد بجای عبارت **for** بکار گرفته شود. یک استثناء زمانی رخ می‌دهد که عبارت افزایش‌دهنده در ساختار **while** بدنبال دستور **continue** آمده باشد. در اینحالت، عمل افزایش قبل از تست شرط ادامه حلقه اجرا نخواهد شد.

برنامه‌نویسی ایده‌ال

تعدادی از برنامه‌نویسان احساس می‌کنند که استفاده از ***break*** و ***continue*** برخلاف قواعد برنامه‌نویسی ساخت‌یافته است، چراکه می‌توان بجای بکارگیری این عبارات از تکنیک‌های برنامه‌نویسی ساخت‌یافته استفاده کرد به نوعی که دیگر نیازی به استفاده از آنها نباشد.

کارائی

*اگر عبارات **break** و **continue** درست بکار گرفته شوند، نسبت به تکنیک‌های ساخت‌یافته سریعتر اجرا می‌شوند.*

۵–۸ عملگرهای منطقی

تا بدین جا، فقط به معرفی *شرط‌های ساده‌ای*، همچون **count <= 10**، **total > 1000** و **number != sentinelValue** پرداخته‌ایم و هر عبارت انتخاب و تکرار فقط اقدام به ارزیابی یک شرط با یکی از عملگرهای >، <، =>، =< و == و =! می‌کرد. برای تصمیم‌گیری که بر مبنای ارزیابی از چندین شرط بود، این بررسی‌ها را در عبارات جداگانه یا عبارتهای **if** یا **if…then** تودرتو انجام می‌دادیم.

به منظور رسیدگی موثرتر به شرط‌های پیچیده، C++ *عملگرهای منطقی* در نظر گرفته است. عملگرها عبارتند از **&&** (*AND* منطقی)، || (*OR* منطقی) و **!** (*NOT* منطقی، یا نفی منطقی). با طرح مثال‌های به بررسی عملکرد این عملگرها می‌پردازیم.

عملگر AND منطقی (&&)

فرض کنید می‌خواهیم از برقرار بودن دو شرط قبل از اینکه برنامه مسیر مشخص‌ی را برای اجرا انتخاب کند، مطمئن شویم. در چنین حالتی، می‌توانیم از عملگر **&&** و بصورت زیر استفاده کنیم:

```
if ( gender == 1 && age >= 65 )
   seniorFemales++;
```

این عبارت **if** متشکل از دو شرط ساده است شرط **gender == 1** تعیین می‌کند که آیا شخص مونث است یا خیر و شرط **age >= 65** مشخص می‌کند که آیا شخص شهروند مسنی است یا خیر. ابتدا این دو شرط ساده مورد ارزیابی قرار می‌گیرند، چرا که تقدم عملگرهای == و =< به نسبت **&&** در مرتبه بالاتری قرار دارند. سپس عبارت **if** به بررسی ترکیبی شرط زیر می‌پردازد

```
gender == 1 && age >= 65
```

اگر فقط و فقط اگر هر دو شرط برقرار باشند، برقراری این شرط درست ارزیابی خواهد شد. هنگامی که ترکیب این شرط درست باشد، به مقدار **seniorFemales** یک واحد افزوده می‌شود. با این وجود، اگر فقط یکی از این دو شرط یا یکی از آنها برقرار نباشد (درست نباشد)، برنامه از انجام عمل افزایش صرفنظر کرده و به اجرای برنامه پس از عبارت **if** می‌پردازد. عبارت قبل را می‌توان با استفاده از پرانتزها بهتر کرد:

```
( gender == 1 ) && ( age >= 65 )
```

خطای برنامه‌نویسی

اگر چه 7>x>3 در جبر شرط درستی است، اما بدرستی در C++ ارزیابی نمی‌شود. برای ارزیابی صحیح در C++ باید از (3 < x && x < 7) استفاده کرد.

جدول شکل ۱۵-۵ عملکرد عملگر **&&** را نشان می‌دهد. در این جدول چهار حالت ممکنه از ترکیب مقادیر *false* و *true* بر روی *عبارت۱* و *عبارت۲* لیست شده‌اند. به این نوع جداول، جدول درستی گفته می‌شود.

عبارت۱	عبارت۲	عبارت۲ && عبارت۱
false	false	false
false	true	false
true	false	false
true	true	true

شکل ۱۵-۵| جدول درستی عملگر &&.

عملگر OR منطقی (||)

حال اجازه دهید تا به بررسی عملگر || (OR شرطی) بپردازیم. فرض کنید مایل هستیم تا قبل از انجام یک عمل مشخص از برقرار بودن هر دو شرط یا یکی از شرط‌ها (درست بودن) مطمئن شویم. در بخشی از برنامه زیر، از عملگر || استفاده شده است:

```
if ((semesterAverage >= 90) || (finalExam >= 90))
        cout << "Student grade is A" << endl;
```

این عبارت هم متشکل از دو شرط ساده است. با ارزیابی شرط **semesterAverage >= 90** مشخص می‌شود که آیا دانشجو به دلیل حفظ کارایی خود در طول ترم قادر به دریافت امتیاز "**A**" بوده است یا خیر. شرط **finalExam >= 90** تعیین می‌کند که آیا دانشجو در آزمون پایان ترم امتیاز "**A**" بدست آورده یا خیر. سپس عبارت **if** به بررسی ترکیبی شرط زیر می‌پردازد

```
(semesterAverage >= 90) || (finalExam >= 90)
```

و اگر یکی از شرط‌ها یا هر دو آنها برقرار باشند، نمره "**A**" به دانشجو اهداء می‌شود. دقت کنید که فقط در صورت برقرار نبودن هر دو شرط (*false* بودن)، عبارت **"Student grade is A"** چاپ نخواهد شد. جدول شکل ۱۶-۵، جدول درستی عملگر OR منطقی است.

عبارت۱	عبارت۲	عبارت۲ || عبارت۱
false	false	false
false	true	true
true	false	true
true	true	true

شکل ١٦-٥ | جدول درستی عملگر ||

عملگر **&&** از تقدم بالاتری نسبت به عملگر || برخوردار است. هر دو عملگر از چپ به راست ارزیابی می‌شوند. یک عبارت حاوی عملگرهای **&&** یا || فقط تا زمان شناخت درست بودن یا نبودن ارزیابی می‌گردد. از اینرو، در ارزیابی عبارت

```
(gender == 1) && (age >= 65)
```

اگر **gender** معادل با **"1"** نباشد (در اینصورت کل عبارت برقرار نخواهد بود)، ارزیابی عبارت دوم بیهود خواهد بود چرا که شرط عبارت اول برقرار نیست. ارزیابی عبارت یا شرط دوم فقط زمانی رخ می‌دهد که **gender** برابر **"1"** باشد (برای برقرار بودن کل عبارت هنوز هم باید شرط **age >= 65** برقرار گردد). این ویژگی در ارزیابی عبارات **&&** و **||** ارزیابی اتصالی نامیده می‌شود. در ضمن این ویژگی سبب افزایش کارائی می گردد.

کارائی

در عبارتی که از عملگر && استفاده می‌کند و شرط‌ها از هم متمایز هستند، آن شرطی را که احتمال برقرار نبودن آن بیشتر است در سمت چپ شرط قرار دهید. در عبارتی که از عملگر || استفاده می‌کند، شرطی را که احتمال برقرار بودن آن بیشتر است در سمت چپ شرط قرار دهید. در اینحالت از ارزیابی اتصالی استفاده شده و زمان اجرای برنامه کاهش می‌یابد.

عملگر نفی منطقی (!)

عملگر**!** (نفی منطقی) به برنامه‌نویس امکان می‌دهد تا نتیجه یک شرط را "معکوس" کند. برخلاف عملگرهای منطقی **&&** و **||** که از دو شرط استفاده می کنند (عملگرهای باینری هستند)، عملگر منطقی نفی یک عملگر غیرباینری است و فقط با یک عملوند بکار گرفته می‌شود. این عملگر قبل از یک شرط جای داده می‌شود. برای مثال به عبارت زیر دقت کنید:

```
if ( !( grade == sentinelValue ) )
  cout << "The next grade is " << grade << endl;
```

وجود پرانتزها در اطراف شرط **grade == sentinelValue** ضروری است، چراکه تقدم عملگر نفی از عملگر برابری (تساوی) بالاتر است. جدول شکل ۱۷-۵ جدول درستی عملگر نفی است.

عبارت	عبارت !
false	true
true	false

شکل ۱۷-۵ | جدول درستی عملگر نفی.

مثالی از عملگرهای منطقی

برنامه شکل ۱۸-۵ به توصیف عملگرهای منطقی مطرح شده در جداول درستی می‌پردازد. در خروجی هر عبارت ارزیابی شده و نتیجه بولی آن بنمایش در آمده است. بطور پیش فرض، مقادیر بولی true و false توسط **cout** و عملگر درج بصورت 1 و 0 بنمایش در آمده‌اند. در خط 11، از **boolalpha** کنترل کننده استریم استفاده کرده‌ایم. تا مشخص کنیم که مقدار هر عبارت بولی بایستی با کلمه "true" یا "false" بنمایش در آید. برای مثال، نتیجه عبارت **false && false** در خط 12 مقدار **false** است، از اینرو در دومین خط خروجی کلمه "false" بکار گرفته شده است. خطوط 15-11 جدول درستی **&&** را

تشکیل می‌دهند. خطوط 22-18 تولید کننده جدول درستی || هستند. خطوط 27-25 جدول درستی ! را ایجاد می‌کنند.

```
1  // Fig. 5.18: fig05_18.cpp
2  // Logical operators.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::boolalpha; // causes bool values to print as "true" or "false"
7
8  int main()
9  {
10    // create truth table for && (logical AND) operator
11    cout << boolalpha << "Logical AND (&&)"
12       << "\nfalse && false: " << ( false && false )
13       << "\nfalse && true: " << ( false && true )
14       << "\ntrue && false: " << ( true && false )
15       << "\ntrue && true: " << ( true && true ) << "\n\n";
16
17    // create truth table for || (logical OR) operator
18    cout << "Logical OR (||)"
19       << "\nfalse || false: " << ( false || false )
20       << "\nfalse || true: " << ( false || true )
21       << "\ntrue || false: " << ( true || false )
22       << "\ntrue || true: " << ( true || true ) << "\n\n";
23
24    // create truth table for ! (logical negation) operator
25    cout << "Logical NOT (!)"
26       << "\n!false: " << ( !false )
27       << "\n!true: " << ( !true ) << endl;
28    return 0; // indicate successful termination
29 } // end main
```

```
Logical AND (&&)
false && false: false
false && true:  false
true && false:  false
true && true:   true

Logical OR (||)
false || false: false
false || true:  true
true || false:  true
true || true:   true

Logical NOT (!)
!false: true
!true:  false
```

شکل ۱۸-۵ | عملگرهای منطقی.

تقدم و شرکت‌پذیری عملگرها

جدول به نمایش درآمده در شکل ۱۹-۵ تقدم عملگرهای معرفی شده تا بدین جا را نشان می‌دهد. تقدم عملگرها از بالا به پایین و به ترتیب کاهش می‌یابد.

عملگر	شرکت‌پذیری	نوع
()	left to right	parentheses
++ -- static_cast<*type***>()**	right to left	unary postfix

++ -- + - !	right to left	unary prefix
* / %	left to right	multiplicative
+ -	left to right	additive
<< >>	left to right	insertion/extraction
< <= > >=	left to right	relational
== !=	left to right	equality
&&	left to right	logical AND
||	left to right	logical OR
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment
,	left to right	comma

شکل ۱۹-۵|تقدم و شرکت‌پذیری عملگرهای معرفی شده تا بدین فصل.

۹-۵ اشتباه گرفتن عملگر تساوی (==) و عملگر تخصیص (=)

یک نوع خطا وجود دارد که برنامه‌نویسان C++، بدون در نظر گرفتن میزان تجربه اکثراً با آن مواجه می‌شوند و بر این اساس یک بخش مجزا برای آن در نظر گرفته شده است. خطاهایی که بطور تصادفی با جابجا نوشتن عملگرهای == (تساوی) و = (تخصیص) رخ می‌دهند. اشتباهی که انجام آن موجب بروز خطای نحوی نمی‌شود و عباراتی که حاوی چنین اشتباهی هستند بدرستی کامپایلر می‌شوند و برنامه شروع بکار می‌کند، اما در زمان اجرا نتایج اشتباهی تولید می‌کنند و خطایی که با آن مواجه هستیم، خطای منطقی زمان اجرا است.

از دو نظر C++ به این مشکل رسیدگی می‌کند. یکی اینکه هر عبارتی که مقدار تولید می‌کند می‌تواند در بخش شرط هر عبارت کنترلی بکار گرفته شود. اگر مقدار عبارت، صفر باشد با آن همانند false و اگر مقدار عبارت، غیرصفر باشد با آن همان‌ند true رفتار می‌شود. دوم اینکه عبارت تخصیصی مقدار تولید می‌کند، یعنی مقداری به متغیر قرار گرفته در سمت چپ عملگر تخصیص، اختصاص می‌یابد. برای مثال، فرض کنید قصد نوشتن عبارت زیر را داشته باشیم

```
if ( payCode == 4 )
    cout<< "You get a bonus!" << endl;
```

اما تصادفاً بنویسیم

```
if ( paycode = 4 )
    cout << "You get a bonus!" << endl;
```

عبارت **if** اول بدرستی جایزه‌ای به شخصی که **paycode** آن برابر **4** است اهدا می‌کند. عبارت **if** دوم (که خطا دارد)، مبادرت به ارزیابی جمله تخصیص در شرط **if** با ثابت **4** می‌کند. هر مقداری غیر از صفر

بعنوان یک مقدار **true** تفسیر می‌شود، از اینرو شرط این عبارت **if** همیشه **true** بوده و همیشه این شخص جایزه دریافت می‌کند، صرفنظر از اینکه مقدار **payCode** آن چند باشد.

خطای برنامه‌نویسی

استفاده از عملگر == با هدف تخصیص و استفاده از عملگر = با هدف تساوی، خطای منطقی است.

اجتناب از خطا

معمولاً برنامه‌نویسان شرط‌هایی همانند 7 == x را با نام متغیر در سمت چپ و مقدار ثابت در سمت راست می‌نویسند. با برعکس نوشتن این ترتیب به نحوی که ثابت در سمت چپ و نام متغیر در سمت راست قرار گرفته باشد (7 == x) در صورتیکه برنامه‌نویس اشتباهاً مبادرت به نوشتن = بجای == کند، کامپایلر متوجه موضوع شده و با آن همانند یک خطای زمان کامپایل رفتار می‌کند، چرا که نمی‌توانید مقدار یک ثابت را تغییر دهید. با انجام اینکار می‌توان از رخ دادن خطاهای منطقی جلوگیری کرد.

به اسامی متغیرها، مقادیر سمت چپ (*lvalues*) گفته می‌شود چرا که از آنها در سمت چپ عملگر تخصیص استفاده می‌شود. به ثابت‌ها، مقادیر سمت راست (*rvalues*) گفته می‌شود، چرا که از آنها فقط در سمت راست عملگر تخصیص استفاده می‌شود. توجه کنید که lvalues را می‌توان به عنوان rvalues استفاده کرد، اما عکس این موضوع صادق نیست. فرض کنید برنامه‌نویس قصد دارد مقداری را به یک متغیر با یک عبارت ساده همانند عبارت زیر تخصیص دهد

```
x = 1;
```

اما بنویسد

```
x == 1;
```

در اینجا هم، خطای نحوی وجود ندارد. بجای آن، کامپایلر بسادگی مبادرت به ارزیابی عبارت رابطه‌ای می‌کند. اگر **x** معادل با 1 باشد، شرط برقرار بوده (true) و عبارت با مقدار true ارزیابی می‌شود. اگر **x** معادل 1 نباشد، شرط برقرار نبوده (false) و عبارت با مقدار **false** ارزیابی می‌شود. صرفنظر از مقدار عبارت، عملیات تخصیص صورت نمی‌گیرد و مقدار از دست می‌رود. مقدار **x** بدون تغییر باقی می‌ماند و می‌تواند در زمان اجرا، خطای منطقی تولید کند. متاسفانه روش مشخصی برای اجتناب از این مشکل وجود ندارد.

اجتناب از خطا

با استفاده از یک ویرایشگر متنی به بررسی تمام = های موجود در برنامه بپردازید و از استفاده صحیح آن مطمئن شوید.

۱۰-۵ چیکده برنامه‌نویسی ساخت‌یافته

همانند یک معمار که براساس دانش خود اقدام به طراحی ساختمان می‌کند، برنامه‌نویسان هم اقدام به طراحی و ایجاد برنامه‌ها می‌کنند و این در حالیست که دانش ما نسبت به معماری بسیار جوانتر بوده و از

ترکیب علوم مختلف ایجاد شده است.تا بدین جا آموختیم که برنامه‌نویسی ساخت‌یافته، برنامه‌های ایجاد می‌کند که درک، تست، خطایابی و اصلاح آنها به آسانی صورت می‌گیرد و اثبات این مسئله از طریق ریاضی هم ممکن است.

عبارتهای کنترلی ++C با استفاده از دیاگرام‌های فعالیت بصورت خلاصه در شکل ۲۰-۵ آورده شده‌اند. دایره‌های کوچکی که در تصاویر بکار رفته‌اند، نشان دهنده یک نقطه ورودی و یک نقطه خروجی برای هر عبارت هستند. اتصال جداگانه از نمادهای دیاگرام به صورت غیرقراردادی می‌تواند ما را به طرف برنامه‌های غیرساخت‌یافته سوق دهد. از اینرو یک برنامه‌نویس حرفه‌ای با انتخاب و ترکیب نمادها بفرمی که محدود به عبارتهای کنترل است و ایجاد برنامه‌های ساخت‌یافته به وسیله ترکیب عبارتهای کنترل در دو روش ساده می‌تواند به یک عبارت مناسب دست یابد.

برای سادگی کار، از یک نقطه ورودی و یک نقطه خروجی در عبارتهای کنترل استفاده می‌شود، از اینرو فقط یک راه برای ورود و یک راه برای خروج در هر عبارت کنترل وجود خواهد داشت. اتصال متوالی این عبارتهای کنترل و به شکل درآوردن برنامه‌های ساخت‌یافته آسان‌تر است، نقطه خروجی یک عبارت کنترل مستقیماً به نقطه ورودی یک عبارت کنترل دیگر متصل می‌شود. عبارتهای کنترل می‌توانند بصورت متوالی قرار گیرند (یکی پس از دیگری در یک برنامه) که به اینحالت *عبارت کنترل پشته‌ای* می‌گوئیم. قوانین برنامه‌نویسی ساخت‌یافته امکان کنترل عبارتهایی به نوع تودرتو یا آشیانه‌ای را فراهم می‌آورند.

شکل ۲۰-۵ | عبارتهای انتخاب و تکرار، تک ورودی/تک خروجی در ++C.

جدول شکل ۲۱-۵ حاوی قوانینی است که برای به شکل درآوردن صحیح برنامه‌های ساخت‌یافته ضروری هستند. در این قوانین فرض بر این است که نماد عمل برای ارائه هر نوع عمل اجرائی شامل ورودی/خروجی است.

قوانین شکل‌دهی برنامه‌های ساخت‌یافته

۱) شروع با ساده‌ترین دیاگرام فعالیت (شکل ۲۲-۵)

۲) هر نماد وضعیت عمل را می‌توان با دو نماد وضعیت عمل در حالت متوالی جایگزین کرد.

۳) هر وضعیت عمل را می‌توان جایگزین هر عبارت کنترلی کرد (توالی، عبارتهای **if...else**، **if**، **switch**، **while**، **do..while**، **for**)

۴) قوانین ۲ و ۳ را می‌توان هر چند بار که مایل هستیم و با هر ترتیبی تکرار کرد.

شکل ۲۱-۵ | قوانین برنامه‌نویسی ساخت‌یافته.

با بکار بردن قوانین ۲۱-۵ همیشه یک دیاگرام فعالیت مرتب و بفرم بلوکی ایجاد می‌شود. برای مثال نتیجه اعمال مکرر قانون دوم بر روی ساده‌ترین دیاگرام فعالیت (شکل ۲۲-۵)، یک دیاگرام فعالیت با تعدادی وضعیت عمل که بفرم متوالی و پشت سرهم قرار گرفته‌اند، است(شکل ۲۳-۵). دقت کنید که قانون دوم یک عبارت کنترلی پشته (بر روی هم قرار گرفته) ایجاد می‌کند، بنابراین به قانون دوم، *قانون پشته‌ای* گفته می‌شود.[

نکته:خطوط عمودی خط تیره در شکل ۲۳-۵ بخشی از UML نیستند. ما از آنها برای متمایز کردن چهار دیاگرام فعالیت استفاده کرده‌ایم که نشاندهنده قانون دوم از جدول ۲۱-۵ باشند.]

قانون سوم را قانون *تودرتو* یا *آشیانه‌ای* می‌نامند. بکار بردن قانون سوم به دفعات بر روی یک فلوچارت ساده، یک فلوچارت مرتب با عبارتهای کنترل تودرتو را نتیجه می‌دهد. برای مثال در شکل ۲۶-۵ مستطیل قرار گرفته در ساده‌ترین فلوچارت، در بار اول با یک عبارت دو انتخابی **(if/then)** جایگزین شده است. سپس مجدداً قانون سوم بر روی دو مستطیل موجود در عبارت دو انتخابی اعمال شده و هر کدام یک از این مستطیل‌ها با یک عبارت دو انتخابی جایگزین شده است. جعبه‌های خط چین که در اطراف هر عبارت دو انتخابی قرار گرفته‌اند نشان می‌دهند که بجای کدام مستطیل جایگزین شده‌اند.

شکل ۲۲-۵ | ساده‌ترین دیاگرام فعالیت.

شکل ۲۳-۵ | بکارگیری مکرر قانون دوم بر روی ساده‌ترین دیاگرام فعالیت.

شکل ۲۴-۵ نمایشی از انواع ساخت بلوکی به روش پشته است که از اعمال قانون دوم ایجاد شده و همچنین ساخت بلوکی به روش تودرتو با استفاده از قانون سوم را نشان می‌دهد. همچنین این شکل حاوی یک نوع از ساخت بلوکی روی هم قرار گرفته است که اینحالت نمی‌تواند در دیاگرام‌های فعالیت ساخت‌یافته بکار گرفته شود.

شکل ٢٤-٥ | اعمال مکرر قانون سوم بر روی ساده‌ترین دیاگرام فعالیت.

قانون چهارم یک عبارت تودرتوی بزرگتر، پیچیده‌تر و عمیق‌تر ایجاد می‌کند. دیاگرامی که با استفاده از قوانین جدول ۲۱-۵ ایجاد شود ترکیبی از تمام حالات ممکنه از دیاگرام‌های فعالیت خواهد بود و از اینرو تمام حالات برنامه‌های ساخت‌یافته را خواهد داشت. زیبائی این روش در این است که فقط با استفاده از هفت عبارت کنترلی ساده تک ورودی/تک خروجی ایجاد شده و اجازه ترکیب آنها در دو روش ساده را فراهم می‌آورد.

اگر قوانین جدول ۲۱-۵ بکار گرفته شوند، ایجاد یک دیاگرام فعالیات غیرساخت‌یافته (همانند شکل ۲۵-۵) غیرممکن خواهد بود. اگر مطمئن نیستید که یک دیاگرام ساخت‌یافته است، می‌توانید با اعمال قوانین جدول ۲۱-۵، بصورت معکوس و ساده کردن دیاگرام از این امر مطلع شوید. اگر دیاگرام قابلیت تبدیل به

یک دیاگرام فعالیت ساده را داشته باشد پس دیاگراتم اصلی ساخت‌یافته است و در غیر اینصورت ساخت‌یافته نمی‌باشد.

توسعه برنامه‌نویسی ساخت‌یافته به سادگی امکان‌پذیر است و Bohm و Jacopini نشان دادند که برای ایجاد برنامه‌ها فقط به سه شکل کنترلی نیاز است:

- توالی
- انتخاب
- تکرار

توالی حالت بدیهی دارد. انتخاب، با پیاده سازی یکی از سه روش زیر ممکن می‌شود:

- عبارت **if** (تک انتخابی)
- عبارت **if...else** (دو انتخابی)
- عبارت **switch** (چند انتخابی)

هر عبارتی که بتوان آنرا با **if...else** و **switch** نوشت، می‌توان با ترکیب عبارتهای **if** هم انجام داد (اگر چه شاید ظاهر خوبی نداشته باشد).

شکل ۲۵-۵ | دیاگرام فعالیت غیرساخت‌یافته.

تکرار می‌تواند با یکی از سه روش زیر پیاه سازی شود:

- عبارت **while**
- عبارت **do…while**
- عبارت **for**

هر عبارت تکراری را می‌توان با اعمال عبارت **while** پیاده سازی کرد (اگر چه شاید ظاهر خوبی نداشته باشد).

اگر بخواهیم از مطالب گفته شده نتیجه‌گیری نمائیم، می‌توان گفت کنترل‌های مورد نیاز در یک برنامه C++ می‌توانند موارد زیر باشند:

- توالی
- عبارت انتخاب **if**

- عبارت تکرار **while**

این عبارتهای کنترلی می‌توانند به دو روش پشته و تودرتو با یکدیگر بکار گرفته شوند که نشان از سر راست بودن و سادگی برنامه‌نویسی ساخت‌یافته دارد.

۵-۱۱ مبحث آموزشی مهندسی نرم‌افزار: شناسایی وضعیت و فعالیت شی‌ها در سیستم ATM

در بخش ۱۳-۴ مبادرت به شناسایی تعدادی از صفات کلاس مورد نیاز برای پیاده‌سازی سیستم ATM و افزودن آنها به دیاگرام کلاس در شکل ۲۴-۴ کردیم. در این بخش، نشان خواهیم داد که چگونه می‌توان این صفات را در وضعیت (حالت) یک شی عرضه کرد. همچنین مبادرت به شناسایی چندین وضعیت کلیدی خواهیم کرد که سبب اشتغال شی‌ها می‌شوند و در مورد تغییر وضعیت دادن شی‌ها در واکنش به انواع رویدادهای رخ داده در سیستم بحث خواهیم کرد. همچنین در ارتباط با روند کار، یا فعالیت‌ها صحبت می‌کنیم که شی‌ها در سیستم ATM انجام می‌دهند. فعالیت شی‌های **BalanceInquiry** و **Withdrawal** را در این بخش معرفی می‌کنیم، که از جمله فعالیت‌های کلیدی در سیستم ATM هستند (نمایش موجودی و برداشت پول).

دیاگرام‌های وضعیت ماشین

هر شی در یک سیستم در میان دنباله‌ای از وضعیت‌های مشخص شده حرکت می‌نماید. وضعیت جاری یک شی توسط مقادیر موجود در صفات شی در آن زمان مشخص می‌شود. *دیاگرام وضعیت ماشین* (که *دیاگرام وضعیت* نامیده می‌شود) مدل‌کننده وضعیت‌های کلیدی یک شی بوده و نشاندهنده شرایط محیطی است که شی در آن شرایط تغییر وضعیت می‌دهد. برخلاف دیاگرام‌های کلاس که بر روی ساختار اصلی سیستم تمرکز دارند، دیاگرام‌های وضعیت، مدل‌کننده برخی از رفتار سیستم هستند. شکل ۲۶-۵ یک دیاگرام وضعیت ساده است که برخی از وضعیت‌های یک شی از کلاس ATM را مدل کرده است. UML هر وضعیت را در یک دیاگرام وضعیت، بصورت یک *مستطیل گوشه‌گرد* با نام وضعیت جای گرفته در میان آن به نمایش در می‌آورد.

یک *دایره توپر* با یک فلش متصل شده نشاندهنده *وضعیت اولیه* می‌باشد. بخاطر دارید که این اطلاعات وضعیت را بصورت صفت **Boolean** برای **userAuthenticated** در دیاگرام کلاس شکل ۲۴-۴ مدل کرده‌ایم. این صفت با وضعیت *false* یا «کاربر تایید نشده است» بر طبق دیاگرام وضعیت مقداردهی اولیه می‌شود.

شکل ۲۶-۵ | دیاگرام وضعیت شی ATM.

جهت فلش‌ها نشاندهنده تراکنش‌های صورت گرفته مابین وضعیت‌ها هستند. یک شی می‌تواند از یک وضعیت در واکنش به رویدادهای مختلف که در سیستم رخ می‌دهند به وضعیت دیگر منتقل گردد. نام یا توصیف رویدادی که سبب تراکنش شده در کنار خطی که متناظر با تراکنش است نوشته می‌شود. برای مثال، شی ATM از وضعیت «کاربر تایید نشده» به وضعیت «کاربر تایید شده» پس از تایید کاربر از سوی پایگاه داده تغییر می‌یابد. از مستند نیازها بخاطر دارید که اعتبارسنجی کاربر از سوی پایگاه داده با مقایسه شماره حساب و PIN وارد شده از سوی کاربر با اطلاعات متناظر در پایگاه داده صورت می‌گیرد.

اگر پایگاه داده تشخیص دهد که کاربر به درستی شماره حساب و PIN را وارد کرده است، شی ATM به وضعیت «تایید کاربر» می‌رود و مقدار صفت **userAuthenticated** خود را به ***true*** تغییر می‌دهد. زمانیکه کاربر با انتخاب گزینه "**exit**" از منوی اصلی اقدام به خروج از سیستم می‌کند، شی ATM به وضعیت «کاربر تایید نشده است» باز می‌گردد و شرایط برای کاربر بعدی ATM مهیا می‌شود.

مهندسی نرم‌افزار

معمولاً طراحان نرم‌افزار هر حالت ممکنه را در دیاگرام‌های وضعیت قرار نمی‌دهند و فقط سعی در عرضه دیاگرام‌های وضعیت با اهمیت بیشتر یا وضعیت‌های پیچیده می‌کنند.

دیاگرام‌های فعالیت

همانند یک دیاگرام وضعیت، یک دیاگرام فعالی‌ت مبادرت به مدل کردن رفتار سیستم می‌کند. برخلاف دیاگرام وضعیت، دیاگرام فعالیت مبادرت به مدل‌سازی روند کار یک شی (توالی از رویدادها) در مدت زمان اجرای برنامه می‌کنند. دیاگرام فعالیت مدل‌کننده اعمال یک شی و ترتیب انجام کار توسط آن شی است. بخاطر دارید که از دیاگرام‌های فعالیت UML برای نمایش جریان کنترل عبارات کنترلی در فصل‌های ۴ و ۵ استفاده کردیم.

دیاگرام فعالیت به نمایش در آمده در شکل ۲۷-۵ فعالیت‌های انجام گرفته در مدت زمان اجرای یک تراکنش **BalanceInquiry** (نمایش موجودی) را مدل کرده است. فرض کرده‌ایم که یک شی **BalanceInquiry** در حال حاضر مقداردهی اولیه شده و یک شماره حساب معتبر به آن تخصیص یافته است، از اینرو شی می‌داند که کدام موجودی را بازیابی خواهد کرد. دیاگرام شامل اعمالی است که پس از انتخاب گزینه نمایش موجودی از سوی کاربر (از منوی اصلی) و قبل از اینکه ATM کاربر را به منوی اصلی بازگرداند، رخ می‌دهند. شی **BalanceInquiry** این اعمال را انجام نمی‌دهد، از اینرو ما آنها را در اینجا مدل نکرده‌ایم.

شکل ۲۷-۵ | دیاگرام فعالیت برای تراکنش BalanceInquiry.

دیاگرام با بازیابی موجودی در دسترس، حساب کاربر و از پایگاه داده آغاز می‌شود. سپس، **BalanceInquiry** کل موجودی را از حساب بازیابی می‌کند. در پایان تراکنش، میزان موجودی بر روی صفحه نمایش ظاهر می‌شود. با این عمل تراکنش کامل می‌شود.

UML یک عمل را در یک دیاگرام فعالیت بصورت وضعیت عمل شده با یک مستطیل که گوشه‌های چپ و راست آن حالت انحناء به خارج دارند نشان می‌دهد. هر وضعیت عمل حاوی یک جمله توضیح عمل است، برای مثال «دریافت میزان موجودی در حساب کاربر از پایگاه داده»، که مشخص می‌کند چه عملی انجام می‌شود. یک فلش، دو وضعیت عمل را به هم متصل کرده است و نشان‌دهنده ترتیب انجام اعمال است. دایره توپر (در بالای شکل ۲۷-۵) نشاندهنده وضعیت اولیه و آغاز روند کار می‌باشد. در این دیاگرام، ابتدا تراکنش مبادرت به اجرای عمل «دریافت میزان موجودی حساب کاربر از پایگاه داده» می‌کند. سپس تراکنش (مرحله دوم) مبادرت به بازیابی کل موجودی می‌نماید. در پایان تراکنش هر دو موجودی را بر روی صفحه نمایش ظاهر می‌سازد. دایره توپر احاطه شده در درون یک دایره (در پایین شکل ۲۷-۵) نشاندهنده وضعیت پایانی است، انتهای روند کار پس از اینکه شی، عمل‌های مدل شده را انجام داده است.

شکل ۲۸-۵ نمایشی از دیاگرام فعالیت برای تراکنش **Withdrawal** است (برداشت پول). فرض می‌کنیم که به شی **Withdrawal** در حال حاضر یک شماره حساب معتبر تخصیص یافته است. انتخاب گزینه برداشت پول از منوی اصلی یا برگشت دادن کاربر به منوی اصلی ATM را مدل نمی‌کنیم، چرا که این اعمال توسط شی **Wihtdrawal** صورت نمی‌گیرند. ابتدا تراکنش، منوی استاندارد میزان برداشت (شکل ۱۷-۲) و گزینه لغو تراکنش را به نمایش در می‌آورد. سپس تراکنش وارد منوی انتخابی از سوی کاربر می‌شود. اکنون جریان فعالیت به یک نماد تصمیم‌گیری می‌رسد. این نقطه تعیین‌کننده عمل بعدی بر مبنای محافظ شرط مربوطه است. اگر کاربر مبادرت به لغو تراکنش کند، سیستم پیغام مناسبی به نمایش در می‌آورد. سپس جریان لغو به یک نماد ادغام می‌رسد، مکانی که جریان فعالیت با سایر تراکنش‌های ممکنه پیوند می‌یابد. توجه کنید که یک نماد ادغام می‌تواند به هر تعداد فلش تراکنش ورودی داشته باشد، اما فقط فلش تراکنش از آن خارج می‌شود. نماد پایین دیاگرام تعیین می‌کند که آیا تراکنش باید از ابتدا تکرار شود یا خیر. زمانیکه کاربر تراکنش را لغو کند، شرط «پول پرداخت شده یا تراکنش لغو شده» برقرار گردیده، سپس تراکنش به وضعیت پایانی فعالیت می‌رسد.

اگر کاربر گزینه برداشت پول را از منو انتخاب کند، تراکنش مبادرت به تنظیم **amount** (صفتی از کلاس **Withdrawal** که قبلاً در شکل ۲۴-۴ مدل شده است) با میزان پول انتخابی از سوی کاربر می‌کند. سپس تراکنش، موجودی حساب کاربر را از پایگاه داده بدست می‌آورد (صفت **availableBalance** از شی **Account** کاربر). سپس جریان فعالیت به یک شرط دیگر می‌رسد. اگر میزان درخواستی کاربر از میزان موجودی بیشتر باشد، سیستم یک پیغام خطا در ارتباط با این موضوع به نمایش در می‌آورد. سپس کنترل با سایر جریان‌های فعالیت قبل از رسیدن به پایین‌ترین شرط موجود در دیاگرام ادغام می‌شود. شرط «پول پرداخت نشده و کاربر تراکنش را لغو نکرده است» برقرار می‌شود و از اینرو جریان فعالیت به بالای دیاگرام برگشت داده شده و تراکنش به کاربر اعلان می‌کند تا مقدار جدیدی وارد سازد.

اگر مقدار درخواستی کمتر یا برابر میزان موجودی کاربر باشد، تراکنش مبادرت به تست اینکه آیا پرداخت‌کننده اتوماتیک به میزان کافی پول نقد برای انجام تقاضای صورت گرفته دارد یا خیر انجام می‌دهد. اگر چنین نباشد، تراکنش یک پیغام خطای مناسب به نمایش در آورده و به نماد ادغام قبل از آخرین نماد تصمیم‌گیری منتقل می‌شود. چون پولی پرداخت نشده از اینرو جریان فعالیت به ابتدای دیاگرام فعالیت برگشت داده می‌شود و تراکنش به کاربر اطلاع می‌دهد تا مقدار جدیدی وارد سازد.

اگر پول به میزان کافی در اختیار باشد، تراکنش با پایگاه داده وارد تعامل شده و به میزان پول درخواستی، حساب کاربر را بدهکار می‌کند (از مقدار موجود در هر دو صفت **availableBalance** و **totalBalance** از شی **Account** کم می‌شود). سپس تراکنش مبادرت به پرداخت پول مورد تقاضا کرده و به کاربر فرمان می‌دهد تا پول را از دستگاه بردارد. سپس جریان اصلی با دو جریان خطا و جریان لغو ادغام می‌شود. در اینحالت، پول پرداخت شده است، از اینرو جریان فعالیت به وضعیت پایانی رسیده است.

شکل ۲۸-۵ | دیاگرام فعالیت برای تراکنش Withdrawal.

اولین گام‌ها را برای مدل کردن رفتار سیستم ATM و نحوه نمایش صفات در ضمن فعالیت را برداشته‌ایم. در بخش ۲۲-۶ به عملیات کلاس‌ها رسیدگی می‌کنیم تا مدل کاملتری از رفتار سیستم ایجاد نمائیم.

تمرینات خودآزمایی مبحث مهندسی نرم‌افزار

۱-۵ تعیین کنید آیا عبارت زیر صحیح است یا اشتباه. در صورت اشتباه بودن علت را توضیح دهید: دیاگرام وضعیت مدل‌کننده جنبه‌های ساختاری یک سیستم است.

۵-۲ دیاگرام فعالیت مدل‌کننده ـــــــــــاست که یک شی انجام می‌دهد و ترتیب انجام آن را مشخص می‌کند.

a) اعمال

b) صفات

c) وضعیت

d) وضعیت تراکنش

۵-۳ براساس مستند نیازها، یک دیاگرام فعالیت برای تراکنش سپرده‌گذاری ایجاد کنید.

پاسخ خودآزمایی مبحث آموزشی مهندسی نرم‌افزار

۵-۱ اشتباه. دیاگرام‌های وضعیت مدل‌کننده برخی از رفتار سیستم هستند.

۵-۲ a

۵-۳ شکل ۲۹-۵ دیاگرام فعالیت برای تراکنش سپرده‌گذاری است. دیاگرام، مدل‌کننده اعمالی است که پس از انتخاب گزینه سپرده‌گذاری از منوی اصلی و قبل از بازگرداندن کاربر به منوی اصلی توسط ATM است.

شکل ۲۹-۵| دیاگرام فعالیت برای تراکنش سپرده‌گذاری (Deposit).

خودآزمایی

۵-۱ کدامیک از عبارات زیر صحیح و کدامیک اشتباه است. اگر عبارتی اشتباه است علت آنرا توضیح دهید.

a) عبارت **default** باید در عبارت انتخاب **switch** بکار گرفته شود.

b)عبارت **break** در حالت **default** ساختار **switch** برای خروج صحیح از **switch** ضروری است.

c) عبارت **x > y && a < b** برقرار است که خواه **x > y** برقرار باشد یا نباشد یا اینکه **a < b** برقرار باشد.

d) عبارتی که حاوی عملگر || است، در صورتیکه یکی از عملوندها یا هر دو عملوند برقرار (صحیح) باشند، کلاً صحیح ارزیابی می‌شود.

۵-۲ عبارت یا عباراتی در C++ بنویسید که موارد خواسته شده زیر را برآورده سازند:

a) محاسبه مجموع اعداد فرد از 99 تا 1 با استفاده از یک عبارت **for**. با فرض اینکه متغیرهای **sum** و **count** از نوع صحیح اعلان شده‌اند.

b) چاپ مقدار 333.546372 در میدانی به پهنای 15 کارکتر با دقت 2 ,1 و 3. هر عدد را در یک خط چاپ کنید.هر عدد را در میدان خود از چپ تراز کنید.

c) مقدار 2.5 را بتوان 3 برسانید. با استفاده از تابع **pow**. نتیجه را با دقت 2 در میدانی به پهنای 10 چاپ کنید.

d) چاپ اعداد از 20 تا 1 با استفاده از یک حلقه **while** و متغیر شمارنده **x**. با فرض اینکه متغیر **x** اعلان شده اما مقداردهی اولیه نشده است. هر پنج عدد در یک سطر چاپ شود.

e) تمرین c را با استفاده از عبارت **for** تکرار کنید.

۵-۳ خطا یا خطاهای موجود در کدهای زیر را یافته و آنها را اصلاح کنید.

a)

```
x=1;
while(x,=10);
  x++;
}
b)
for(y=1; y!=1.0;y+=.1)
   count<<y<<endl;
c)
switch(n)
{
 case1:
     cout<< "The number is 1"<<endl;
 case2:
     cout<<"The number is 2"<<endl;
     break;
 default:
     cout<<"The number is not 1 or 2"<<endl;
     break;
}
```

d) کد زیر باید مقادیر ۱ تا ۱۰ را چاپ کند

```
n=1;
while(n<10)
 cout<<n++<<endl;
```

پاسخ خودآزمایی

۵-۱ a) اشتباه. استفاده از **default** اختیاری است. b) اشتباه. عبارت break برای خروج از ساختار switch بکار گرفته می‌شود. c) اشتباه. به هنگام استفاده از عملگر && باید هر دو عبارت رابطه‌ای برای برقرار بودن کل عبارت، برقرار باشند. d) صحیح.

۵-۲

```
a)
sum=0;
for(count=1;count<=99;count+=2)
   sum+=count;
b)
cout<<fixed<<left
<<setprecision(1)<<setw(15)<<333.5467372
<< setprecision(2)<<setw(15)<<333.5467372
<< setprecision(3)<<setw(15)<<333.5467372
<<endl;
```

خروجی

```
333.5      333.55      333.546
```

```
c)
cout<<fixed<<setprecision(2)
      <<setw(10)<<pow(2.5,3)
      <<endl;
```

خروجی

```
15.63
```

```
d)
x=1;
while(x<=20)
{
 cout<<x;
 if(x%5==0)
   cout<<endl;
 else
    cout<<'\t';
 x++;
}
e)
for(x=1;x<=20;x++)
{
 cout<<x;
 if(x%5==0)
  cout<<endl;
 else
  cout<<'\t';
}
```

یا

```
 for(x=1;x<=20;x++)
{
 if(x%5==0)
  cout<<x<<endl;
 else
  cout<<x<<'\t';
}
```

۵-۳

a)

خطا: سیمکولن پس از سرآیند while، حلقه بی‌نهایت بوجود می‌آورد.

ااصلاح: جایگزین کردن سیمکولن با یک } یا حذف ; و {

b)

خطا: استفاده از یک عدد اعشاری در کنترل عبارت تکرار for

اصلاح: استفاده از یک عدد صحیح و انجام محاسبه مناسب به ترتیبی که مقادیر مورد نظر بدست آیند.

```
For(y=1; y!=10;y++)
 cout<<(static_cast<double>(y)/10)<<endl;
```

c)

خطا: فاقد عبارت break در اولین case.

ااصلاح: افزودن یک break در انتهای عبارت اولین case. توجه کنید در صورتیکه برنامه‌نویس مایل بوده باشد که عبارت case 2: همیشه پس از اجرای case 1: اجرا شود، اینکار خطا محسوب نمی‌شود.

d)

خطا: از عملگر رابطه‌ای صحیحی استفاده نشده است (در شرط تکرار حلقه while).

ااصلاح: استفاده از => بجای > یا تغییر 10 به 11.

تمرینات

۵-۴ خطا یا خطاهای موجود در عبارات زیر را پیدا کنید:

a)
```
for (x=100, x>=1,x++)
      cout << x << endi;
```

b) کد زیر باید مقادیر صحیح زوج یا فرد را چاپ کند

```
switch (value %2)
{
case 0:
    cout<< "Even integer" << endl;
case 1:
    cout << " Odd integer" << endl;
}
```

c) کد زیر باید مقادیر فرد ۱۹ تا ۱ را چاپ کند

```
for (x=19; x>=1; x+=2)
    cout << x << endl;
```

d) کد زیر باید مقادیر زوج ۲ تا ۱۰۰ را چاپ کند.

```
courter= 2;
do
{
cout << counter << endl;
counter +=2;
} while (counter < 100);
```

۵-۵ برنامه‌ای بنویسید که با استفاده از یک عبارت for مجموع، توالی از مقادیر صحیح را محاسبه کند. فرض کنید که نخستین عدد، نشاندهنده تعداد مقادیری باشد که وارد خواهد شد. برنامه شما باید در هر عبارت ورودی یک مقدار دریافت کند. برای مثال، نمونه می‌تواند بصورت زیر باشد

```
5   100   200   300   400  500
```

در این دنباله، 5 نشان می‌دهد که پنج مقدار وارد و جمع خواهند شد.

۵-۶ برنامه‌ای بنویسید که با استفاده از یک عبارت **for** مبادرت به محاسبه و چاپ میانگین چندین مقدار صحیح کند. فرض کنید مقدار مراقبتی 9999 باشد. یک ورودی نمونه می‌تواند بصورت زیر باشد.

```
10   8   11   7   9   9999
```

۵-۷ برنامه زیر چه کاری انجام می‌دهد؟

```
// Exercise 5.7: ex05_07.cpp
// What does this program print?
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

int main()
{
   int x; // declare x
   int y; // declare y

   // prompt user for input
   cout << "Enter two integers in the range 1-20: ";
```

```
   cin >> x >> y; // read values for x and y

   for ( int i = 1; i <= y; i++ ) // count from 1 to y
   {
      for ( int j = 1; j <= x; j++ ) // count from 1 to x
         cout << '@'; // output @

      cout << endl; // begin new line
   } // end outer for

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

۵-۸ برنامه‌ای بنویسید که با استفاده از یک عبارت for کوچکترین مقدار را از میان چندین مقدار صحیح پیدا کند. فرض کنید که اولین مقدار، نشاندهنده تعداد مقادیر باقیمانده باشد و این مقدار در مقایسه شرکت نمی‌کند.

۵-۹ برنامه‌ای بنویسید که با استفاده از یک عبارت for مبادرت به محاسبه و چاپ حاصلضرب اعداد فرد قرار گرفته از 1 تا 15 نماید.

۵-۱۰ برنامه‌ای بنویسید که مقدار فاکتوریل اعداد از 5 تا 1 را محاسبه کنید. خروجی برنامه را در فرمت جدولی به نمایش درآورید.

۵-۱۱ برنامه محاسبه سود مطرح شده در بخش ۴-۵ را به نحوی تغییر دهید که محاسبه سود را برای درصدهای 5, 6, 7, و 10 بدست آورد. از یک عبارت for برای ایجاد تغییر در سود استفاده کنید.

۵-۱۲ برنامه‌ای بنویسید که الگوهای زیر را بصورت جداگانه و زیر هم به نمایش درآورد. از حلقه‌های **for** برای ایجاد الگوها استفاده کنید. برنامه را به نحوی تغییر دهید تا هر چهار الگو را با یکدیگر و در کنار هم چاپ کند.

```
   (A)          (B)          (C)          (D)

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

۵-۱۳ یکی از برنامه‌های جالب کامپیوتری ترسیم گراف‌ها و نمودارها است. برنامه‌ای بنویسید که پنج عدد دریافت کند (هر یک مابین 1 و 30). فرض کنید که کاربر فقط مقادیر معتبر وارد می‌سازد. برای هر عددی که خوانده می‌شود، بایستی برنامه یک خط حاوی کاراکتر ستاره برابر با آن عدد چاپ کند. برای مثال اگر برنامه عدد 7 را دریافت کند، باید ******** را چاپ نماید.

۵-۱۴ یک فروشگاه که توسط سیستم پستی تجارت می‌کند پنج محصول مختلف عرضه می‌کند که قیمت خرده‌فروشی آنها به ترتیب زیر است:
محصول 1 به قیمت 2.98$، محصول 2 به قیمت 4.50$، محصول 3 به قیمت 9.98$، محصول 4 به قیمت 4.49$ و محصول 5 به قیمت 6.87$. برنامه‌ای بنویسید که جهت مقادیر را بصورت زیر دریافت کند:
a) شماره محصول
b) تعداد فروخته شده

در این برنامه باید از یک عبارت **switch** برای تعیین قیمت خرده‌فروشی برای هر محصول استفاده کنید. برنامه باید مبلغ کل فروش تمام محصولات را محاسبه و به نمایش در آورد. از یک حلقه کنترل مراقبتی برای تعیین زمانیکه باید حلقه متوقف شده و نتایج نهایی به نمایش در آیند، استفاده کنید.

۵-۱۵ برنامه GradeBook شکل‌های ۵-۹ الی ۵-۱۱ را به نحوی تغییر دهید که میانگین را برای مجموعه امتیازات محاسبه نماید. امتیاز A دارای ارزش 4، امتیاز B دارای ارزش 3 و الی آخر است.

۵-۱۶ برنامه شکل ۵-۶ را به نحوی اصلاح کنید که فقط از مقادیر صحیح برای محاسبه نرخ سود استفاده کند.

۵-۱۷ با فرض i=1، j=2، k=3 و m=2 باشد. هر کدامیک از عبارت زیر چه چیزی چاپ خواهند کرد؟ آیا وجود پرانتزها ضروری است؟

a) `cout << (i==1) << endl;`
b) `cout << (i==3) << endl;`
c) `cout << (i>=1 && j<4) << endl;`
d) `cout << (m<=99 && k<m) << endl;`
e) `cout << (j>=i || k==m) << endl;`
f) `cout << (k+m<j || 3-j>=k) << endl;`
g) `cout << (!m) <<endl;`
h) `cout << (!(j-m)) << endl;`
i) `cout << (!(K>m)) << endl;`

۵-۱۸ برنامه‌ای بنویسید که یک جدول از اعداد باینری، اکتال و هگزا دسیمال معادل با اعداد دسیمال در محدوده 1 الی 256 را به نمایش درآورد.

۵-۱۹ مقدار π را از دنباله بی‌نهایت زیر محاسبه کنید:

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{11} + \ldots$$

مقدار π را پس از 1000 عبارت اول در این دنباله، چاپ کنید.

۲۰-۵ (سه‌گانه فیثاغورث) یک مثلث راست گوشه می‌تواند اضلاعی داشته باشد که همگی از نوع صحیح باشند. به این سه نوع ضلع سه‌گانه فیثاغورث می‌گویند. این سه ضلع باید رابطه‌ای را برآورد سازند که در آن مجموع مربع دو ضلع برابر با مربع وتر باشد. تمام مثلث‌های راست گوشه برای side1, side2 و hypotenuse (وتر) را که بزرگتر از 500 نمی‌باشند را پیدا کنید. از یک **for** تودرتو استفاده نمائید.

۲۱-۵ در شرکتی پرداختی‌ها برای مدیران (کسانی که حقوق هفتگی ثابت دریافت می‌کند)، کارگران ساعتی (کسانی که یک دستمزد ثابت ساعتی برای کار تا سقف 40 ساعت دریافت کرده و برای هر ساعت اضافه کاری 1.5 برابر دستمزد ساعتی دریافتی دارند)، کارگران کمیسیون (کسانی که $250 به همراه 5.7 درصد از حقوق ناخالص هفتگی دریافت می‌کنند) یا مقاطعه‌کارها (کسانی که یک مبلغ ثابت برای هر ایتم تولیدی دریافت می‌کنند، هر مقاطعه کار در این شرکت فقط بر روی یک نوع ایتم کار می‌کند) صورت می‌گیرد. برنامه‌ای بنویسید که حقوق هفتگی هر کارمند را محاسبه کند. در ابتدای کار از تعداد کارمندان اطلاعی ندارید. هر کارمندی دارای کد پرداختی متعلق به خود است. مدیران دارای کد 1، کارگران ساعتی دارای کد 2، کارگران کمیسیون دارای کد 3 و مقاطعه‌کاران دارای کد 4 هستند. با استفاده از یک عبارت **switch** مبادرت به محاسبه حقوق هر کارمند براساس کد کنید. در درون **switch** به کاربر اعلان کنید تا اطلاعات مورد نیاز برای محاسبه حقوق کارمند را وارد سازد.

۲۲-۵ (قوانین دمرگان) در این فصل در مورد عملگرهای منطقی **&&**، **||** و **!** صحبت کردیم. گاهی اوقات استفاده از قوانین دمرگان می‌تواند نحوه استفاده از عبارات منطقی را مناسبتر سازد. این قوانین نشان می‌دهند که عبارت **!(conditional1 && conditional2)** بطور منطقی برابر است با عبارت **(!conditional1||!conditional2)**. همچنین عبارت **!(conditional1||conditional2)** بطور منطقی برابر است با عبارت **(!conditional1 && !conditional2)**. با استفاده از قوانین دمرگان عبارات معادل برای هر یک از موارد زیر بنویسید، سپس برنامه‌ای بنویسید تا نشان دهد که عبارت اصلی و عبارت جدید در هر مورد با هم برابر هستند.

a) `!( x < 5 ) && !( y >= 7)`
b) `!( a == b ) || !( g != 5 )`
c) `!( ( x <= 8 ) && ( y> 4 ) )`
d) `!( ( i > 4 ) || ( j <= 6 ) )`

۲۳-۵ برنامه‌ای بنویسید که لوزی شکل زیر را چاپ کند.

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

۵-۲۴ برنامه نوشته شده در تمرین ۵-۲۳ را به نحوی تغییر دهید تا یک عدد فرد در محدودهٔ 1 تا 19 که مشخص کننده تعداد سطرها در لوزی است دریافت کرده، سپس لوزی را با آن سایز به نمایش در آورد.

۵-۲۵ انتقادی که از عبارات break و continue می‌شود این است که آنها را غیرساختیافته می‌دانند. در واقع می‌توان همیشه این عبارات را با عبارات ساختیافته جایگزین کرد، اگر چه شاید انجام چنین کاری چندان استادانه نباشد. توضیح دهید که چگونه می‌توانید هر عبارت break را از حلقه‌ای در یک برنامه حذف کرده و آن را با یک عبارت معادل ساختیافته جایگزین کنید.

۵-۲۶ این بخش از کد چه کاری انجام می‌دهد؟

```
for ( int i = 1; i <= 5; i++ )
{
   for ( int j = 1; j <= 3; j++ )
   {
      for ( int j = 1; k <= 4; j++ )
         cout << '*';

      cout << endl ;
   } // end inner for

   cout << endl;
} // end outer for
```

۵-۲۷ توضیح دهید که به چه روشی می‌توان هر عبارت **continue** را از یک حلقه در برنامه حذف کرده و آن را با یک عبارت ساختیافته جایگزین کرد.

۵-۲۸ برنامه‌ای بنویسید که از عبارات تکرار و switch برای چاپ آهنگ The Twelve Days of Chrismas استفاده کند. باید از یک عبارت switch برای چاپ روز (یعنی "First" , "Second" ,...) استفاده شود. باید از یک عبارت switch جداگانه برای چاپ مابقی شعر استفاده کنید. برای داشتن شعر کامل این آهنگ می‌توانید به وب‌سایت www.12days.com/ library/carols/12dayssofxmas مراجعه کنید.

۵-۲۹ مطابق یک افسانه، Peter Minuit در سال 1626 جزیره Manhattan را به قیمت 24 دلار خریداری کرده است. آیا وی سرمایه‌گذاری خوبی انجام داده است؟ برای پاسخ دادن به این سوال برنامه محاسبه سود مطرح شده در شکل ۵-۶ را برای شروع کار با سرمایه اصلی $24.00 تنظیم کرده و مبلغ سود سپرده‌گذاری را تا این سال محاسبه کنید.