

فصل هفتم

آرایه‌ها و بردارها

اهداف

- استفاده از ساختمان داده آرایه برای عرضه مجموعه‌ای از ایتم‌های داده مرتبط باهم.
- استفاده از آرایه برای ذخیره سازی، مرتب سازی و جستجوی لیست‌ها و جداول.
- اعلان آرایه، مقداردهی اولیه آرایه‌ها و مراجعه به عناصر مختلف آرایه.
- ارسال آرایه‌ها به توابع.
- تکنیک‌های اولیه جستجو و مرتب سازی.
- اعلان و کار با آرایه‌های چندبعدی.
- استفاده از الگوی `vector` از کتابخانه استاندارد `C++`.



رئوس مطالب	
۷-۱	مقدمه
۷-۲	آرایه‌ها
۷-۳	اعلان آرایه‌ها
۷-۴	مثال‌هایی از کاربرد آرایه
۷-۵	ارسال آرایه به توابع
۷-۶	مبحث آموزشی: کلاس GradeBook با استفاده از آرایه برای ذخیره‌سازی نمرات
۷-۷	جستجوی آرایه‌ها: جستجوی خطی
۷-۸	مرتب‌سازی آرایه‌ها
۷-۹	آرایه‌های چند بعدی
۷-۱۰	مبحث آموزشی: کلاس GradeBook با استفاده از آرایه دو بعدی
۷-۱۱	مبحث آموزشی مهندسی نرم‌افزار: همکاری ماینین شی‌های سیستم ATM

۷-۱ مقدمه

در این فصل به معرفی یکی از مباحث مهم در ساختمان‌های داده‌ها می‌پردازیم، کلکسیونی از ایتم‌های داده مرتبط باهم. آرایه‌ها، ساختمان‌های داده متشکل از ایتم‌های داده مرتبط بهم و از یک نوع هستند. از فصل سوم با کلاس‌ها آشنا هستید. در فصل ۹، در ارتباط با نظریه ساختمان بحث خواهیم کرد. ساختمان‌ها و کلاس‌های دو قادر به نگهداری ایتم‌های داده مرتبط هستند که این داده‌ها می‌توانند از نوع‌های مختلف باشند. آرایه‌ها، ساختمان‌ها و کلاس‌ها از موجودیت‌های «استاتیک» محسوب می‌شوند که در اینحالت سایز آنها در مدت زمان اجرای برنامه ثابت باقی می‌مانند. البته می‌توان به کمک کلاس ذخیره‌سازی اتوماتیک در این روش اعمال نفوذ کرد.

پس از بحث در مورد نحوه اعلان، ایجاد و مقداردهی اولیه آرایه‌ها، در این فصل به بررسی چندین مثال کاربردی خواهیم پرداخت که نحوه کار با آرایه‌ها را نشان می‌دهند. سپس به توضیح نحوه ارائه رشته‌های کاراکتری توسط آرایه‌های کاراکتری می‌پردازیم. مثالی در ارتباط با جستجوی آرایه‌ها به منظور یافتن عناصر خاصی در یک آرایه مطرح می‌کنیم. همچنین در این فصل به معرفی یکی از مهمترین برنامه‌های کاربردی در علم کامپیوتر می‌پردازیم، که مرتب‌سازی داده‌ها می‌باشد. دو بخش از این فصل اختصاص به مبحث آموزشی کلاس GradeBook مطرح شده در فصل‌های ۶ الی ۳ دارد. در واقع، از آرایه‌ها به نحوی استفاده شده تا کلاس قادر به نگهداری مجموعه‌ای از نمرات در حافظه شده و بتواند این نمرات را تجزیه و تحلیل نماید، دو قابلیتی که در کلاس GradeBook نسخه‌های قبلی وجود نداشت. این مثال‌ها و مثال‌های



دیگر این فصل به توضیح روشی می‌پردازند که در آن آرایه‌ها به برنامه‌نویس امکان سازماندهی و کنترل بر روی داده‌ها را می‌دهند.

سبک آرایه که در سرتاسر این فصل از آن استفاده کردایم، سبک آرایه‌های مبتنی بر اشاره‌گر در C است. در فصل هشتم با اشاره‌گرها آشنا خواهید شد. بخش پایانی این فصل در ارتباط با شی‌هایی بنام بردار (vector) است که تکامل یافته آرایه‌ها می‌باشد. متوجه خواهید شد که این آرایه‌ها مبتنی بر شی به نسبت آرایه‌های مبتنی بر اشاره‌گر سبک C این‌تر و تطبیق‌پذیرتر هستند.

۲-۲ آرایه‌ها

یک آرایه گروهی از مکان‌های حافظه پشت سر هم هم نوع می‌باشند. برای اشاره به یک مکان مشخص یا عنصری در یک آرایه، نام آرایه و سپس شماره مکان یا موقعیت عنصر مورد نظر آورده می‌شود.

در شکل ۲-۱ یک آرایه از نوع صحیح با نام **c** نشان داده شده است. این آرایه حاوی ۱۲ عنصر است که هر کدام را می‌توان با بکار بردن نام آرایه و شماره موقعیت در درون یک جفت برآکت [] مورد مراجعه قرار داد. اولین عنصر در هر آرایه، عنصر صفر نامیده می‌شود، از این‌رو اولین عنصر در آرایه **c**، بصورت **c[0]**، دومین عنصر بصورت **c[1]**، و هفتمین عنصر بصورت **c[6]** و الی آخر مورد مراجعه قرار می‌گیرد. در حالت کلی عنصر **i**ام در آرایه **c** بصورت **c[i]** مورد مراجعه قرار می‌گیرد. به عددی که در درون برآکت‌ها آورده می‌شود، شاخص (یا ساب‌اسکریپت) گفته می‌شود. شاخص باید یک عدد صحیح یا عبارت صحیحی باشد. اگر برنامه‌ای از یک عبارت بعنوان شاخص استفاده کند، ابتدا این عبارت برای تعیین مقدار شاخص ارزیابی می‌شود. برای مثال، اگر متغیر **a** معادل ۵ باشد و متغیر **b** معادل ۶، عبارت

$$c[a + b] = 2;$$

اقدام به افزودن عدد ۲ به عنصر یازدهم آرایه **c[11]** خواهد کرد. اجازه دهید تا از نزدیک به بررسی آرایه **c** در شکل ۲-۱ پردازیم. نام کل آرایه **c** است. به ۱۲ عنصر این آرایه بصورت **c[0]** تا **c[11]** می‌توان دسترسی پیدا کرد. مقدار **c[0]** برابر ۴۵، مقدار **c[1]** برابر ۶، مقدار **c[2]** برابر ۰، مقدار **c[7]** برابر ۶۲ و مقدار **c[11]** برابر ۷۸ است. مقادیر ذخیره شده در آرایه‌ها از قابلیت بکارگیری در محاسبات برنامه‌های مختلف دارا هستند.

برای مثال، برای چاپ مجموع مقادیر در سه عنصر ابتدایی آرایه **c** می‌توانیم از عبارت زیر استفاده کنیم:

```
cout << c[0] + c[1] + c[2] << endl;
```



برای تقسیم مقدار، هفتمین عنصر آرایه c به ۲ و تخصیص نتیجه به متغیر x می‌توانیم از عبارت زیر استفاده کنیم:

$$x = c[6] / 2;$$

نام آرایه (دقت کنید که تمام عناصر این آرایه دارای نام یکسان c هستند)

شماره موقعیت (شاخص یا ساب‌اسکریپت)

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78



شکل ۲-۷ آرایه‌ای حاوی ۱۲ عنصر.

خطای برنامه‌نویسی



توجه به تفاوت موجود مابین «هفتمین عنصر آرایه» و «عنصر هفتم آرایه» مهم است. شاخص آرایه‌ها با صفر شود، از اینرو «هفتمین عنصر آرایه» دارای شاخص ۶ است در حالیکه «عنصر هفتم آرایه» دارای شاخص ۷ می‌باشد و در واقع هشتمین عنصر آرایه است. متأسفانه این تفاوت غالباً موجب رخدادن خطای *off-by-one* می‌شود. برای اجتناب از چنین خطایی، به عناصر آرایه بطور صریح و با نام آرایه و شماره شاخص مراجعه می‌کنیم (برای مثال $[6]$ یا $[7]$).

در واقع براکت‌های در برگیرنده شاخص یک آرایه، یک عملگر در C++ هستند. براکت‌ها تقدم یکسان با پرانتزها دارند. در جدول شکل ۷-۲ الیت و تقدم عملگرها معرفی شده تا بین مرحله آورده شده است. در این جدول نمایش اولویت‌ها از بالا به پایین است.

عملگر	نوع	ارتباط
۰ []	چپ به راست	حداکثر
++ -- ! static_cast<type>(operand)	چپ به راست	غیرباینی
++ -- + - !	راست به چپ	
* / %	چپ به راست	تعددی
+ -	چپ به راست	افزاینده کاهنده
<<>>	چپ به راست	درج/استخراج
< <= > >=	چپ به راست	رابطه‌ای
== !=	چپ به راست	برابری



عملگر	ارتباط	نوع
&&	چپ به راست	AND منطقی
	چپ به راست	OR منطقی
?:	راست به چپ	شرطی
= += *= /= %=	راست به چپ	تخصیصی
،	چپ به راست	کاما

شکل ۷-۲ | تقدم و الوبت عملگرها.

۷-۳ اعلان آرایه‌ها

آرایه‌ها اشغالگر فضای حافظه هستند. برنامه‌نویس تعیین کننده نوع عناصر و تعداد آنها بصورت زیر است: **[سایز آرایه] نام آرایه نوع**؛ و کامپایلر میزان فضایی مورد نیاز برای آرایه را رزرو می‌کند. «سایز آرایه» باید یک عدد صحیح بزرگتر از صفر باشد. برای مثال، برای اینکه کامپایلر 12 عنصر برای یک آرایه صحیح بنام **c** رزرو کند، از اعلان زیر استفاده می‌کنیم.

```
int c[12]; // c is an array of 12 integers
```

می‌توان با یک اعلان برای چندین آرایه، حافظه رزرو کرد. در اعلان زیر مبادرت به رزرو 100 عنصر برای آرایه صحیح **b** و 27 عنصر برای آرایه صحیح **x** شده است.

```
int b[100], // b is an array of 100 integers
*x[27]; // x is an array of 27 integers
```

برنامه‌نویسی ایده‌آل



به منظور افزایش خوانایی، اصلاح پذیری آسانتر و نوشتن راحت توضیحات، ترجیح می‌دهیم که یک آرایه در هر اعلان، اعلان گردد.

آرایه‌ها قادر به نگهداری مقادیری هستند که از آن نوع اعلان شده‌اند. برای مثال، یک آرایه از نوع **char** می‌تواند برای ذخیره‌سازی یک رشته کاراکتری بکار گرفته شود. تا بدین مرحله، از شی‌های **string** برای ذخیره سازی رشته‌های کاراکتری استفاده کرده‌ایم. در بخش ۷-۴ به معرفی نحوه استفاده از آرایه‌های کاراکتری برای ذخیره سازی رشته‌ها خواهیم پرداخت.

۷-۴ مثال‌هایی از کاربرد آرایه

در این بخش به ارائه چندین مثال می‌پردازیم که نحوه اعلان، تخصیص و مقداردهی آرایه‌ها و همچنین کار با عناصر آرایه‌ها را نشان می‌دهد.

اعلان آرایه و استفاده از یک حلقه برای مقداردهی اولیه عناصر آرایه

در برنامه شکل ۷-۳ مبادرت به اعلان آرایه صحیح **n** با 10 عنصر شده است (خط 12). در خطوط 15-16 از یک عبارت **for** برای مقداردهی اولیه عناصر آرایه با صفر استفاده شده است. اولین عبارت خروجی



(خط 18) نشان‌دهنده سرآیندهای ستون است که عبارت `for` موجود در خطوط 21-22 عناصر و مقادیر آرایه را با فرمت جدولی در زیر آنها چاپ می‌کند. بخاطر دارید که `setw` تصریح کننده طول میدان است.

```

1 // Fig. 7.3: fig07_03.cpp
2 // Initializing an array.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include <iomanip>
8 using std::setw;
9
10 int main()
11 {
12     int n[ 10 ]; // n is an array of 10 integers
13
14     // initialize elements of array n to 0
15     for ( int i = 0; i < 10; i++ )
16         n[ i ] = 0; // set element at location i to 0
17
18     cout << "Element" << setw( 13 ) << "Value" << endl;
19
20     // output each array element's value
21     for ( int j = 0; j < 10; j++ )
22         cout << setw( 7 ) << j << setw( 13 ) << n[ j ] << endl;
23
24     return 0; // indicates successful termination
25 } // end main

```

Element	Value
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

شکل ۷-۳ | مقداردهی اولیه عناصر آرایه با صفر و چاپ آرایه.

مقداردهی اولیه آرایه در زمان اعلان با لیست مقداردهی

می‌توان عناصر آرایه را در زمان اعلان آرایه و با قرار دادن نماد مساوی پس از نام آرایه و یک لیست جدا شده با کاما (قرار گرفته در میدان اکولادها، { }) مقداردهی اولیه کرد. در برنامه شکل ۷-۴ از یک لیست مقداردهی اولیه برای مقداردهی یک آرایه صحیح با 10 مقدار (خط 13) و چاپ آرایه با فرمت جدولی (خطوط 15-19) استفاده شده است.

اگر مقداردهی اولیه کمتر از تعداد عناصر آرایه باشد، مابقی عناصر آرایه با صفر مقداردهی خواهند شد. برای مثال، عناصر آرایه `n` در شکل ۷-۳ را می‌توان با عبارت زیر، تماماً با صفر مقداردهی اولیه کرد

```
int n[10] = {0}; // initialize elements of array n to 0
```

در این اعلان بصورت صریح اولین عنصر با صفر مقداردهی اولیه شده و 9 عنصر باقیمانده بصورت ضمنی با صفر مقداردهی اولیه می‌شوند، چرا که تعداد مقداردهی اولیه کمتر از تعداد عناصر آرایه است.



آرایه‌های اتوماتیک بصورت ضمنی با صفر مقداردهی اولیه نمی‌شوند، در حالیکه آرایه‌های استاتیک می‌توانند چنین کاری کنند. برنامه نویس بایستی حداقل اولین عنصر آرایه را در لیست مقداردهی اولیه، با صفر مقداردهی کند تا مابقی عناصر باقیمانده از آرایه بصورت ضمنی با صفر مقداردهی اولیه شوند. روش مقداردهی اولیه عرضه شده در برنامه ۳-۷ در هر بار اجرای برنامه بکار گرفته می‌شود.

اگر سایز آرایه به هنگام اعلان به همراه یک لیست مقداردهی اولیه، از قلم یافتد، کامپایلر بر حسب تعداد عناصر موجود در لیست مقداردهی اولیه، مبادرت به تعیین تعداد عناصر آرایه می‌کند. برای مثال،

```
int n[] = { 1, 2, 3, 4, 5};
```

یک آرایه با پنج عنصر بوجود می‌آورد.

اگر سایز آرایه و لیست مقداردهی اولیه در اعلان یک آرایه مشخص شده باشند، بایستی تعداد عناصر موجود در لیست مقداردهی اولیه کمتر یا برابر با سایز آرایه باشد. برای مثال در اعلان آرایه زیر

```
int n[5] = { 32, 27, 64, 18, 95, 14};
```

با خطای کامپایل مواجه خواهد شد، چرا که لیست مقداردهی اولیه دارای شش عنصر است در حالیکه آرایه فقط پنج عنصر دارد.

خطای برنامه‌نویسی



تدارک دیدن عناصر بیشتر در لیست مقداردهی اولیه، به نسبت سایز آرایه، خطای کامپایل است.

خطای برنامه‌نویسی



فراموش کردن مقداردهی اولیه عناصر یک آرایه که باید مقداردهی اولیه شوند، یک خطای منطقی است.

```

1 // Fig. 7.4: fig07_04.cpp
2 // Initializing an array in a declaration.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include <iomanip>
8 using std::setw;
9
10 int main()
11 {
12     // use initializer list to initialize array n
13     int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
14
15     cout << "Element" << setw( 13 ) << "Value" << endl;
16
17     // output each array element's value
18     for ( int i = 0; i < 10; i++ )
19         cout << setw( 7 ) << i << setw( 13 ) << n[ i ] << endl;
20
21     return 0; // indicates successful termination
22 } // end main

```

Element	Value
0	32
1	27
2	64
3	18
4	95
5	14
6	90



7	70
8	60
9	37

شکل ۷-۴ | مقداردهی اولیه عناصر آرایه به هنگام اعلان.

تعیین سایز آرایه با متغیر ثابت و تنظیم عناصر آرایه از طریق محاسبه

در برنامه شکل ۷-۵ عناصر یک آرایه ده عنصری بنام `s` با مقدار زوج ۲,۴,۶,...,۲۰ تنظیم شده (خطوط ۱۷-۱۸) و آرایه با فرمت جدولی چاپ شده است (خطوط ۲۴-۲۰). این اعداد با ضرب هر مقدار پی در پی شمارنده حلقه در ۲ و جمع آن با ۲ تولید می‌شوند (خط ۱۸).

```

1 // Fig. 7.5: fig07_05.cpp
2 // Set array s to the even integers from 2 to 20.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include <iomanip>
8 using std::setw;
9
10 int main()
11 {
12     // constant variable can be used to specify array size
13     const int arraySize = 10; // must initialize in declaration
14
15     int s[ arraySize ]; // array s has 10 elements
16
17     for ( int i = 0; i < arraySize; i++ ) // set the values
18         s[ i ] = 2 + 2 * i;
19
20     cout << "Element" << setw( 13 ) << "Value" << endl;
21
22     // output contents of array s in tabular format
23     for ( int j = 0; j < arraySize; j++ )
24         cout << setw( 7 ) << j << setw( 13 ) << s[ j ] << endl;
25
26     return 0; // indicates successful termination
27 } // end main

```

Element	Value
0	2
1	4
2	6
3	8
4	10
5	12
6	14
7	16
8	18
9	20

شکل ۷-۵ | تولید مقداری برای وارد کردن به آرایه.

در خط ۱۳ از یک توصیف کننده `const` برای اعلان یک متغیر ثابت بنام `arraySize` با مقدار ۱۰ استفاده شده است. بایستی متغیرهای ثابت به هنگام اعلان مقداردهی شوند. پس از آن مقدار این متغیرها قابل تغییر نیست (همانند برنامه ۷-۶ و ۷-۷). به متغیرهای ثابت، ثابت‌های نامی یا متغیرهای فقط خواندنی هم گفته می‌شود.



خطای برنامه‌نویسی



در صورت تخصیص یک مقدار به یک متغیر ثابت پس از اعلان آن، با خطای کامپایل مواجه خواهید شد.

خطای برنامه‌نویسی



نتیجه تخصیص یک مقدار به یک متغیر ثابت در یک عبارت اجرایی، خطای کامپایل است. می‌توان متغیرهای ثابت را در هر کجای که یک عبارت ثابت مورد نیاز است بکار گرفت. در برنامه شکل ۷-۵ متغیر ثابت **arraySize** سایز آرایه **s** را در خط ۱۵ تصریح کرده است.

خطای برنامه‌نویسی



فقط می‌توان از ثابت‌ها برای اعلان سایز آرایه‌های اتوماتیک و استاتیک استفاده کرد. عدم استفاده از یک ثابت به این منظور، خطای کامپایل بدنیال خواهد داشت. با استفاده از متغیرهای ثابت در تعیین سایز آرایه، خوانایی برنامه افزایش پیدا می‌کند. در برنامه شکل ۷-۵ اولین ساختار **for** یک آرایه ۱۰۰ عنصری را با تغییر ساده مقدار **arraySize** در اعلان خود از ۱۰ تا ۱۰۰۰ پر می‌کند. اگر متغیر ثابت **arraySize** بکار گرفته نشده بود، مجبور بودیم تا خطوط ۱۵، ۱۷ و ۲۳ برنامه را برای کار با آرایه ۱۰۰۰ عنصری تغییر دهیم. همانطوری که برنامه‌ها بزرگتر می‌شوند، این تکنیک می‌تواند در نوشتن برنامه‌های واضح‌تر و اصلاح‌پذیرتر بکار گرفته شود.

مهندسی نرم‌افزار



تعریف سایز هر آرایه بصورت یک متغیر ثابت بجای یک ثابت لیترال می‌تواند در ایجاد برنامه‌های بسط پذیر موثر باشد.

برنامه‌نویسی ایده‌آل



تعریف سایز آرایه بصورت یک متغیر ثابت بجای یک ثابت لیترال، سبب ایجاد برنامه‌های واضح‌تر می‌شود. این تکنیک سبب حذف اعداد جادویی می‌گردد.

```

1 // Fig. 7.6: fig07_06.cpp
2 // Using a properly initialized constant variable.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 int main()
8 {
9     const int x = 7; // initialized constant variable
10    cout << "The value of constant variable x is: " << x << endl;
11    return 0; // indicates successful termination
12 } // end main
The value of constant variable x is:7

```

شکل ۷-۶ | مقداردهی اولیه و استفاده از یک متغیر ثابت.

```

1 // Fig. 7.7: fig07_07.cpp
2 // A const variable must be initialized.
3
4 int main()

```



```

5  {
6      const int x; // Error: x must be initialized
7
8      x = 7; // Error: cannot modify a const variable
9
10     return 0; // indicates successful termination
11 } // end main
BoRland C++ command-line compiler error message:
Error E2304 fig07_07.cpp 6: Constant variablr 'x' must be initialized
    in function main()
Error E2304 fig07_07.cpp 8: Cannot modify a const object in function
    main()

Microsoft Visual C++ .NET compiler error message:
C:\cpphtp5e_examples\ch07\fig07_07.cpp(6):error C2734: 'x': const object
    must be initialized if not extern
C:\cpphtp5e_examples\ch07\fig07_07.cpp(8):error C2166: l-value specifies
    const object

GNU C++ compiler error message:
fig07_07.cpp:6: error: uninitialized const 'x'
fig07_07.cpp:8: error: assignment of read-only variable 'x'

```

شکل ۷-۷ | متغیر ثابت با یستی مقداردهی اولیه شود.

با است آوردن مجموع عناصر آرایه

غالباً عناصر یک آرایه نشانده‌نده دنباله‌ای از مقادیر هستند که در محاسبات بکار گرفته می‌شوند. برای مثال، اگر عناصر یک آرایه نشانده‌نده نمرات تعدادی از دانشجویان باشد، ممکن است استاد علاقمند به دانستن میانگین نمرات کلاس این عدد از دانشجویان باشد. در این مثال از کلاس **GradeBook** در برنامه‌های شکل ۷-۱۶ و ۷-۱۷ استفاده شده است. همچنین در برنامه‌های شکل ۷-۲۳ و ۷-۲۴ از این تکنیک استفاده کرده‌ایم. در برنامه شکل ۷-۸ آرایه **a** با ده عنصر اعلان، تخصیص و مقداردهی اولیه شده است (خط 10).

```

1 // Fig. 7.8: fig07_08.cpp
2 // Compute the sum of the elements of the array.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 int main()
8 {
9     const int arraySize = 10; // constant variable indicating size of array
10    int a[ arraySize ] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
11    int total = 0;
12
13    // sum contents of array a
14    for ( int i = 0; i < arraySize; i++ )
15        total += a[ i ];
16
17    cout << "Total of array elements: " << total << endl;
18
19    return 0; // indicates successful termination
20 } // end main
Total of array elements: 849

```

شکل ۷-۸ | محاسبه مجموع عناصر موجود در یک آرایه.



خطوط 14-15 در بدنه ساختار `for`، عمل جمع را انجام می‌دهد. بصورت جایگزین می‌توان مقادیر تدارک دیده شده بعنوان مقادیر اولیه برای آرایه `a` را از طریق کاربر یا یک فایل وارد برنامه ساخت. برای کسب اطلاعات بیشتر در زمینه وارد کردن مقادیر به برنامه‌ها می‌توانید به فصل ۱۷ مراجعه کنید. برای مثال، عبارت `for`

```
for ( int j = 0; j < arraySize; j++ )
    cin >> a [ j ];
```

در هر بار یک مقدار از صفحه کلید خوانده و آنرا در عنصر `a[j]` ذخیره می‌سازد.

نمایش گرافیکی داده‌های آرایه توسط نمودارهای میله‌ای

بسیاری از برنامه‌ها داده‌های خود را با فرمت‌های گرافیکی یا بصری به اطلاع کاربران خود می‌رسانند. برای مثال، غالباً مقادیر عددی بصورت میله‌های در یک نمودار میله‌ای به نمایش در می‌آیند که میله‌های بلندتر نشانده‌نده مقادیر عددی بزرگ‌تر هستند. یکی از ساده‌ترین روش‌های نمایش گرافیکی داده‌های عددی استفاده از یک نمودار میله‌ای است که هر مقدار عددی را بصورت میله‌ای از ستاره‌ها (*) به نمایش در می‌آورد.

غالباً اساتید علاقمند به بررسی توزیع نمرات در یک امتحان یا آزمون هستند. فرض کنید که نمرات عبارتند از 87، 68، 94، 83، 100، 78، 91، 85، 76 و 87 باشند. توجه کنید که در اینجا فقط یک نمره 100، دو نمره در محدوده 90، چهار نمره در محدوده 80، دو نمودار محدوده 70 و یک نمره در محدوده 60 قرار دارد و هیچ نمره‌ای پایین‌تر از 60 وجود ندارد. در مثال بعدی (شکل ۷-۹) این نمرات در یک آرایه 11 عنصری ذخیره شده که هر یک متناظر با یک رده از نمرات می‌باشد. برای مثال، `n[0]` نشانده‌نده تعداد نمرات در محدوده 0-9، `n[7]` نشانده‌نده تعداد نمرات در محدوده 79-80 است و `n[10]` نشانده‌نده تعداد نمره‌های 100 است. دو نسخه از کلاس **GradeBook** (شکل‌های ۷-۱۶ و ۷-۱۷) و شکل‌های ۷-۲۳ و ۷-۲۴) حاوی کدی هستند که دفعات تکرار این نمرات را محاسبه می‌کنند. در این مرحله، بصورت غیراتوماتیک آرایه‌ای را با مجموعه‌ای از نمرات ایجاد می‌کنیم.

```
1 // Fig. 7.9: fig07_09.cpp
2 // Bar chart printing program.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include <iomanip>
8 using std::setw;
9
10 int main()
11 {
12     const int arraySize = 11;
13     int n[ arraySize ] = { 0, 0, 0, 0, 0, 0, 1, 2, 4, 2, 1 };
14
15     cout << "Grade distribution:" << endl;
```



```

16 // for each element of array n, output a bar of the chart
17 for ( int i = 0; i < arraySize; i++ )
18 {
19     // output bar labels ("0-9:", ... , "90-99:", "100:")
20     if ( i == 0 )
21         cout << " 0-9: ";
22     else if ( i == 10 )
23         cout << " 100: ";
24     else
25         cout << i * 10 << "-" << ( i * 10 ) + 9 << ": ";
26
27     // print bar of asterisks
28     for ( int stars = 0; stars < n[ i ]; stars++ )
29         cout << '*';
30
31     cout << endl; // start a new line of output
32 } // end outer for
33
34
35     return 0; // indicates successful termination
36 } // end main
Grade distribution :
0-9:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
100: *

```

شکل ۷-۹ | برنامه چاپ نمودار میله‌ای.

برنامه اعداد را از آرایه خوانده و اطلاعات را بصورت نمودار میله‌ای به نمایش در می‌آورد. برنامه طول هر گراف را با میله‌ای از ستاره که نشانده‌نده تعداد نمرات در آن محدوده هستند، نشان می‌دهد. برای قرار دادن یک برچسب (عنوان) برای هر میله، خطوط 26-21 محدوده هر نمره را (مثلاً "70-79") برحسب مقدار جاری متغیر شمارنده **i** چاپ می‌کنند. عبارت **for** تودرتو (خطوط 29-30) میله‌ها را چاپ می‌کند. به شرط تکرار حلقه در خط 29 دقت کنید (**stars<=n[i]**). هر بار که برنامه به **for** داخلی می‌رسد، شمارش حلقه از 0 تا **n[i]** صورت می‌گیرد، از این‌رو با استفاده از این مقدار در آرایه **n** تعداد ستاره‌های که باید به نمایش درآیند، مشخص می‌شود. در این مثال، حاصل **n[0]-n[5]** صفر است، چرا که هیچ دانشجویی نمره‌ای کمتر از 60 دریافت نکرده است. بنابر این برنامه در کنار شش محدوده امتیاز اول هیچ ستاره‌ای به نمایش در نیاورده است.

خطای برنامه‌نویسی

اگرچه امکان استفاده از یک متغیر کنترلی یکسان در یک عبارت **for** و عبارت **for** دوم قرار گرفته در درون اولی وجود دارد، اما بدلیل اجتناب از رخداد خطاهای منطقی از آن اجتناب کردۀ‌ایم.





گاهی اوقات، برنامه‌ها از متغیرهای شمارنده برای تحلیل داده‌ها استفاده می‌کنند. در برنامه ۶-۹ از شمارنده‌های مختلف در برنامه پرتاب طاس برای ردگیری تعداد رخ داده‌ای هر وجه طاس استفاده کردیم. نسخه آرایه‌ای این برنامه در شکل ۷-۱۰ آورده شده است.

در برنامه ۱۰-۷ از آرایه **frequency** برای شمارش رخ داده‌ای هر وجه طاس استفاده شده است (خط ۲۰). یک عبارت در خط ۲۶ این برنامه جایگزین ساختار **switch** در خطوط ۳۰-۵۲ از برنامه ۶-۹ شده است. در خط ۲۶ از یک مقدار تصادفی برای تعیین اینکه کدام عنصر **frequency** در زمان تکرار هر حلقه افزایش یافته، استفاده شده است. محاسبه بکار رفته در خط ۲۶ یک شاخص تصادفی از ۱ تا ۶ ایجاد می‌کند، از این‌رو آرایه **frequency** بایستی بقدر کافی برای نگهداری شش شمارنده بزرگ باشد. با این همه، ما از یک آرایه هفت عنصری استفاده کرده‌ایم تا **frequency[0]** را در نظر نگیریم. در اینحالت بسیار منطقی خواهد بود که برای وجه ۱ طاس مقدار **frequency[1]** بجای **frequency[0]** افزایش یابد. بنابر این از مقدار هر وجه بعنوان شاخص آرایه **frequency** استفاده شده است. همچنین خطوط ۶۱-۵۶ از برنامه ۶-۹ را با حلقه‌ای که در میان آرایه **frequency** برای چاپ نتایج حرکت می‌کند، جایگزین کرده‌ایم (خطوط ۳۱-۳۳).

```

1 // Fig. 7.10: fig07_10.cpp
2 // Roll a six-sided die 6,000,000 times.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include <iomanip>
8 using std::setw;
9
10 #include <cstdlib>
11 using std::rand;
12 using std::srand;
13
14 #include <ctime>
15 using std::time;
16
17 int main()
18 {
19     const int arraySize = 7; // ignore element zero
20     int frequency[ arraySize ] = { 0 };
21
22     srand( time( 0 ) ); // seed random number generator.
23
24     // roll die 6,000,000 times; use die value as frequency index
25     for ( int roll = 1; roll <= 6000000; roll++ )
26         frequency[ 1 + rand() % 6 ]++;
27
28     cout << "Face" << setw( 13 ) << "Frequency" << endl;
29
30     // output each array element's value
31     for ( int face = 1; face < arraySize; face++ )
32         cout << setw( 4 ) << face << setw( 13 ) << frequency[ face ]
33         << endl;
34
35     return 0; // indicates successful termination
36 } // end main

```



Face	Frequency
1	1000167
2	1000149
3	1000152
4	998748
5	999626
6	1001158

شکل ۷-۱۰ | برنامه پرتاب طاس با استفاده از آرایه بجای switch

استفاده از آرایه‌ها برای تحلیل تابع

در مثال بخش قبلی از آرایه برای بررسی اطلاعات جمع‌آوری شده از یک امتحان استفاده شده بود حال به مسئله زیر توجه کنید:

از چهل دانشجو در مورد کیفیت غذای عرضه شده در رستوران دانشگاه سوال شده و پاسخ دانشجویان می‌تواند در محدوده ۱۰ تا ۱ قرار داشته باشد. به اینصورت که ۱ نشان‌هندۀ کیفیت بسیار پایین و ۱۰ کیفیت عالی است. پاسخ چهل دانشجو را در یک آرایه قرار داده و میزان و تعداد پاسخ‌های همسان را مشخص سازید.

این مسئله با استفاده از یک آرایه در برنامه شکل ۷-۱۰ ارائه شده است. در این برنامه علاقمند هستیم تا تعداد پاسخ‌های مطرح شده و نوع آنها را دسته‌بندی نمائیم. آرایه **responses** (خطوط ۱۹-۲۰) یک آرایه ۴۰ عنصری از نوع صحیح و حاوی پاسخ‌های دانشجویان است. دقت کنید که این آرایه بصورت **const** اعلان شده است، و از این‌رو مقادیر آن قابل تغییر نمی‌باشند (و نباید تغییر داده شوند). با استفاده از آرایه **frequency**، با ۱۱ عنصر، می‌توانیم تعداد پاسخ‌های همسان را شمارش کنیم (خط ۲۲). اولین عنصر آرایه، با **frequency[0]** را نادیده گرفته‌ایم چراکه بسیار منطقی است که پاسخ ۱ در **frequency[1]** قرار داده شود. می‌توانیم هر پاسخ را بصورت مستقیم عنوان یک شاخص بر روی آرایه **frequency** بکار گیریم.

```

1 // Fig. 7.11: fig07_11.cpp
2 // Student poll program.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include <iomanip>
8 using std::setw;
9
10 int main()
11 {
12     // define array sizes
13     const int responseSize = 40; // size of array responses
14     const int frequencySize = 11; // size of array frequency
15
16     // place survey responses in array responses
17     const int responses[ responseSize ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8,
18         10, 1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7,
19         5, 6, 6, 5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };

```



```

20 // initialize frequency counters to 0
21 int frequency[ frequencySize ] = { 0 };
22
23 // for each answer, select responses element and use that value
24 // as frequency subscript to determine element to increment
25 for ( int answer = 0; answer < responseSize; answer++ )
26     frequency[ responses[ answer ] ]++;
27
28 cout << "Rating" << setw( 17 ) << "Frequency" << endl;
29
30 // output each array element's value
31 for ( int rating = 1; rating < frequencySize; rating++ )
32     cout << setw( 6 ) << rating << setw( 17 ) << frequency[ rating ]
33     << endl;
34
35 return 0; // indicates successful termination
36 } // end main

```

Rating	Frequency
1	2
2	2
3	2
4	2
5	5
6	11
7	5
8	7
9	1
10	3

شکل ۷-۱۱ | برنامه تحلیل پاسخ دانشجویان.

کارائی



گاهی اوقات توجه به کارائی سبب کاهش توجه به وضوح برنامه می‌شود.

حلقه **for** (خطوط 27-26) یک به یک پاسخ‌ها را از آرایه **responses** خوانده و یک واحد به ده شمارنده در آرایه **frequency** اضافه می‌کند (از **frequency[1]** تا **frequency[10]**). عبارت کلیدی در خط 27 حلقه متبلور است، این عبارت به شمارنده مقتضی **frequency** که توسط مقدار **responses[answer]** تعیین می‌شود، یک واحد اضافه می‌کند.

اجازه دهید تا به بررسی چند تکرار ساختار **for** پردازیم. هنگامی شمارنده **answer** برابر **0** است، عبارت **responses[0]** مقدار **responses[0]** را خواهد داشت (مقدار 1 در خط 17). از اینرو، در واقع عبارت **frequency[1]++** بصورت **frequency[responses[answer]]++** تفسیر خواهد شد، به این معنی که اولین شمارنده در آرایه **frequency** یک واحد افزایش می‌یابد. در ارزیابی عبارت کار با ارزیابی مقدار داخلی ترین براکت‌های آغاز می‌شود. مقدار **answer** وارد عبارت شده و به ارزیابی براکت‌های بعدی پرداخته می‌شود (**responses[answer]**)، که مقدار مورد استفاده بعنوان شاخص برای آرایه **frequency** است و تعیین می‌کند کدام شمارنده باید افزایش یابد (در این مورد شمارنده **1**).



هنگامی که **answer** برابر ۱ است، عبارت **responses[answer]** مقدار دومین عنصر را خواهد داشت (مقدار ۲). در نتیجه، عبارت

```
frequency[responses[answer]]++
```

بصورت **frequency[2]++** تفسیر شده و موجب می‌شود تا عنصر ۲ آرایه (سومین عنصر در آرایه) افزایش یابد. زمانیکه **answer** برابر ۲ است، عبارت **frequency[answer]** (مقدار ۶) را خواهد داشت، از اینرو

```
frequency[responses[answer]]++
```

بصورت **frequency[6]++** تفسیر شده و موجب می‌شود تا عنصر ۶ آرایه (هفتمین عنصر در آرایه) افزایش یابد. دقت کنید که علیرغم تعداد پاسخ‌های مطرح شده، فقط به ۱۱ عنصر آرایه برای تحلیل نتایج نیاز است چرا که تمام پاسخ‌ها در محدوده مقادیر ۱۰ تا ۱ قرار دارند و مقادیر شاخص برای ۱۱ عنصر آرایه ۱۱ تا ۰ هستند.

اگر داده‌ای حاوی مقدار خارج از محدوده نظیر ۱۳ باشد، برنامه مبادرت به افزودن ۱ به **[13]** خواهد کرد، که در اینحالت از مزه‌های آرایه خارج خواهد شد. در زبان C++, چنین مراجعه‌ای توسط کامپایلر و در زمان اجرا مجاز شناخته می‌شود. در چنین وضعیتی برنامه از مز آرایه عبور کرده و داده را در مکانی از حافظه ذخیره می‌کند، این عمل می‌تواند در مقدار متغیر دیگری در برنامه تغییر ایجاد کند و در پاسخ برنامه اشکال بوجود آورد.

خطای برنامه‌نویسی

مراجعه به یک عنصر خارج از مزه‌های یک آرایه، خطای زمان اجرا بدنیال خواهد داشت.



اجتناب از خطای



زمانیکه حلقه‌ای در درون یک آرایه اجرا می‌شود، باید شاخص آرایه در بین صفر و مز بالایی آرایه بماند. مقادیر اولیه و پایانی بکار رفته در ساختار تکرار باید از دسترسی به عناصری خارج از مزه‌های آرایه اجتناب کنند. یک زبان بسط پذیر است. در بخش ۷-۱۱ به معرفی کلاس **vector** (بردار) خواهیم پرداخت که به برنامه نویسان امکان انجام فرآیندهای را می‌دهد که انجام آنها در آرایه‌های توکار C++ وجود ندارند. برای مثال، می‌توانیم مستقیماً به مقایسه بردارها پرداخته و یک بردار را به بردار دیگری تخصیص دهیم. در فصل ۱۱ با نحوه پیاده سازی آرایه‌ها بصورت کلاس‌های تعریف شده توسط کاربر آشنا خواهید شد. این تعریف جدید از آرایه امکان می‌دهد تا کل آرایه را با دستوارت **cin** و **cout** وارد و خارج کرده و آرایه‌ها را به هنگام ایجاد مقداردهی اولیه کرده از دسترسی به خارج از محدوده عناصر آرایه اجتناب کرده و



شاخص‌ها را تغییر داد (حتی نوع شاخص‌ها). از این‌رو نیازی نیست که اولین عنصر آرایه، عنصر صفر باشد. حتی می‌توانیم از شاخص‌های غیر صحیح استفاده کنیم.

اجتناب از خطأ



در فصل ۱۱، بانحوه ایجاد کلاس‌های که نشانده‌نده آرایه‌های هوشمند هستند آشنا خواهید شد، که در زمان اجرا مبادرت به بررسی مزدی‌های آرایه می‌کنند. با استفاده از چنین آرایه‌های می‌توان جلوی برخی از خطاهای را گرفت.

استفاده از آرایه‌های کاراکتری برای ذخیره‌سازی و کنترل رشته‌ها

تا بدین مرحله، فقط در مورد آرایه‌های صحیح صحبت کردیم. با این همه، امکان دارد آرایه‌های از نوع‌های مختلف داشته باشیم. در این بخش به معرفی نحوه ذخیره‌سازی رشته‌های کاراکتری در آرایه‌های کاراکتری می‌پردازیم. بخاطر دارید که، در ابتدای فصل سوم، از شی‌های **string** برای ذخیره سازی رشته‌های کاراکتری همانند نام دوره در کلاس **GradeBook** استفاده کردیم. رشته‌ای همانند "hello" در واقع یک آرایه کاراکتری است. در حالیکه شی‌های **string** برای کاهش خطای مناسب هستند، آرایه‌های کاراکتری که نشانده‌نده رشته‌ها می‌باشند دارای ویژگی‌های منحصر بفردی هستند که در این بخش با آنها آشنا خواهید شد. همانطوری که به یادگیری **C++** ادامه می‌دهید، با قابلیت‌های **C++** مواجه خواهید شد که استفاده از آرایه‌های کاراکتری را لازم می‌کنند. همچنین امکان دارد کدهای موجود را برای استفاده از آرایه‌های کاراکتری به روز کنید.

یک آرایه کاراکتری را می‌توان با استفاده از یک رشته لیترال مقداردهی اولیه کرد. برای مثال، اعلان

```
char string1[] = "first";
```

عناصر آرایه **string1** را با کاراکترهای جداگانه در رشته لیترال "first" مقداردهی می‌کند. سایز آرایه **string1** در اعلان فوق توسط کامپایلر و برپایه طول رشته تعیین می‌شود. توجه به این نکته مهم است که رشته "first" حاوی پنج کاراکتر به همراه یک کاراکتر پایان دهنده رشته بنام کاراکتر **null** است. بنابر این آرایه **string1** حاوی شش عنصر می‌باشد. ثابت کاراکتری نشانده‌نده کاراکتر **null** رشته '0' است (یک خط مورب و بدنبال آن صفر). تمام رشته‌های عرضه شده توسط آرایه‌های کاراکتری با این کاراکتر خاتمه می‌یابند. یک آرایه کاراکتری که عرضه کننده رشته است بایستی به میزان کافی بزرگ اعلان شده باشد تا بتواند کاراکترهای موجود در رشته را به همراه کاراکتر **null** نگهداری کند.

همچنین آرایه‌های کاراکتری را می‌توان با ثابت‌های کاراکتری مجزا از هم در یک لیست مقداردهی اولیه، مقداردهی کرد. اعلان زیر معادل با اعلان فوق است.

```
char string1[] = {'0', 't', 's', 'r', 'z', 'f'};
```



آرایه‌ها و بردارها

به گوتشن‌های قرار گرفته در اطراف هر ثابت کاراکتری توجه کنید. همچنین توجه کنید که بصورت صریح کاراکتر **null** در لیست مقداردهی اولیه تدارک دیده شده است. بدون آن، این آرایه فقط نشانده‌نده، آرایه‌ای از کاراکترهاست، نه یک رشته. همانطوری که در فصل هشتم شاهد خواهید بود، تدارک ندیدن کاراکتر **null** برای یک رشته می‌تواند خطای منطقی بوجود آورد. بدلیل اینکه یک رشته یک آرایه کاراکتری است، می‌توانیم به کاراکترهای مجزا یک رشته و به کمک شاخص آرایه مستقیماً دسترسی پیدا کنیم. برای مثال، **string1[0]** کاراکتر 'f'، **string1[3]** کاراکتر 's' و **string1[5]** کاراکتر **null** است.

همچنین می‌توانیم یک رشته را مستقیماً به یک آرایه کاراکتری از طریق صفحه کلید و با استفاده از دستور **cin** و **>>** وارد کنیم. برای مثال، اعلان

```
char string2[20];
```

یک آرایه کاراکتری با ظرفیت نگهداری 19 کاراکتر و یک کاراکتر **null** ایجاد می‌کند، عبارت

```
cin >> string2;
```

رشته‌ای از صفحه کلید بدرون **string2** خوانده و کاراکتر **null** را به انتهای رشته وارد شده توسط کاربر الصاق می‌کند. توجه کنید که در عبارت فوق فقط نام آرایه بدون هیچ گونه اطلاعاتی در مورد سایز آرایه بکار گرفته شده است. این وظیفه برنامه‌نویس است تا مطمئن شود که آرایه مورد نظر قادر به نگهداری رشته تایپ شده از سوی کاربر است. بطور پیش‌فرض، **cin** کاراکترها را از صفحه کلید تا رسیدن به اولین کاراکتر **white-space** می‌خواند (صرف نظر از سایز آرایه). از این‌رو، وارد کردن داده با **cin** و **>>** می‌تواند داده را بیش از مرز فوچانی آرایه وارد سازد (بخش ۱۳-۸ در این ارتباط است).

خطای برنامه‌نویسی

تدارک ندیدن **>> cin** به همراه یک آرایه که بمیزان کافی برای ذخیره سازی رشته تایپ شده توسط صفحه کلید بزرگ نباشد، می‌تواند سبب از دست رفتن داده‌ها در برنامه شده و خطاهای جدی در زمان اجرا بوجود آورد.

یک آرایه کاراکتری با یک رشته خاتمه یافته با **null** می‌تواند با دستور **cout** و **<<** چاپ شود. عبارت

```
cout << string2;
```

آرایه **string2** را چاپ می‌کند. توجه کنید که **<< cout** همانند **cin >>** توجیهی به میزان بزرگی آرایه کاراکتری ندارد. کاراکترهای رشته تا رسیدن به کاراکتر **null** به خروجی ارسال می‌شوند.

برنامه شکل ۷-۱۲ به توصیف نحوه مقداردهی اولیه یک آرایه کاراکتری با یک رشته لیترال، خواندن یک رشته به یک آرایه کاراکتری، چاپ آرایه کاراکتری بصورت یک رشته و دسترسی به کاراکترهای جداگانه رشته پرداخته است.

```
1 // Fig. 7.12: fig07_12.cpp
2 // Treating character arrays as strings.
3 #include <iostream>
```



```

4  using std::cout;
5  using std::cin;
6  using std::endl;
7
8 int main()
9 {
10    char string1[ 20 ]; // reserves 20 characters
11    char string2[] = "string literal"; // reserves 15 characters
12
13    // read string from user into array string1
14    cout << "Enter the string \"hello there\": ";
15    cin >> string1; // reads "hello" [space terminates input]
16
17    // output strings
18    cout << "string1 is: " << string1 << "\nstring2 is: " << string2;
19
20    cout << "\nstring1 with spaces between characters is:\n";
21
22    // output characters until null character is reached
23    for ( int i = 0; string1[ i ] != '\0'; i++ )
24        cout << string1[ i ] << ' ';
25
26    cin >> string1; // reads "there"
27    cout << "\nstring1 is: " << string1 << endl;
28
29    return 0; // indicates successful termination
30 } // end main

```

```

Enter the string "hello there" : hello there
string1 is: hello
string2 is: string literal
String1 with spaces between character is:
h e l l o
string1 is: there

```

شکل ۱۲-۷-آرایه‌های کاراکتری برای پردازش رشته‌ها.

در خطوط 24-23 از شکل ۱۲-۷ از یک عبارت **for** برای ایجاد حلقه در میان آرایه **string1** و چاپ کاراکترهای مجزا شده توسط فاصله‌ها استفاده شده است. شرط موجود در عبارت **for** شرط **string1[i] != '\0'** تا مواجه شدن با کاراکتر **null** در رشته برقرار خواهد بود.

آرایه‌های محلی استاتیک و اتوماتیک

در فصل ششم در ارتباط با کلاس ذخیره‌سازی **static** بحث کردیم. یک متغیر محلی استاتیک در تعریف یک تابع، در مدت زمان اجرای برنامه وجود خواهد داشت و در بدنه همان تابع قابل استفاده و رویت است.

کارائی

می‌توانیم از **static** در اعلان یک آرایه محلی استفاده کنیم، در اینصورت دیگر آرایه در هر بار فراخوانی تابع توسط برنامه، ایجاد و مقداردهی نشده و در هر بار با خاتمه یافتن تابع، آرایه از بین نمی‌رود. اینکار می‌تواند سبب افزایش کارایی شود، بویژه به هنگام کار با آرایه‌های بزرگ.



برنامه‌ها، آرایه محلی استاتیک را به هنگام اعلان آنها و اولین بار که با آنها برخورد می‌کنند، مقداردهی اولیه می‌نمایند. اگر یک آرایه استاتیک بصورت صریح توسط برنامه‌نویس مقداردهی اولیه نشود، هر



عنصر آن آرایه با صفر و توسط کامپایلر در زمان ایجاد آرایه مقداردهی خواهد شد. بخاطر داشته باشید C++ چنین مقداردهی اولیه‌ای را برای متغیرهای اتوماتیک انجام نمی‌دهد.

برنامه شکل ۷-۱۳ به توصیف تابع **staticArrayInit** (خطوط ۴۱-۲۵) با یک آرایه استاتیک محلی (خط ۴۷) و تابع **automaticArrayInit** (خطوط ۶۰-۴۴) با یک آرایه محلی (خط ۴۷) پرداخته است.

```
1 // Fig. 7.13: fig07_13.cpp
2 // Static arrays are initialized to zero.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 void staticArrayInit( void ); // function prototype
8 void automaticArrayInit( void ); // function prototype
9
10 int main()
11 {
12     cout << "First call to each function:\n";
13     staticArrayInit();
14     automaticArrayInit();
15
16     cout << "\n\nSecond call to each function:\n";
17     staticArrayInit();
18     automaticArrayInit();
19     cout << endl;
20
21     return 0; // indicates successful termination
22 } // end main
23
24 // function to demonstrate a static local array
25 void staticArrayInit( void )
26 {
27     // initializes elements to 0 first time function is called
28     static int array1[ 3 ]; // static local array
29
30     cout << "\nValues on entering staticArrayInit:\n";
31
32     // output contents of array1
33     for ( int i = 0; i < 3; i++ )
34         cout << "array1[" << i << "] = " << array1[ i ] << " ";
35
36     cout << "\nValues on exiting staticArrayInit:\n";
37
38     // modify and output contents of array1
39     for ( int j = 0; j < 3; j++ )
40         cout << "array1[" << j << "] = " << ( array1[ j ] += 5 ) << " ";
41 } // end function staticArrayInit
42
43 // function to demonstrate an automatic local array
44 void automaticArrayInit( void )
45 {
46     // initializes elements each time function is called
47     int array2[ 3 ] = { 1, 2, 3 }; // automatic local array
48
49     cout << "\n\nValues on entering automaticArrayInit:\n";
50
51     // output contents of array2
52     for ( int i = 0; i < 3; i++ )
53         cout << "array2[" << i << "] = " << array2[ i ] << " ";
54
55     cout << "\nValues on exiting automaticArrayInit:\n";
56
57     // modify and output contents of array2
58     for ( int j = 0; j < 3; j++ )
59         cout << "array2[" << j << "] = " << ( array2[ j ] += 5 ) << " ";
60 } // end function automaticArrayInit
```



```

First call to each function:
Values on entering staticArrayInit:
array1[0] = 0 array1[1] = 0 array1[2] = 0
Values on exiting staticArrayInit:
array1[0] = 5 array1[1] = 5 array1[2] = 5

Values on entering automaticArrayInit:
array2[0] = 1 array2[1] = 2 array2[2] = 3
Values on exiting automaticArrayInit:
array2[0] = 6 array2[1] = 7 array2[2] = 8

Second call to each function:

Values on entering staticArrayInit:
array1[0] = 5 array1[1] = 5 array1[2] = 5
Values on exiting staticArrayInit:
array1[0] = 10 array1[1] = 10 array1[2] = 10

Values on entering automaticArrayInit:
array2[0] = 1 array2[1] = 2 array2[2] = 3
Values on exiting automaticArrayInit:
array2[0] = 6 array2[1] = 7 array2[2] = 8

```

شکل ۷-۱۳ | مقداردهی اولیه آرایه‌های استاتیک و اتوماتیک.

تابع **staticArrayInit** دو بار فراخوانی شده است (خطوط ۱۳ و ۱۷). آرایه محلی استاتیک توسط کامپایلر در اولین فراخوانی تابع با صفر مقداردهی شده است. تابع، آرایه را چاپ کرده و به عنصر مقدار ۵ را افزوده و مجدداً آرایه را چاپ می‌کند. بار دوم تابع فراخوانی می‌شود، آرایه استاتیک حاوی مقادیر تغییر یافته و ذخیره شده از اولین فراخوانی تابع است. تابع **automaticArrayInit** هم دو بار فراخوانی شده است (خطوط ۱۹ و ۲۱). عناصر آرایه محلی اتوماتیک با مقادیر ۱، ۲ و ۳ مقداردهی اولیه می‌شوند (خط ۴۷). تابع آرایه را چاپ کرده، مقدار ۵ به هر عنصر افزوده و آرایه را مجدداً چاپ می‌کند. بار دوم تابع فراخوانی می‌شود و عناصر آرایه مجدداً با ۱، ۲ و ۳ مقداردهی اولیه می‌شوند. آرایه دارای کلاس ذخیره سازی اتوماتیک است، از این‌رو در هر بار فراخوانی **automaticArrayInit** آرایه مجدداً ایجاد می‌شود.

خطای برنامه‌نویسی

فرض اینکه عناصر یک تابع با آرایه استاتیک محلی در هر بار فراخوانی تابع مقداردهی اولیه خواهند شد، می‌تواند شما را به سمت خطاهای منطقی در برنامه هدایت کند.

۷-۵ ارسال آرایه به توابع

برای ارسال آرایه بعنوان یک آرگومان به یک تابع، باید نام آرایه بدون استفاده از براکت‌ها مشخص شود. برای مثال، اگر آرایه **hourlyTemperatures** بصورت زیر اعلان شده باشد:

```
int hourlyTemperatures[ 24 ];
```

در فراخوانی تابع



```
modifyArray( hourlyTemperatures );
```

آرایه **hourlyTemperatures** به تابع **modifyArray** ارسال می‌شود. زمانیکه آرایه‌ای به یک تابع ارسال می‌شود، سایز آرایه هم به همراه آن ارسال می‌شود، از اینرو تابع قادر به پردازش تعداد مشخص از عناصر در آرایه خواهد بود. در بخش ۷-۱۱ به هنگام معرفی کلاس **vector** شاهد خواهید بود که سایز یک **vector** حالت توکار داشته و هر شی از سایز خود مطلع است. بنابر این به هنگام ارسال یک شی **vector** به یک تابع، نیازی نیست تا سایز **vector** بعنوان یک آرگومان ارسال شود. ارسال آرایه‌ها به توابع در C++ بصورت مراجعه صورت می‌گیرد. تابع فراخوانی شده قادر به تغییر مقدار عناصر در آرایه اصلی خواهد بود.

مقدار نام آرایه آدرس حافظه‌ای در کامپیوتر است که نشان‌دهنده اولین عنصر آرایه می‌باشد. بدلیل اینکه آدرس شروع آرایه ارسال می‌شود، تابع فراخوانی شده بطور دقیق از مکان ذخیره شده آرایه در حافظه مطلع است. بنابر این، زمانیکه تابع فراخوانی شده مبادرت به تغییر عناصر آرایه در درون بدن خود می‌کند، این تغییرات بر روی عناصر واقعی آرایه در مکان‌های اصلی آن در حافظه اعمال می‌شوند.

اگر چه کل آرایه بصورت مراجعه ارسال می‌شود، اما می‌توان عناصر جداگانه آرایه را به همان روش ساده‌ای که متغیرها ارسال می‌شوند، ارسال کرد. در آرایه‌ای که دارای عناصری از نوع داده اصلی همانند **int** است، می‌توان از روش ارسال با مقدار استفاده کرد و این امر بستگی به تعریف تابع دارد. به چنین واحدهای منفرد داده‌ای گاهای موجودیت‌های **scalar** می‌گویند. برای ارسال یک عنصر آرایه به یک تابع، از نام شاخص عنصر آرایه بعنوان یک آرگومان در فراخوانی تابع استفاده می‌شود.

در تابعی که یک آرایه را از طریق فراخوانی تابع دریافت می‌کند، باید لیست پارامتری آن برای دریافت آرایه آماده شده باشد. برای مثال، سرآیند تابع **modifyArray** می‌تواند بصورت زیر نوشته شده باشد

```
void modifyArray( int[] b, int arraySize )
```

این اعلان نشان می‌دهد که **modifyArray** در انتظار دریافت یک آرایه از نوع صحیح برای پارامتر **b** و تعداد عناصر آرایه در پارامتر **arraySize** است. نیازی به قرار دادن سایز آرایه در میان براکت‌ها نیست. اگر چنین کاری انجام دهید توسط کامپایلر نادیده گرفته خواهد شد. چراکه C++ آرایه‌ها را به روش مراجعه به توابع ارسال می‌کند. زمانیکه تابع فراخوانی شده از آرایه پارامتری بنام **b** استفاده می‌کند، در واقع به سایز واقعی آن در فراخوان مراجعه می‌نماید.

به ظاهر عجیب نمونه اولیه تابع **modifyArray** توجه کنید

```
void modifyArray( int [], int );
```



این نمونه اولیه را می‌توان بصورت زیر هم نوشت

```
void modifyArray ( int anyArrayName[], int anyVariableName );
اما همانطوری که در فصل سوم آموختیم، کامپایلرهای C++ اسامی متغیرها در نمونه‌های اولیه (prototypes) را درنظر نمی‌گیرند. بخاطر دارید که، نمونه اولیه به کامپایلر، تعداد آرگومان‌ها و نوع هر آرگومان را اعلان می‌کند (البته ترتیب آنها را هم نشان می‌دهد).
```

برنامه شکل ۷-۱۴ به توصیف تفاوت موجود مابین ارسال کل آرایه و عنصری از یک آرایه پرداخته است. خطوط ۲۳-۲۲ پنج عنصر اصلی از آرایه صحیح بنام **a** را چاپ می‌کنند. خط ۲۸ آرایه **a** و سایز آنرا به تابع **modifyArray** ارسال می‌کند (خطوط ۴۵-۵۰) که هر عنصر **a** را در ۲ ضرب می‌کند (از طریق پارامتر **b**). سپس، خطوط ۳۲-۳۳ آرایه **a** را مجدداً در **main** چاپ می‌کنند. همانطوری که در خروجی برنامه دیده می‌شود، عناصر **a** براستی توسط **modifyArray** تغییر یافته‌اند. سپس، خط ۳۶ مقدار عددی **a[3]** را چاپ کرده، سپس خط ۳۸ عنصر **a[3]** را به تابع **modifyElement** ارسال می‌کند (خطوط ۵۴-۵۸)، که آن پارامتر در ۲ ضرب شده و مقدار جدید چاپ می‌شود. توجه کنید که به هنگام چاپ **a[3]** توسط خط ۳۹ در **main**، مقدار تغییر نیافته است، چرا که عناصر جداگانه آرایه به روش مقدار ارسال می‌شوند.

```
1 // Fig. 7.14: fig07_14.cpp
2 // Passing arrays and individual array elements to functions.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include <iomanip>
8 using std::setw;
9
10 void modifyArray( int [], int ); // appears strange
11 void modifyElement( int );
12
13 int main()
14 {
15     const int arraySize = 5; // size of array a
16     int a[ arraySize ] = { 0, 1, 2, 3, 4 }; // initialize array a
17
18     cout << "Effects of passing entire array by reference:"
19         << "\n\nThe values of the original array are:\n";
20
21     // output original array elements
22     for ( int i = 0; i < arraySize; i++ )
23         cout << setw( 3 ) << a[ i ];
24
25     cout << endl;
26
27     // pass array a to modifyArray by reference
28     modifyArray( a, arraySize );
29     cout << "The values of the modified array are:\n";
30
31     // output modified array elements
32     for ( int j = 0; j < arraySize; j++ )
33         cout << setw( 3 ) << a[ j ];
34
35     cout << "\n\nEffects of passing array element by value:"
36         << "\n\na[3] before modifyElement: " << a[ 3 ] << endl;
37
```



```
38     modifyElement( a[ 3 ] ); // pass array element a[ 3 ] by value
39     cout << "a[3] after modifyElement: " << a[ 3 ] << endl;
40
41     return 0; // indicates successful termination
42 } // end main
43
44 // in function modifyArray, "b" points to the original array "a" in memory
45 void modifyArray( int b[], int sizeOfArray )
46 {
47     // multiply each array element by 2
48     for ( int k = 0; k < sizeOfArray; k++ )
49         b[ k ] *= 2;
50 } // end function modifyArray
51
52 // in function modifyElement, "e" is a local copy of
53 // array element a[ 3 ] passed from main
54 void modifyElement( int e )
55 {
56     // multiply parameter by 2
57     cout << "Value of element in modifyElement: " << ( e *= 2 ) << endl;
58 } // end function modifyElement
Effects of passing entire array by reference:
The values of the original array are:
0 1 2 3 4
The values of the modified array are:
0 2 4 6 8

Effects of passing array element by value:
a[3] before modifyElement: 6
value of element in modifyElement: 12
a[3] after modifyElement: 6
```

شکل ۷-۱۴ | ارسال کل آرایه و عناصر جداگانه آرایه به توابع.

احتمال دارد شرایطی در برنامه پیش آید که نبایستی تابعی اجازه تغییر در عناصر آرایه را داشته باشد. نوع **const** توسط C++ تدارک دیده شده که می‌تواند برای جلوگیری از تغییر مقادیر آرایه در کد تابع فراخوانی شده، بکار گرفته شود. زمانیکه تابعی یک آرایه را به همراه **const** مشخص می‌کند، عناصر آرایه در بدن تابع تبدیل به ثابت شده و هرگونه اقدام به تغییر عناصر آرایه در بدن تابع، خطای کامپایل بدنیال خواهد داشت. اینکار به برنامه‌نویسان کمک می‌کند تا جلوی تغییرات ناخواسته در عناصر آرایه را در بدن توابع بگیرند.

برنامه شکل ۷-۱۵ به بررسی عملکرد توصیف کننده **const** پرداخته است. تابع **tryToModifyArray** در خطوط 21-26 به همراه پارامتر **const int b[]** تعریف شده است که نشان می‌دهد آرایه **b** ثابت بوده و نمی‌تواند تغییر داده شود. نتیجه هر سه بار اقدام تابع به تغییر عناصر آرایه **b** (خطوط 23-25) خطای کامپایل است. برای مثال کامپایلر Microsoft Visual C++.NET خطای "l-value specifies const object." را تولید می‌کند. این پیغام خطای براین نکته دلالت دارد که استفاده از یک شی **const** (برای مثال، **b[0]**) بعنوان یک **lvalue** (مقدار سمت چپ) خطای است. نمی‌توانید یک مقدار جدید به یک شی **const** با قرار دادن آن در سمت چپ عملگر تخصیص، تخصیص دهید. توجه کنید که پیغام‌های خطای در میان کامپایلرها متفاوت



از هم است (همانند پیغام‌های به نمایش درآمده در برنامه شکل ۱۵-۷). در فصل ۱۰ مجدداً در ارتباط با توصیف کننده **const** صحبت خواهیم کرد.

خطای برنامه‌نویسی



نتیجه فراموش کردن این نکته که آرایه‌ها به صورت مراجعه ارسال می‌شوند و می‌توانند توسط توابع فراخوانی شده تغییر داده شوند، خطای منطقی بدنیال خواهد داشت.

مهندسی نرم‌افزار



اعمال نوع **const** در کنار یک پارامتر آرایه در تعریف یک تابع برای اجتناب از تغییر عناصر آرایه در بدنیه تابع فراخوانی شده، نمونه دیگری از قاعده و اگذاری حداقل امتیاز است. توابع نبایستی قادر به تغییر دادن آرایه باشند، مگر اینکه به اینکار واقعاً نیاز باشد.

```

1 // Fig. 7.15: fig07_15.cpp
2 // Demonstrating the const type qualifier.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 void tryToModifyArray( const int [] ); // function prototype
8
9 int main()
10 {
11     int a[] = { 10, 20, 30 };
12
13     tryToModifyArray( a );
14     cout << a[ 0 ] << ' ' << a[ 1 ] << ' ' << a[ 2 ] << '\n';
15
16     return 0; // indicates successful termination
17 } // end main
18
19 // In function tryToModifyArray, "b" cannot be used
20 // to modify the original array "a" in main.
21 void tryToModifyArray( const int b[] )
22 {
23     b[ 0 ] /= 2; // error
24     b[ 1 ] /= 2; // error
25     b[ 2 ] /= 2; // error
26 } // end function tryToModifyArray

```

Борланд C++ command-line compiler error message:

```

Error E2024 fig07_15.cpp 23: Cannot modify a const object
    in function tryToModifyArray(const int * const)
Error E2024 fig07_15.cpp 24: Cannot modify a const object
    in function tryToModifyArray(const int * const)
Error E2024 fig07_15.cpp 25: Cannot modify a const object
    in function tryToModifyArray(const int * const)

```

Microsoft Visual C++ .NET compiler error message:

```

C:\cpphttp5_examples\ch07\fig07_15.cpp(23) : error C2166: l-value specifies
    const object
C:\cpphttp5_examples\ch07\fig07_15.cpp(24) : error C2166: l-value specifies
    const object
C:\cpphttp5_examples\ch07\fig07_15.cpp(25) : error C2166: l-value specifies
    const object

```

GNU C++ compiler error message:

```

fig07_15.cpp:23: error: assignment of read-only location
fig07_15.cpp:24: error: assignment of read-only location
fig07_15.cpp:25: error: assignment of read-only location

```



شکل ۷-۱۵ | اعمال نوع `const` بر روی یک پارامتر آرایه.

۷-۶ بحث آموزشی: کلاس `GradeBook` با استفاده از آرایه برای ذخیره‌سازی نمرات این بخش نسخه بهبود یافته‌ای از کلاس `GradeBook` را که در فصل سوم معرفی شد و در طول فصل‌های چهارم الی ششم توسعه یافته، عرضه می‌کند. بخاطر دارید که این کلاس یک دفترچه نمره را نشان می‌دهد که نمرات را ذخیره کرده و به تحلیل آنها می‌پردازد. نسخه‌های قبلی این کلاس مبادرت به پردازش مجموعه‌ای از نمرات وارد شده توسط کاربر می‌کردند، اما قادر به نگهداری مقادیر نمرات در اعضای داده کلاس نبودند. بنابر این، تکرار محاسبات نیازمند، ورود مجدد همان نمرات توسط کاربر بود. یک راه حل برای این مشکل، ذخیره هر نمره وارد شده در یک عضو داده مجزا از کلاس است. برای مثال، می‌توانیم اعضای داده `grade1`, `grade2`, `grade10`, ..., `grade2` را در کلاس `GradeBook` برای ذخیره نمره دانشجو ایجاد کنیم. با این همه، کدی که مجموع نمرات را محاسبه و میانگین کلاس را بدست می‌آورد می‌تواند بسیار درهم شود. در این بخش، این مشکل را با ذخیره‌سازی نمرات در یک آرایه حل می‌کنیم.

ذخیره‌سازی نمرات در آرایه‌ای از کلاس `GradeBook`

نسخه کلاس `GradeBook` عرضه شده در این بخش (برنامه‌های ۷-۱۶ و ۷-۱۷) از یک آرایه صحیح برای ذخیره‌سازی نمرات چندین دانشجو در یک امتحان، استفاده می‌کند. آرایه `grades` بصورت یک عضو داده در خط ۲۹ از برنامه شکل ۷-۱۶ اعلام شده است، بنابر این هر شی `GradeBook` مجموعه نمرات متعلق بخود را نگهداری خواهد کرد.

```

1 // Fig. 7.16: GradeBook.h
2 // Definition of class GradeBook that uses an array to store test grades.
3 // Member functions are defined in GradeBook.cpp
4
5 #include <string> // program uses C++ Standard Library string class
6 using std::string;
7
8 // GradeBook class definition
9 class GradeBook
10 {
11 public:
12     // constant -- number of students who took the test
13     const static int students = 10; // note public data
14
15     // constructor initializes course name and array of grades
16     GradeBook( string, const int [] );
17
18     void setCourseName( string ); // function to set the course name
19     string getCourseName(); // function to retrieve the course name
20     void displayMessage(); // display a welcome message
21     void processGrades(); // perform various operations on the grade data
22     int getMinimum(); // find the minimum grade for the test
23     int getMaximum(); // find the maximum grade for the test
24     double getAverage(); // determine the average grade for the test
25     void outputBarChart(); // output bar chart of grade distribution
26     void outputGrades(); // output the contents of the grades array
27 private:
28     string courseName; // course name for this grade book
29     int grades[ students ]; // array of student grades

```



```
30 } ; // end class GradeBook
```

شکل ۷-۱۶ | تعریف کلاس GradeBook با استفاده از آرایه برای ذخیره نمرات.

```

1  // Fig. 7.17: GradeBook.cpp
2  // Member-function definitions for class GradeBook that
3  // uses an array to store test grades.
4  #include <iostream>
5  using std::cout;
6  using std::cin;
7  using std::endl;
8  using std::fixed;
9
10 #include <iomanip>
11 using std::setprecision;
12 using std::setw;
13
14 #include "GradeBook.h" // GradeBook class definition
15
16 // constructor initializes courseName and grades array
17 GradeBook::GradeBook( string name, const int gradesArray[] )
18 {
19     setCourseName( name ); // initialize courseName
20
21     // copy grades from gradeArray to grades data member
22     for ( int grade = 0; grade < students; grade++ )
23         grades[ grade ] = gradesArray[ grade ];
24 } // end GradeBook constructor
25
26 // function to set the course name
27 void GradeBook::setCourseName( string name )
28 {
29     courseName = name; // store the course name
30 } // end function setCourseName
31
32 // function to retrieve the course name
33 string GradeBook::getCourseName()
34 {
35     return courseName;
36 } // end function getCourseName
37
38 // display a welcome message to the GradeBook user
39 void GradeBook::displayMessage()
40 {
41     // this statement calls getCourseName to get the
42     // name of the course this GradeBook represents
43     cout << "Welcome to the grade book for\n" << getCourseName() << "!"
44     << endl;
45 } // end function displayMessage
46
47 // perform various operations on the data
48 void GradeBook::processGrades()
49 {
50     // output grades array
51     outputGrades();
52
53     // call function getAverage to calculate the average grade
54     cout << "\nClass average is " << setprecision( 2 ) << fixed <<
55     getAverage() << endl;
56
57     // call functions getMinimum and getMaximum
58     cout << "Lowest grade is " << getMinimum() << "\nHighest grade is "
59     << getMaximum() << endl;
60
61     // call function outputBarChart to print grade distribution chart
62     outputBarChart();
63 } // end function processGrades
64
65 // find minimum grade
66 int GradeBook::getMinimum()
```



```
67  {
68      int lowGrade = 100; // assume lowest grade is 100
69
70      // loop through grades array
71      for ( int grade = 0; grade < students; grade++ )
72      {
73          // if current grade lower than lowGrade, assign it to lowGrade
74          if ( grades[ grade ] < lowGrade )
75              lowGrade = grades[ grade ]; // new lowest grade
76      } // end for
77
78      return lowGrade; // return lowest grade
79  } // end function getMinimum
80
81 // find maximum grade
82 int GradeBook::getMaximum()
83 {
84     int highGrade = 0; // assume highest grade is 0
85
86     // loop through grades array
87     for ( int grade = 0; grade < students; grade++ )
88     {
89         // if current grade higher than highGrade, assign it to highGrade
90         if ( grades[ grade ] > highGrade )
91             highGrade = grades[ grade ]; // new highest grade
92     } // end for
93
94     return highGrade; // return highest grade
95 } // end function getMaximum
96
97 // determine average grade for test
98 double GradeBook::getAverage()
99 {
100     int total = 0; // initialize total
101
102     // sum grades in array
103     for ( int grade = 0; grade < students; grade++ )
104         total += grades[ grade ];
105
106     // return average of grades
107     return static_cast< double >( total ) / students;
108 } // end function getAverage
109
110 // output bar chart displaying grade distribution
111 void GradeBook::outputBarChart()
112 {
113     cout << "\nGrade distribution:" << endl;
114
115     // stores frequency of grades in each range of 10 grades
116     const int frequencySize = 11;
117     int frequency[ frequencySize ] = { 0 };
118
119     // for each grade, increment the appropriate frequency
120     for ( int grade = 0; grade < students; grade++ )
121         frequency[ grades[ grade ] / 10 ]++;
122
123     // for each grade frequency, print bar in chart
124     for ( int count = 0; count < frequencySize; count++ )
125     {
126         // output bar labels ("0-9:", ..., "90-99:", "100:")
127         if ( count == 0 )
128             cout << " 0-9: ";
129         else if ( count == 10 )
130             cout << " 100: ";
131         else
132             cout << count * 10 << "-" << ( count * 10 ) + 9 << ": ";
133
134         // print bar of asterisks
135         for ( int stars = 0; stars < frequency[ count ]; stars++ )
136             cout << '*';
```



```

137         cout << endl; // start a new line of output
138     } // end outer for
139 } // end function outputBarChart
140
141 // output the contents of the grades array
142 void GradeBook::outputGrades()
143 {
144     cout << "\nThe grades are:\n\n";
145
146     // output each student's grade
147     for ( int student = 0; student < students; student++ )
148         cout << "Student " << setw( 2 ) << student + 1 << ":" << setw( 3 )
149         << grades[ student ] << endl;
150 } // end function outputGrades

```

شکل ۷-۱۷ | توابع عضو برای کار با آرایه نمرات.

به سایز آرایه در خط 29 از شکل ۷-۱۶ توجه کنید که در آن عضو داده `students` بصورت سراسری (`public`) و `const static` مشخص شده است (اعلان شده در خط 13). این عضو داده سراسری است، از اینروست که از در دسترس سرویس گیرنده‌های کلاس قرار دارد. بزودی شاهد مثالی از یک برنامه سرویس گیرنده خواهید بود که از این ثابت استفاده می‌کند. اعلان `students` با توصیف کننده `const` نشان می‌دهد که این عضو داده ثابت است، مقدار آن پس از مقداردهی اولیه قابل تغییر نخواهد بود. کلمه کلیدی `static` در اعلان این متغیر بر این نکته دلالت دارد که عضو داده در میان تمام شی‌ها، کلاس به اشتراک گذاشته خواهد شد، تمام شی‌های `GradeBook` به تعداد دانشجویان نمره ذخیره خواهد کرد. از بخش ۳-۶ بخاطر دارید، زمانیکه هر شی از کلاس از یک صفت کپی شده خود نگهداری می‌کند، متغیری که نشانده‌نده صفت است بعنوان یک عضو داده شناخته می‌شود، هر شی (نمونه) از کلاس دارای یک کپی مجزا از متغیر در حافظه است. آنها متغیرهای برای هر شی از کلاس هستند که دارای یک کپی مجزا نیستند. اینحالت با اعضای داده `static` رخ می‌دهد که بعنوان متغیرهای کلاس هم شناخته می‌شوند. زمانیکه شی‌های یک کلاس حاوی عضوهای داده `static` ایجاد می‌شوند، تمام شی‌های آن کلاس یک کپی از عضوهای داده `static` را به اشتراک می‌گذارند. یک عضو داده `static` می‌تواند از طریق تعریف کلاس و تعریف تابع عضو همانند هر عضو داده دیگر، در دسترس قرار گیرد. همانطوری که بزودی شاهد خواهید بود یک عضو داده سراسری `static` می‌تواند از خارج از کلاس در دسترس قرار گیرد، حتی زمانیکه هیچ شی از کلاس وجود نداشته باشد. اینکار با استفاده از نام کلاس و بدنبال آن عملگر تفکیک قلمرو باینتری (::) و نام عضو داده صورت می‌گیرد. در فصل دهم با اعضای داده `static` بیشتر آشنا خواهید شد.

سازنده کلاس (اعلان شده در خط 16 از شکل ۷-۱۶ و تعریف شده در خطوط 17-24 از شکل ۷-۱۷) دارای دو پارامتر است، نام دوره و یک آرایه از نمرات. زمانیکه برنامه یک شی `GradeBook` ایجاد می‌کند (خط 13 از fig07_18.cpp)، برنامه یک آرایه از نوع `int` را به سازنده ارسال می‌کند، که مقادیر



موجود در آرایه ارسالی را به عضو `grades` کپی می‌کند (خطوط 22-23 از شکل ۷-۱۷). مقادیر نمرات در آرایه ارسالی می‌توانند از طریق کاربر یا از طریق یک فایل وارد برنامه شده باشند. در برنامه تست به نمایش درآمده، فقط یک آرایه با مجموعه‌ای از نمرات را مقداردهی کرده‌ایم (خطوط 10-11 از شکل ۷-۱۸). پس از ذخیره نمرات در عضو داده `grades` از کلاس `GradeBook`، تمام توابع عضو کلاس در صورت نیاز قادر به دسترسی به آرایه `grades` خواهند بود تا محاسبات مورد نظر را انجام دهند.

```
1 // Fig. 7.18: fig07_18.cpp
2 // Creates GradeBook object using an array of grades.
3
4 #include "GradeBook.h" // GradeBook class definition
5
6 // function main begins program execution
7 int main()
8 {
9     // array of student grades
10    int gradesArray[ GradeBook::students ] =
11        { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
12
13    GradeBook myGradeBook(
14        "CS101 Introduction to C++ Programming", gradesArray );
15    myGradeBook.displayMessage();
16    myGradeBook.processGrades();
17    return 0;
18 } // end main
```

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

```
The grades are:
```

```
Student 1: 87
Student 2: 68
Student 3: 94
Student 4: 100
Student 5: 83
Student 6: 78
Student 7: 85
Student 8: 91
Student 9: 76
Student 10: 87
```

```
Class average is 84.90
Lowest grade is 68
Highest grade is 100
```

```
Grade distribution:
```

```
0-9:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
100: *
```



شکل ۷-۱۸ | ایجاد یک شی GradeBook با استفاده از آرایه نمرات، سپس فراخوانی تابع عضو processGrades برای تحلیل نمرات.

تابع عضو **processGrade** (اعلان شده در خط 21 از شکل ۷-۱۶ و تعریف شده در خطوط 48-63 از شکل ۷-۱۷) حاوی دنباله‌ای از فراخوانی تابع عضو است که گزارشی از تحلیل نمرات چاپ می‌کند. خط 51 تابع عضو **outputGrades** را برای چاپ محتويات آرایه **grades** فراخوانی کرده است. خطوط 148-150 در تابع عضو **outputGrades** از یک عبارت **for** برای چاپ نمره هر دانشجو استفاده کرده است. اگر چه شاخص آرایه از صفر شروع می‌شود، اما مایل هستیم تعداد دانشجویان از ۱ آغاز شود. بنابر این خطوط 149-150 مقدار **student+1** را بعنوان شماره دانشجو در تولید برچسب‌های "Student 1: "، "Student 2: "، و الی آخر چاپ می‌کنند.

سپس تابع عضو **processGrade** مبادرت به فراخوانی تابع عضو **getAverage** (خطوط 54-55) برای بدست آوردن میانگین نمرات در آرایه می‌کند. تابع عضو **getAverage** (اعلان شده در خط 24 از شکل ۷-۱۶ و تعریف شده در خطوط 98-108) از یک عبارت **for** برای بدست آوردن مجموع مقادیر موجود در آرایه **grades** قبل از انجام محاسبه میانگین استفاده کرده است. توجه کنید که محاسبه میانگین بکار رفته در خط 107 از **const static** برای عضو داده **students** به منظور تعیین تعداد نمرات استفاده کرده است.

خطوط 58-59 در تابع عضو **processGrades** توابع **getMaximum** و **getMinimum** را برای تعیین پایین‌ترین و بالاترین نمرات هر دانشجو در امتحان فراخوانی می‌کنند. اجازه دهید تا به بررسی نحوه عملکرد تابع عضو **getMinimum** در یافتن کمترین نمره پیردازیم. بدلیل اینکه بالاترین نمره می‌تواند 100 باشد، فرض می‌کنیم که 100 کمترین نمره است (خط 68). سپس مبادرت به مقایسه هر عنصر آرایه با کمترین نمره می‌کنیم، تا مقادیر کمتر پیدا شوند. خطوط 71-76 در تابع **getMinimum** در میان آرایه حرکت کرده (حلقه) و خطوط 74-75 هر نمره را با مقدار **lowGrade** مقایسه می‌کند. اگر نمره کمتر از **lowGrade** باشد، **lowGrade** با آن نمره مقداردهی می‌شود. زمانیکه خط 78 اجرا شود، متغیر **lowGrade** حاوی کمترین نمره در آرایه خواهد بود. عملکرد تابع عضو **getMaximum** (خطوط 82-88) مشابه تابع عضو **getMinimum** است. سرانجام، خط 62 در تابع عضو **processGrades**، تابع عضو **outputBarChart** را برای چاپ نمودار توزیعی از نمرات را با تکنیک مشابه بکار رفته در برنامه شکل ۷-۹ فراخوانی می‌کند. در آن مثال، بصورت غیراتوماتیک تعداد نمرات در هر رده را محاسبه می‌کردیم (یعنی ۰-۹، ۱۰-۱۹، ...، ۹۰-۹۹). در این مثال، خطوط 121-120 از تکنیک مشابهی که در برنامه‌های ۱۰-۱۱ و ۷-۱۱ در محاسبه تکرار نمرات در هر رده بکار برده شده‌اند، استفاده می‌کنند. در خط 117 آرایه



با 11 عنصر از نوع صحیح برای ذخیره سازی فراوانی نمرات در هر رده از نمرات، اعلان و ایجاد شده است. برای هر نمره در آرایه **grades** خطوط 121-120 مقدار عنصر مقتضی در آرایه **frequency** را افزایش می‌دهند. برای تعیین اینکه کدام عنصر افزایش خواهد یافت، خط 121 مبادرت به تقسیم نمره (grade) جاری بر 10 می‌کند (با استفاده از تقسیم صحیح). برای مثال، اگر نمره برابر 85 باشد، خط 121 مقدار **frequency**[8] را برای به روز کردن تعداد نمرات در محدوده 89-80 افزایش می‌دهد. سپس خطوط 139-124 نمودار میله‌ای را برپایه مقدایر در آرایه **frequency** چاپ می‌کنند (به شکل ۷-۱۸ نگاه کنید). همانند خطوط 30-29 از برنامه شکل ۷-۹، خطوط 136-135 از برنامه شکل ۷-۱۷ از مقداری در آرایه **frequency** برای تعیین تعداد ستاره‌ها در نمایش هر میله استفاده می‌کنند.

تست کلاس *GradeBook*

برنامه شکل ۷-۱۸ یک شی از کلاس **GradeBook** (شکل‌های ۷-۱۶ و ۷-۱۷) را با استفاده از آرایه **gradesArray** ایجاد می‌کند (اعلان و مقداردهی شده در خطوط 10-11). توجه کنید که از عملگر تفکیک قلمرو باینری (::) در عبارت "GradeBook::students" برای دسترسی به ثابت استاتیک **students** از کلاس **GradeBook** استفاده کرده‌ایم (خط 10). از این ثابت در اینجا برای ایجاد آرایه‌ای استفاده کرده‌ایم که هم سایز آرایه **grades** ذخیره شده بعنوان یک عضو داده در کلاس **GradeBook** باشد. خطوط 14-13 نام دوره و **gradeArray** را به سازنده **GradeBook** ارسال می‌کنند. خط 15 پیغام خوش‌آمدگویی را چاپ کرده و خط 16 تابع عضو **processGrades** را فراخوانی می‌کند. خروجی برنامه نشانده‌نده تحلیل 10 نمره در **myGradeBook** است.

۷-۲ جستجوی آرایه‌ها: جستجوی خطی

غالباً، برنامه‌ها با مقدایر عظیمی از اطلاعات ذخیره شده در آرایه‌ها کار می‌کنند. در چنین مواردی تعیین اینکه آیا آرایه‌ای حاوی مقداری برابر با مقدار کلید است، ضروری می‌باشد. فرآیند مشخص کردن مکان مقدار یک عنصر خاص در آرایه، جستجو نامیده می‌شود. در این بخش، به بررسی یک تکنیک جستجو، بنام جستجوی خطی خواهیم پرداخت. در تمرین ۷-۳۳ از شما خواسته شده تا نسخه بازگشته روش جستجوی خطی را پیاده‌سازی کنید. در فصل ۲۰، به بررسی روش موثرتری بنام جستجوی باینری می‌پردازیم.



تابع **linearSearch** در برنامه ۷-۱۹، برای انجام جستجوی خطی است. تابع **linearSearch** (خطوط ۳۷-۴۴) از یک ساختار **for** حاوی یک عبارت **if** برای مقایسه هر عنصر آرایه با کلید جستجو است (خط ۴۰). اگر عناصر آرایه در حال جستجو، مرتب نباشند، تابع بطور متوسط کلید جستجو را با نیمی از عناصر آرایه مقایسه خواهد کرد. روش جستجوی خطی بر روی آرایه‌های کوچک یا آرایه‌های نامربوط بخوبی کار می‌کند. با این همه، در مورد آرایه‌های بزرگ، جستجوی خطی کارایی مناسبی ندارد. اگر آرایه مرتب شده باشد (عناصر آرایه ترتیب خاص دارند)، می‌توانید از تکنیک جستجوی بازنگری که در فصل ۲۰ با آن آشنا خواهید شد استفاده کنید.

```

1 // Fig. 7.19: fig07_19.cpp
2 // Linear search of an array.
3 #include <iostream>
4 using std::cout;
5 using std::cin;
6 using std::endl;
7
8 int linearSearch( const int [], int, int ); // prototype
9
10 int main()
11 {
12     const int arraySize = 100; // size of array a
13     int a[ arraySize ]; // create array a
14     int searchKey; // value to locate in array a
15
16     for ( int i = 0; i < arraySize; i++ )
17         a[ i ] = 2 * i; // create some data
18
19     cout << "Enter integer search key: ";
20     cin >> searchKey;
21
22     // attempt to locate searchKey in array a
23     int element = linearSearch( a, searchKey, arraySize );
24
25     // display results
26     if ( element != -1 )
27         cout << "Found value in element " << element << endl;
28     else
29         cout << "Value not found" << endl;
30
31     return 0; // indicates successful termination
32 } // end main
33
34 // compare key to every element of array until location is
35 // found or until end of array is reached; return subscript of
36 // element if key or -1 if key not found
37 int linearSearch( const int array[], int key, int sizeOfArray )
38 {
39     for ( int j = 0; j < sizeOfArray; j++ )
40         if ( array[ j ] == key ) // if found,
41             return j; // return location of key
42
43     return -1; // key not found
44 } // end function linearSearch
Enter integer search key: 36
Found value in element 18

Enter integer search key: 37
Value not found

```



شکل ۷-۱۹ | جستجوی خطی در آرایه.

۷-۸ مرتب‌سازی آرایه‌ها

مرتب‌سازی داده‌ها (ترتیب دهی داده‌ها به یک روش مشخص همانند مرتب‌سازی صعودی یا نزولی) از اعمالی است که در بیشتر برنامه‌ها صورت می‌گیرد. برای مثال، یک بانک اقدام به مرتب کردن تمام چک‌ها با شماره حساب می‌کند و از این‌رو می‌تواند صورت حساب‌های بانکی جداگانه‌ای در پایان هر ماه مهیا نماید. شرکت‌های تلفن لیست صورت حساب‌های خود را با نام خانوادگی مرتب می‌کنند تا یافتن شماره‌های تلفن آسان‌تر شود. در واقع هر سازمانی نیاز دارد تا به مرتب‌سازی داده‌های خود اقدام کند. مرتب‌سازی یکی از مسائل پیچیده در علم کامپیوتر است که تحقیقات فراوانی در این زمینه صورت گرفته است. در این بخش به بررسی یکی از ساده‌ترین طرح‌های مرتب‌سازی می‌پردازیم. در تمرینات انتهای این فصل و فصل ۲۰، به توضیح یک الگوریتم مرتب‌سازی پیچیده خواهیم پرداخت.

کارایی



گاهی اوقات، الگوریتم‌های ساده از کارایی پایینی برخوردار هستند. خاصیت چنین الگوریتم‌های در نوشتن آسان، تست و خطا‌بایی است. ممکن است الگوریتم‌های پیچیده برای استفاده در برنامه‌های که نیاز به حل‌آکثر کارایی دارند، بکار گرفته شوند.

مرتب‌سازی درجی

برنامه شکل ۷-۲۰ مقادیر ۱۰ عنصر آرایه **data** را به ترتیب صعودی مرتب می‌کند. از تکنیکی بنام مرتب‌سازی درجی استفاده می‌کنیم، این تکنیک ساده بوده اما از کارایی کافی برخوردار نیست. در اولین تکرار این الگوریتم، دومین عنصر برداشته شده و اگر کوچکتر از اولین عنصر باشد، جای آنرا با اولین عنصر عوض می‌کند (یعنی برنامه دومین عنصر را در قبل از اولین عنصر درج می‌کند). در تکرار دوم به مقایسه سومین عنصر پرداخته و آنرا در مکان یا موقعیت صحیح با توجه به دو عنصر اول قرار می‌دهد، از این‌رو هر سه عنصر مرتب شده‌اند. در تکرار^۱ این الگوریتم، اولین عناصر نام در آرایه مرتب شده خواهد بود.

خط ۱۳ از برنامه شکل ۷-۲۰ مبادرت به اعلان و مقداردهی اولیه آرایه **data** با مقادیر زیر می‌کند:

34 56 4 77 51 93 30 5 52

ابتدا برنامه به **[0]** و **[1]** نگاه می‌کند، که دارای مقادیر ۳۴ و ۵۶ هستند. در حال حاضر این دو عنصر مرتب می‌باشند، از این‌رو برنامه بکار ادame می‌دهد، اگر مقادیر مرتب نباشند، برنامه جای آنها را تعویض می‌کند.

```

1 // Fig. 7.20: fig07_20.cpp
2 // This program sorts an array's values into ascending order.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6

```



```
7 #include <iomanip>
8 using std::setw;
9
10 int main()
11 {
12     const int arraySize = 10; // size of array a
13     int data[ arraySize ] = { 34, 56, 4, 10, 77, 51, 93, 30, 5, 52 };
14     int insert; // temporary variable to hold element to insert
15
16     cout << "Unsorted array:\n";
17
18     // output original array
19     for ( int i = 0; i < arraySize; i++ )
20         cout << setw( 4 ) << data[ i ];
21
22     // insertion sort
23     // loop over the elements of the array
24     for ( int next = 1; next < arraySize; next++ )
25     {
26         insert = data[ next ]; // store the value in the current element
27
28         int moveItem = next; // initialize location to place element
29
30         // search for the location in which to put the current element
31         while ( ( moveItem > 0 ) && ( data[ moveItem - 1 ] > insert ) )
32         {
33             // shift element one slot to the right
34             data[ moveItem ] = data[ moveItem - 1 ];
35             moveItem--;
36         } // end while
37
38         data[ moveItem ] = insert; // place inserted element into the array
39     } // end for
40
41     cout << "\nSorted array:\n";
42
43     // output sorted array
44     for ( int i = 0; i < arraySize; i++ )
45         cout << setw( 4 ) << data[ i ];
46
47     cout << endl;
48     return 0; // indicates successful termination
49 } // end main
```

```
Unsorted array:
34 56 4 10 77 51 93 30 5 52
Sorted array:
4 5 10 30 34 51 52 56 77 93
```

۷-۲۰ | مرتب سازی آرایه به روش درجی.

در تکرار دوم، برنامه به [2] با مقدار 4 نگاه می‌کند. این مقدار کمتر از 56 است، از این‌رو برنامه 4 را در متغیر موقتی ذخیره کرده و 56 را یک عنصر به سمت راست حرکت می‌دهد. سپس برنامه تعیین می‌کند که 4 کمتر از 34 است و بنابر این 34 را یک عنصر به راست حرکت می‌دهد. اکنون برنامه به ابتدای آرایه رسیده است، از این‌رو 4 در data[0] قرار داده شده است. وضعیت آرایه در این مرحله بصورت زیر است:

```
4 34 56 10 77 51 93 30 5 52
```

در تکرار سوم، برنامه مقدار [3] یعنی 10 را در متغیر موقت ذخیره می‌سازد. سپس برنامه 10 را با 56 مقایسه کرده و 56 را یک عنصر به راست حرکت می‌دهد، چرا که بزرگتر از 10 است. سپس برنامه مبادرت به مقایسه 10 با 34 کرده، 34 را یک عنصر به راست حرکت می‌دهد. زمانیکه برنامه به مقایسه 10



با ۴ می‌پردازد، متوجه می‌شود که ۱۰ بزرگتر از ۴ است و ۱۰ را در **data[1]** قرار می‌دهد. اکنون آرایه بفرم زیر است:

4 10 34 56 77 51 93 30 5 52

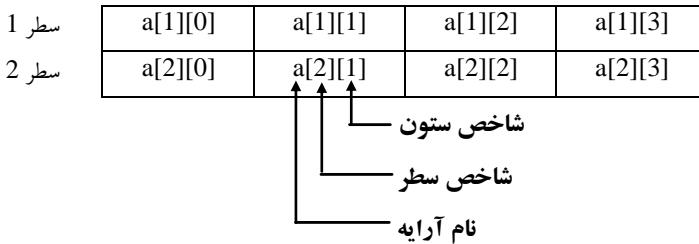
با استفاده از این الگوریتم، در تکرار **th**ⁱ، اولین عنصر نام از آرایه اصلی مرتب شده خواهد بود. عملیات مرتب سازی توسط عبارت **for** در خطوط ۲۴-۳۹ صورت می‌گیرد که در میان عناصر آرایه حرکت می‌کند. در هر تکرار خط ۲۶ بصورت موقت مقدار عنصری که در بخش مرتب شده آرایه درج خواهد شد را در متغیر موقت **insert** ذخیره می‌کند (اعلان شده در خط ۱۴). خط ۲۸ مبادرت به اعلان و مقداردهی اولیه متغیر **moveItem** می‌کند که مسیر درج عنصر را نگهداری می‌کند. حلقه خطوط ۳۱-۳۶ مسئول یافتن موقعیت صحیح عنصری است که باید درج شود. حلقه زمانی خاتمه می‌پذیرد که برنامه به انتهای آرایه برسد یا به عنصری برسد که کمتر از مقدار درج شده است. خط ۳۴ عنصر را به راست حرکت داده و خط ۳۵ موقعیت را برای درج عنصر بعدی یک واحد کاهش می‌دهد. پس از اتمام حلقه **while**، خط ۳۸ عنصر را در آن مکان درج می‌کند. زمانیکه عبارت **for** در خطوط ۲۴-۳۹ خاتمه یافته، عناصر آرایه مرتب شده خواهد بود.

اساس مرتب سازی درجی ساده بودن برنامه آن است، با این همه عملکرد کننده دارد. این سرعت کم در برخورد با آرایه‌های بزرگ‌بیشتر آشکار می‌شود. در تمرینات این فصل، تعدادی الگوریتم جایگزین برای مرتب سازی آرایه عرضه شده است. در فصل ۲۰ با تکنیک‌های کاربردی‌تر آشنا خواهید شد.

۷-۹ آرایه‌های چند بعدی

تا بدین جا با آرایه‌ای یک بعدی (یا یک شاخص‌دار) آشنا شده‌اید، که فقط حاوی یک سطر از مقادیر بودند. در این بخش، به توضیح آرایه‌های چند بعدی (یا آرایه‌های با چند شاخص) که نیاز به دو یا چند شاخص برای هویت دادن به عناصر آرایه دارند، خواهیم پرداخت. بحث ما بر روی آرایه‌های دو بعدی (یا دو شاخص‌دار) یا آرایه‌های که دارای چندین سطر از مقادیر هستند، متمرکز خواهد بود. طبق قرارداد برای مشخص کردن یکی از عناصر جدول، بایستی آنرا با دو شاخص نشان دهیم. شاخص اول، نشاندهنده سطر و شاخص دوم نشاندهنده ستون است. در شکل ۷-۲۱ یک آرایه دو بعدی منظم بنام **a**، حاوی سه سطر و چهار ستون دیده می‌شود. یک آرایه دو بعدی با **m** سطر و **n** ستون، یک آرایه **n** در **m** نامیده می‌شود.

سطر 0	ستون 0	ستون 1	ستون 2	ستون 3
سطر 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]



شکل ۷-۲۱ | آرایه دو بعدی با سه سطر و چهار ستون.

هر عنصر در آرایه a ، بصورت $a[i][j]$ مشخص می‌شود، که در این فرم a نام آرایه، j و i شاخص‌های آن هستند که بصورت منحصر بفرد نشانده‌اند سطر و ستون هر عنصر در آرایه a است. توجه کنید که بدليل عملکرد شاخص‌ها بر مبنای شمارش از صفر، اسمی عناصر در سطر اول دارای شاخص، 0 هستند و اسمی عناصر در ستون چهارم دارای شاخص، 3 می‌باشند.

خطای برنامه‌نویسی



مراجعه به یک عنصر آرایه دو بعدی $a[x][y]$ بصورت $a[x][y]$ خطأ است. در واقع با $a[x, y]$ بصورت $a[y]$ رفتار می‌شود، چرا که $C++$ عبارت y, x را فقط بعنوان y ارزیابی می‌کند.

مقداردهی آرایه‌های چند بعدی در اعلان‌ها همانند آرایه‌های یک بعدی صورت می‌گیرد. برای مثال، یک آرایه دو بعدی منظم بنام b با مقادیر 1 و 2 در سطر 0 و مقادیر 3 و 4 در سطر 1 را می‌توان بصورت زیر اعلان و مقداردهی اولیه کرد:

```
int b[2][2] = {{1,2},{3,4}};
```

مقادیر گروه‌بندی شده با سطر در درون برآکت‌ها، اقدام به مقداردهی $[0][0]$ با 1 و $[1][0]$ با 2 می‌کنند و $[0][1]$ با 3 و $[1][1]$ با 4 مقداردهی می‌شود. در آرایه‌های منظم، هر سطر دارای شماره یکسان است.

برنامه شکل ۷-۲۲، نحوه مقداردهی اولیه یک آرایه در زمان اعلان را نشان می‌دهد و همچنین با استفاده از حلقه‌های تودرتوی **for** اقدام به حرکت در میان عناصر آرایه‌ها شده است.

```

1 // Fig. 7.22: fig07_22.cpp
2 // Initializing multidimensional arrays.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 void printArray( const int [][] [ 3 ] ); // prototype
8
9 int main()
10 {
11     int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
12     int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };

```



```
13 int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };
14 cout << "Values in array1 by row are:" << endl;
15 printArray( array1 );
16
17 cout << "\nValues in array2 by row are:" << endl;
18 printArray( array2 );
19
20 cout << "\nValues in array3 by row are:" << endl;
21 printArray( array3 );
22 return 0; // indicates successful termination
23
24 } // end main
25
26 // output array with two rows and three columns
27 void printArray( const int a[][ 3 ] )
28 {
29     // loop through array's rows
30     for ( int i = 0; i < 2; i++ )
31     {
32         // loop through columns of current row
33         for ( int j = 0; j < 3; j++ )
34             cout << a[ i ][ j ] << ' ';
35
36         cout << endl; // start new line of output
37     } // end outer for
38 } // end function printArray
```

```
Values in array1 by row are:
```

```
1 2 3
4 5 6
```

```
Values in array1 by row are:
```

```
1 2 3
4 5 0
```

```
Values in array1 by row are:
```

```
1 2 0
4 0 0
```

شکل ۷-۲۲ | مقداردهی آرایه‌های چند بعدی.

در اعلان آرایه **array1** (خط 11) اقدام به مقداردهی شش عنصر در دو زیر لیست شده است. اولین زیرلیست، سطر اول (سطر صفر) از آرایه را با مقادیر 3 و 2، 1 و زیر لیست دوم، سطر دوم (سطر یک) از آرایه را با مقادیر 6 و 5، 4 مقداردهی می‌کند. اگر براکت‌های اطراف هر زیر لیست از لیست مقداردهی اولیه **array1** حذف شود، کامپایلر مبادرت به مقداردهی اولیه عناصر سطر صفر بدنال عناصر سطر یک می‌کند.

اعلان **array2** در خط 12 فقط حاوی پنج مقداردهی اولیه است. مقدار دهی به سطر صفر و سپس سطر یک تخصیص یافته‌اند. هر عنصری که دارای مقداردهی صریح نباشد با صفر مقداردهی اولیه می‌شود، از اینرو **array2[1][2]** با صفر مقداردهی اولیه می‌شود.

اعلان صورت گرفته برای **array3** در خط 13 سه مقداردهی در دو زیر لیست تدارک دیده است. زیر لیست برای سطر صفر بصورت صریح، دو عنصر اول از سطر صفر را با 1 و 2 مقداردهی می‌کند و عنصر سوم بصورت ضمنی با صفر مقداردهی می‌شود. زیر لیست سطر 1 بصورت صریح، اولین عنصر را با 4 و دو عنصر آخر را بصورت ضمنی با صفر مقداردهی می‌کند.



برنامه، تابع `printArray` را برای چاپ عناصر آرایه فراخوانی می‌کند. توجه کنید که در تعریف تابع (خطوط 38-27) پارامتر `const int a[][3]` مشخص شده است. زمانیکه تابع، آرایه یک بعدی را بعنوان آرگومان دریافت می‌کند، برآکت‌های آرایه در لیست پارامتری تابع خالی هستند. نیازی نیست که سایز بُعد اول (یعنی تعداد سطرها) در یک آرایه دو بعدی مشخص گردد، اما به سایز بُعد، بعدی نیاز است. کامپایلر با استفاده از این سایزها موقعیت عناصر آرایه چند بُعدی در حافظه را تعیین می‌کند. تمام عناصر آرایه صرفظ از تعداد ابعاد آرایه بصورت متواالی در حافظه ذخیره می‌شوند. در یک آرایه دو بُعدی، سطر صفرم در حافظه‌ای که بدنیال سطر یکم قرار دارد ذخیره می‌شود. در یک آرایه دو بُعدی، هر سطر، یک آرایه یک بُعدی است. برای یافتن یک عنصر در یک سطر خاص، بایستی تابع بطور دقیق از تعداد عناصر موجود در هر سطر مطلع باشد، بنابر این می‌تواند به تعداد صحیح از روی مکان‌های حافظه پرسش کند. از این‌رو، به هنگام دسترسی به `a[1][2]` تابع می‌داند که باید از سه عنصر سطر صفرم در حافظه پرسش کند تا به سطر یکم دست یابد. سپس تابع به عنصر 2 از آن سطر دسترسی پیدا می‌کند.

در بسیاری از آرایه‌ها، از ساختار تکرار `for` برای کار با عناصر آرایه استفاده می‌شود. فرض کنید که آرایه‌ای بنام `a` موجود است. ساختار `for` تمام عناصر موجود در سطر دوم از آرایه `a` را با صفر تنظیم می‌کند:

```
for (column = 0; column < 4; column++)
  a[2][column] = 0;
```

در این عبارت، سطر سوم مشخص شده و از این‌رو می‌دانیم که اولین شاخص برابر 2 خواهد بود. (0) سطر اول و 1 سطر دوم است). حلقه `for` فقط بر روی شاخص دوم عمل می‌کند (شاخص ستون). ساختار `for` قبلی معادل عبارات تخصیصی زیر است:

```
a[2][0] = 0;
a[2][1] = 0;
a[2][2] = 0;
a[2][3] = 0;
```

ساختار تودرتوی `for` که در زیر آورده شده، مجموع تمام عناصر موجود در آرایه `a` را تعیین می‌کند:

```
total = 0;
for (row = 0; row < 3; row++)
  for (col = 0; column < 4; column++)
    total += a[row][column];
```

ساختار تودرتوی `for` مجموع تعداد عناصر در یک سطر را در هر بار محاسبه می‌کند. ساختار `for` خارجی با تنظیم شاخص `row` با 0 شروع شده و بنابراین عناصر سطر اول می‌تواند با ساختار داخلی `for`



محاسبه گردد. سپس **for** خارجی اقدام به افزایش **row** به 1 می‌کند و بنابراین سطر دوم می‌تواند محاسبه شود. ساختار خارجی **for**، مقدار **row** را به 2 افزایش می‌دهد و از این‌رو سطر سوم نیز محاسبه می‌شود. زمانیکه ساختار خارجی **for** به اتمام برسد، نتایج به نمایش در می‌آیند.

۷-۱۰ مبحث آموزشی: کلاس GradeBook با استفاده از آرایه دو بعدی

در بخش ۷-۶ به معرفی کلاس GradeBook (برنامه‌های ۷-۱۶ و ۷-۱۷) پرداختیم که از یک آرایه، یک بعدی برای ذخیره نمرات دانشجویان در یک امتحان استفاده می‌کرد. در اکثر ترم‌ها، دانشجویان بیش از یک امتحان برگزار می‌کنند. امکان دارد استاد مایل باشند تا نمرات یک ترم را تحلیل نمایند، هم برای دانشجو و هم برای کل کلاس.

ذخیره‌سازی نمرات دانشجویان در یک آرایه دو بعدی در کلاس GradeBook

برنامه‌های شکل ۷-۲۳ و ۷-۲۴ حاوی نسخه‌ای از کلاس GradeBook هستند که از یک آرایه دو بعدی **grades** برای ذخیره سازی تعداد نمرات دانشجویان در چند آزمون استفاده می‌کنند. هر سطر از آرایه نشانده‌نده نمرات یک دانشجو در کل دوره بوده و هر ستون نشانده‌نده تمام نمرات کل دانشجویان در یک امتحان خاص است. یک برنامه سرویس‌گیرنده همانند fig07_25.cpp آرایه‌ای را بعنوان یک آرگومان به سازنده GradeBook ارسال می‌کند. در این مثال، از یک آرایه 10 در 3 حاوی نمرات ده دانشجو در سه آزمون استفاده کرده‌ایم.

پنج تابع عضو (اعلان شده در خطوط 27-23 از شکل ۷-۲۳) برای کار با آرایه به منظور پردازش نمرات بکار گرفته شده‌اند. هر کدامیک از این توابع عضو مشابه توابع موجود در آرایه یک بعدی نسخه ۷-۱۶ و ۷-۱۷ GradeBook هستند (برنامه‌های ۷-۱۶ و ۷-۱۷). تابع عضو **getMinimum** (تعریف شده در خطوط 65-82 شکل ۷-۲۴) تعیین کننده کمترین نمره هر دانشجو در ترم است. تابع عضو **getMaximum** تعیین کننده بالاترین نمره هر دانشجو در ترم است (تعریف شده در خطوط 102-85 از شکل ۷-۲۴). تابع عضو **getAverage** (خطوط 115-105 از شکل ۷-۲۴) تعیین کننده میانگین یک دانشجو در طول ترم است. تابع عضو **outputBarChart** (خطوط 118-149 از شکل ۷-۲۴) نمودار میله‌ای از توزیع نمرات دانشجویان در طول ترم است. تابع عضو **outputGrades** (خطوط 152-177 از شکل ۷-۲۴) محتویات آرایه دو بعدی را در فرمت جدولی در کنار میانگین ترمی هر دانشجو چاپ می‌کند.

```

1 // Fig. 7.23: GradeBook.h
2 // Definition of class GradeBook that uses a
3 // two-dimensional array to store test grades.
4 // Member functions are defined in GradeBook.cpp

```



```

5 #include <string> // program uses C++ Standard Library string class
6 using std::string;
7
8 // GradeBook class definition
9 class GradeBook
10 {
11 public:
12     // constants
13     const static int students = 10; // number of students
14     const static int tests = 3; // number of tests
15
16     // constructor initializes course name and array of grades
17     GradeBook( string, const int [] [ tests ] );
18
19     void setCourseName( string ); // function to set the course name
20     string getCourseName(); // function to retrieve the course name
21     void displayMessage(); // display a welcome message
22     void processGrades(); // perform various operations on the grade data
23     int getMinimum(); // find the minimum grade in the grade book
24     int getMaximum(); // find the maximum grade in the grade book
25     double getAverage( const int [], const int ); // find average of grades
26     void outputBarChart(); // output bar chart of grade distribution
27     void outputGrades(); // output the contents of the grades array
28 private:
29     string courseName; // course name for this grade book
30     int grades[ students ] [ tests ]; // two-dimensional array of grades
31 } // end class GradeBook

```

شکل ۷-۲۳ | تعریف کلاس GradeBook با یک آرایه دو بعدی برای ذخیره‌سازی نمرات.

```

1 // Fig. 7.24: GradeBook.cpp
2 // Member-function definitions for class GradeBook that
3 // uses a two-dimensional array to store grades.
4 #include <iostream>
5 using std::cout;
6 using std::cin;
7 using std::endl;
8 using std::fixed;
9
10 #include <iomanip> // parameterized stream manipulators
11 using std::setprecision; // sets numeric output precision
12 using std::setw; // sets field width
13
14 // include definition of class GradeBook from GradeBook.h
15 #include "GradeBook.h"
16
17 // two-argument constructor initializes courseName and grades array
18 GradeBook::GradeBook( string name, const int gradesArray [] [ tests ] )
19 {
20     setCourseName( name ); // initialize courseName
21
22     // copy grades from gradeArray to grades
23     for ( int student = 0; student < students; student++ )
24
25         for ( int test = 0; test < tests; test++ )
26             grades[ student ] [ test ] = gradesArray[ student ] [ test ];
27 } // end two-argument GradeBook constructor
28
29 // function to set the course name
30 void GradeBook::setCourseName( string name )
31 {
32     courseName = name; // store the course name
33 } // end function setCourseName
34
35 // function to retrieve the course name
36 string GradeBook::getCourseName()
37 {
38     return courseName;
39 } // end function getCourseName
40

```



```
41 // display a welcome message to the GradeBook user
42 void GradeBook::displayMessage()
43 {
44     // this statement calls getCourseName to get the
45     // name of the course this GradeBook represents
46     cout << "Welcome to the grade book for\n" << getCourseName() << "!"
47     << endl;
48 } // end function displayMessage
49
50 // perform various operations on the data
51 void GradeBook::processGrades()
52 {
53     // output grades array
54     outputGrades();
55
56     // call functions getMinimum and getMaximum
57     cout << "\nLowest grade in the grade book is " << getMinimum()
58     << "\nHighest grade in the grade book is " << getMaximum() << endl;
59
60     // output grade distribution chart of all grades on all tests
61     outputBarChart();
62 } // end function processGrades
63
64 // find minimum grade
65 int GradeBook::getMinimum()
66 {
67     int lowGrade = 100; // assume lowest grade is 100
68
69     // loop through rows of grades array
70     for ( int student = 0; student < students; student++ )
71     {
72         // loop through columns of current row
73         for ( int test = 0; test < tests; test++ )
74         {
75             // if current grade less than lowGrade, assign it to lowGrade
76             if ( grades[ student ][ test ] < lowGrade )
77                 lowGrade = grades[ student ][ test ]; // new lowest grade
78         } // end inner for
79     } // end outer for
80
81     return lowGrade; // return lowest grade
82 } // end function getMinimum
83
84 // find maximum grade
85 int GradeBook::getMaximum()
86 {
87     int highGrade = 0; // assume highest grade is 0
88
89     // loop through rows of grades array
90     for ( int student = 0; student < students; student++ )
91     {
92         // loop through columns of current row
93         for ( int test = 0; test < tests; test++ )
94         {
95             // if current grade greater than lowGrade, assign it to highGrade
96             if ( grades[ student ][ test ] > highGrade )
97                 highGrade = grades[ student ][ test ]; // new highest grade
98         } // end inner for
99     } // end outer for
100
101    return highGrade; // return highest grade
102 } // end function getMaximum
103
104 // determine average grade for particular set of grades
105 double GradeBook::getAverage( const int setOfGrades[], const int grades )
106 {
107     int total = 0; // initialize total
108
109     // sum grades in array
110     for ( int grade = 0; grade < grades; grade++ )
```



```

111     total += setOfGrades[ grade ];
112
113     // return average of grades
114     return static_cast< double >( total ) / grades;
115 } // end function getAverage
116
117 // output bar chart displaying grade distribution
118 void GradeBook::outputBarChart()
119 {
120     cout << "\nOverall grade distribution:" << endl;
121
122     // stores frequency of grades in each range of 10 grades
123     const int frequencySize = 11;
124     int frequency[ frequencySize ] = { 0 };
125
126     // for each grade, increment the appropriate frequency
127     for ( int student = 0; student < students; student++ )
128
129         for ( int test = 0; test < tests; test++ )
130             ++frequency[ grades[ student ][ test ] / 10 ];
131
132     // for each grade frequency, print bar in chart
133     for ( int count = 0; count < frequencySize; count++ )
134     {
135         // output bar label ("0-9:", ..., "90-99:", "100:")
136         if ( count == 0 )
137             cout << " 0-9: ";
138         else if ( count == 10 )
139             cout << " 100: ";
140         else
141             cout << count * 10 << "-" << ( count * 10 ) + 9 << ": ";
142
143         // print bar of asterisks
144         for ( int stars = 0; stars < frequency[ count ]; stars++ )
145             cout << '*';
146
147         cout << endl; // start a new line of output
148     } // end outer for
149 } // end function outputBarChart
150
151 // output the contents of the grades array
152 void GradeBook::outputGrades()
153 {
154     cout << "\nThe grades are:\n\n";
155     cout << "          "; // align column heads
156
157     // create a column heading for each of the tests
158     for ( int test = 0; test < tests; test++ )
159         cout << "Test " << test + 1 << " ";
160
161     cout << "Average" << endl; // student average column heading
162
163     // create rows/columns of text representing array grades
164     for ( int student = 0; student < students; student++ )
165     {
166         cout << "Student " << setw( 2 ) << student + 1;
167
168         // output student's grades
169         for ( int test = 0; test < tests; test++ )
170             cout << setw( 8 ) << grades[ student ][ test ];
171
172         // call member function getAverage to calculate student's average;
173         // pass row of grades and the value of tests as the arguments
174         double average = getAverage( grades[ student ], tests );
175         cout << setw( 9 ) << setprecision( 2 ) << fixed << average << endl;
176     } // end outer for
177 } // end function outputGrades

```



تابع عضو `outputGrades` و `outputBarChart` توسط عبارات `getMaximum` و `getMinimum` یک به یک بکار گرفته شده و بر روی آرایه `grades` اعمال می‌شوند. برای مثال به عبارت `for` تودرتو در تابع عضو `getMinimum` (خطوط 70-79) توجه کنید. عبارت `for` خارجی با تنظیم `student` با صفر شروع می‌شود (یعنی شاخص سطر)، از این‌رو عناصر سطر صفر می‌توانند با متغیر `lowGrade` در بدنه عبارت `for` داخلی مقایسه گردند. حلقه عبارت `for` داخلی در میان نمرات یک سطر خاص حرکت کرده و هر نمره را با `lowGrade` مقایسه می‌کند. اگر نمره کمتر از `lowGrade` باشد، `lowGrade` را با آن نمره تنظیم می‌کند. سپس عبارت `for` خارجی مبادرت به افزایش شاخص سطر به 1 می‌کند. عناصر سطر 1 با متغیر `lowGrade` مقایسه می‌شوند. سپس عبارت `for` خارجی مقدار شاخص سطر را به 2 افزایش می‌دهد و عناصر سطر 2 با متغیر `lowGrade` مقایسه می‌شوند. این عمل تا پیمایش تمام سطرهای `grades` تکرار می‌گردد. پس از کامل شدن اجرای عبارت تودرتو، متغیر `lowGrade` حاوی کمترین نمره در آرایه دو بعدی خواهد بود. عملکرد تابع عضو `getMinimum` شبیه تابع عضو `getMaximum` است.

تابع عضو `outputBarChart` در شکل ۷-۲۴ تقریباً مشابه با تابع موجود در شکل ۷-۱۷ است. با این همه، کل نمرات یک ترم را به نمایش در می‌آورد. این تابع عضو از یک `for` تودرتو (خطوط 127-130) برای ایجاد یک آرایه یک بعدی `frequency` برپایه تمام نمرات در آرایه دو بعدی استفاده کرده است. مابقی کد در هر دو تابع عضو `outputBarChart` که نمودار را نمایش درمی‌آورند، یکسان است. تابع عضو `outputGrades` (خطوط 152-177) هم از عبارت `for` تودرتو برای چاپ مقادیر آرایه `grades` به همراه میانگین نمرات هر دانشجو استفاده کرده است. خروجی برنامه شکل ۷-۲۵ نشانده‌نده نتیجه با فرمت جدولی است. خطوط 158-159 سرآیند ستون برای هر تست را چاپ می‌کند. از یک عبارت `for` به روش شمارنده کنترل استفاده کرده‌ایم تا بتوانیم هر تست را با یک عدد تشخیص دهیم. به همین ترتیب، عبارت `for` در خطوط 164-176 برچسب اولین سطر را با استفاده از یک متغیر شمارنده برای شناسایی هر دانشجو چاپ می‌کند (خط 166).

```
1 // Fig. 7.25: fig07_25.cpp
2 // Creates GradeBook object using a two-dimensional array of grades.
3
4 #include "GradeBook.h" // GradeBook class definition
5
6 // function main begins program execution
7 int main()
8 {
9     // two-dimensional array of student grades
10    int gradesArray[ GradeBook::students ][ GradeBook::tests ] =
11        { { 87, 96, 70 },
12          { 68, 87, 90 },
13          { 94, 100, 90 },
14          { 100, 81, 82 },
15          { 83, 65, 85 },
16          { 78, 87, 65 },
17          { 85, 75, 83 },
18          { 91, 94, 100 },
```



```

19         { 76, 72, 84 },
20         { 87, 93, 73 } };
21
22 GradeBook myGradeBook(
23     "CS101 Introduction to C++ Programming", gradesArray );
24 myGradeBook.displayMessage();
25 myGradeBook.processGrades();
26 return 0; // indicates successful termination
27 } // end main

```

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

The grades are:

	Test 1	Test 2	Test 3	Average
student 1	87	96	70	84.33
student 2	68	87	90	81.67
student 3	94	100	90	94.67
student 4	100	81	82	87.67
student 5	83	65	85	77.67
student 6	78	87	65	76.67
student 7	85	75	83	81.00
student 8	91	94	100	95.00
student 9	76	72	84	77.33
student 10	87	93	73	84.33

Lowest grade in the grade book is 65
Highest grade in the grade book is 100

Overall grade distribution:

0-9:	
10-19:	
20-29:	
30-39:	
40-49:	
50-59:	
60-69:	***
70-79:	*****
80-89:	*****
90-99:	*****
100:	***

شکل ۷-۲۵ | ایجاد یک شی از کلاس GradeBook با استفاده از آرایه دو بعدی نمرات، و فراخوانی تابع `processGrades` برای تحلیل نمرات.

اگر چه شاخص آرایه با صفر شروع می‌شود، اما دقت کنید که در خطوط ۱۵۹ و ۱۶۶ خروجی با عبارت `student+1` و `test+1` تنظیم می‌شود، تا شماره تست و تعداد دانشجویان از ۱ شروع شود (به شکل ۷-۲۵ نگاه کنید). عبارت `for` داخلی در خطوط ۱۶۹-۱۷۰ از متغیر شمارنده `for` خارجی بنام `student` برای حرکت در میان یک سطر خاص از آرایه `grades` و چاپ نمره هر دانشجو استفاده کرده است. سرانجام، خط ۱۷۴ میانگین هر دانشجو را با ارسال سطر جاری از `grades[student]` (یعنی `grades[student]` به تابع `getAverage` بدهست می‌آورد. تابع `getAverage` (خطوط ۱۱۵-۱۰۵) دو آرگومان دریافت می‌کند (یک آرایه یک بعدی از نتایج آزمون برای یک دانشجو خاص و نتایج آزمون (تست) در آرایه). زمانیکه خط ۱۷۴ تابع `getAverage` را فراخوانی می‌کند، آرگومان اول `grades[student]` است که مشخص



کننده یک سطر خاص از آرایه دو بعدی `grades` است که باید به `getAverage` ارسال شود. برای مثال، برپایه آرایه ایجاد شده در برنامه شکل ۷-۲۵، آرگومان `grades[1]` نشانده‌نده سه مقدار (آرایه یک بعدی از نمرات) ذخیره شده در سطر ۱ از آرایه دو بعدی `grades` است. می‌توان به آرایه دو بعدی بصورت آرایه‌ای که عناصر آن در آرایه‌های یک بعدی قرار دارند، نگاه کرد.تابع عضو `getAverage` مجموع عناصر آرایه را محاسبه کرده، آنرا به تعداد نتایج آزمون تقسیم و نتیجه را بصورت یک مقدار `double` برگشت می‌دهد (خط ۱۱۴).

تست کلاس GradeBook

برنامه شکل ۷-۲۵ یک شی از کلاس `GradeBook` (شکل‌های ۷-۲۳ و ۷-۲۴) با استفاده از یک آرایه دو بعدی بنام `gradesArray` از نوع `int` ایجاد می‌کند (اعلان و مقداردهی شده در خطوط ۲۰-۲۱). توجه کنید که خط ۱۰ به ثابت استاتیک `GradeBook` کلاس `student` و `tests` دسترسی پیدا می‌کند تا سایز هر بعد آرایه `gradesArray` بدست آید. خطوط ۲۲-۲۳ نام دوره و `gradesArray` را به سازنده `GradeBook` ارسال می‌کنند. سپس خطوط ۲۴-۲۵ تابع عضو `processGrades` و `displayMessage` را به ترتیب برای نمایش پیغام خوش‌آمدگویی و بدست آوردن گزارشی از نمرات دانشجویان در طول ترم فراخوانی می‌کنند.

۷-۱۱ معرفی کلاس استاندارد `vector`

در این بخش به معرفی کلاس الگوی `vector` (بردار) از کتابخانه استاندارد C++ می‌پردازیم که عرضه کننده نوع قدرتمندی از آرایه با قابلیت‌های بیشتر است. همانطوری که در فصل‌های بعدی کتاب و دوره‌های C++ پیشرفته مشاهده خواهید کرد، آرایه‌های مبتنی بر اشاره‌گر در سبک C (نوع آرایه‌های معرفی شده تا بدين جا) زمینه بسیار زیادی برای تولید خطا دارند. برای مثال، همانطوری که قبل‌گفته شد، برنامه می‌تواند به آسانی از مز آرایه خارج شود، چرا که C++ کنترلی بر روی شاخص‌های آرایه انجام نمی‌دهد تا جلوی خارج شدن آنها را از مز آرایه بگیرد. دو آرایه را نمی‌توان بطور موثر با عملگرهای مقایسه‌ای یا رابطه‌ای با یکدیگر مقایسه کرد. همانطوری که در فصل ۸ خواهید آموخت، متغیرهای اشاره‌گر (که بعنوان اشاره‌گر شناخته می‌شوند) حاوی آدرس‌های حافظه بعنوان مقادیر خود هستند. اسامی آرایه‌ها اشاره‌گرهای ساده‌ای هستند که شروع آرایه در حافظه را نشان می‌دهند، و البته دو آرایه همیشه در مکان‌های مختلف حافظه جای داده می‌شوند. زمانیکه آرایه‌ای به یک تابع که برای کار با آرایه‌ها با هر سایز طراحی شده است، ارسال می‌گردد باید سایز آرایه هم در نظر گرفته شود. علاوه بر این، نمی‌توان یک آرایه را به کمک عملگر تخصیص به آرایه دیگری انتساب داد. اسامی آرایه‌ها از نوع اشاره‌گرهای ثابت (*const*) هستند و همانطوری که در فصل هشتم خواهید آموخت، یک ثابت اشاره‌گر نمی‌تواند در



سمت چپ یک عملگر تخصیص قرار داده شود. این قابلیت‌ها و رفتارهای دیگر به هنگام بررسی آرایه‌ها جزء ماهیت طبیعی آنها بنظر می‌رسند، اما C++ چنین قابلیت‌های را تدارک نمیدهد. با این همه کتابخانه استاندارد C++ دارای کلاس الگو بنام **vector** (بردار) است که به برنامه نویسان امکان ایجاد آرایه‌های بسیار قدرتمند و با خطاهای بسیار کم را می‌دهد. در فصل ۱۱، با قابلیت‌های بسیار زیاد **vector** آشنا خواهید شد.

کلاس **vector** برای استفاده در هر برنامه C++ در اختیار است. امکان دارد نشانه‌گذاری که در مثال‌های استفاده می‌کنیم برای شما چندان آشنا نباشد، چرا که بردارها از نشانه‌گذاری الگو استفاده می‌کنند. بخش ۱۸-۶ را که در ارتباط با الگوهای تابع بود بخاطر آورید. در فصل ۱۴ در ارتباط با الگوهای کلاس صحبت خواهیم کرد. برای این بخش کافیست به گرامر بکار رفته در مثال دقت کنید.

برنامه شکل ۷-۲۶ به توصیف قابلیت‌های تدارک دیده شده توسط کلاس **vector** پرداخته که در آرایه‌های مبتنی بر اشاره‌گر سبک C وجود ندارند. کلاس **vector** از بسیاری از ویژگیهای تدارک دیده شود توسط کلاس **Array** برخوردار است که در فصل ۱۱ به ساخت آنها اقدام خواهیم کرد. کلاس **vector** در سرآیند **<vector>** تعریف شده (خط ۱۱) و متعلق به فضای نامی **std** می‌باشد (خط ۱۲).

خطوط ۲۰-۱۹ دو شی **vector** برای ذخیره سازی مقادیری از نوع **int** بنام **integers1** حاوی هفت عنصر، و **integers2** حاوی ۱۰ عنصر ایجاد کرده‌اند. بطور پیش‌فرض، تمام عناصر هر شی **vector** با صفر تنظیم می‌شوند. دقت کنید که بردارها می‌توانند برای ذخیره سازی هر نوع داده با جایگزین کردن **int** در **<vector>** با نوع داده مقتضی تعریف شوند. این نشانه‌گذاری که تصریح کننده نوع ذخیره شده در **vector** است، مشابه نشانه‌گذاری الگوهای است که در بخش ۱۸-۶ توضیح داده شده است. در فصل ۱۴ با این گرامر به تفصیل آشنا خواهید شد.

```
1 // Fig. 7.26: fig07_26.cpp
2 // Demonstrating C++ Standard Library class template vector.
3 #include <iostream>
4 using std::cout;
5 using std::cin;
6 using std::endl;
7
8 #include <iomanip>
9 using std::setw;
10
11 #include <vector>
12 using std::vector;
13
14 void outputVector( const vector< int > & ); // display the vector
15 void inputVector( vector< int > & ); // input values into the vector
16
17 int main()
18 {
19     vector< int > integers1( 7 ); // 7-element vector< int >
20     vector< int > integers2( 10 ); // 10-element vector< int >
```



```
21 // print integers1 size and contents
22 cout << "Size of vector integers1 is " << integers1.size()
23     << "\nvector after initialization:" << endl;
24     outputVector( integers1 );
25
26 // print integers2 size and contents
27 cout << "\nSize of vector integers2 is " << integers2.size()
28     << "\nvector after initialization:" << endl;
29     outputVector( integers2 );
30
31 // input and print integers1 and integers2
32 cout << "\nEnter 17 integers:" << endl;
33     inputVector( integers1 );
34     inputVector( integers2 );
35
36 cout << "\nAfter input, the vectors contain:\n"
37     << "integers1:" << endl;
38     outputVector( integers1 );
39     cout << "integers2:" << endl;
40     outputVector( integers2 );
41
42 // use inequality (!=) operator with vector objects
43 cout << "\nEvaluating: integers1 != integers2" << endl;
44
45 if ( integers1 != integers2 )
46     cout << "integers1 and integers2 are not equal" << endl;
47
48 // create vector integers3 using integers1 as an
49 // initializer; print size and contents
50 vector< int > integers3( integers1 ); // copy constructor
51
52 cout << "\nSize of vector integers3 is " << integers3.size()
53     << "\nvector after initialization:" << endl;
54     outputVector( integers3 );
55
56 // use assignment (=) operator with vector objects
57 cout << "\nAssigning integers2 to integers1:" << endl;
58     integers1 = integers2; // integers1 is larger than integers2
59
60 cout << "integers1:" << endl;
61     outputVector( integers1 );
62     cout << "integers2:" << endl;
63     outputVector( integers2 );
64
65 // use equality (==) operator with vector objects
66 cout << "\nEvaluating: integers1 == integers2" << endl;
67
68 if ( integers1 == integers2 )
69     cout << "integers1 and integers2 are equal" << endl;
70
71 // use square brackets to create rvalue
72 cout << "\nintegers1[5] is " << integers1[ 5 ];
73
74 // use square brackets to create lvalue
75 cout << "\n\nAssigning 1000 to integers1[5]" << endl;
76     integers1[ 5 ] = 1000;
77     cout << "integers1:" << endl;
78     outputVector( integers1 );
79
80 // attempt to use out-of-range subscript
81 cout << "\nAttempt to assign 1000 to integers1.at( 15 )" << endl;
82     integers1.at( 15 ) = 1000; // ERROR: out of range
83
84 return 0;
85 } // end main
86
87 // output vector contents
88 void outputVector( const vector< int > &array )
89 {
90     size_t i; // declare control variable
```



```

91     for ( i = 0; i < array.size(); i++ )
92     {
93         cout << setw( 12 ) << array[ i ];
94
95         if ( ( i + 1 ) % 4 == 0 ) // 4 numbers per row of output
96             cout << endl;
97     } // end for
98
99
100    if ( i % 4 != 0 )
101        cout << endl;
102 } // end function outputVector
103
104 // input vector contents
105 void inputVector( vector< int > &array )
106 {
107     for ( size_t i = 0; i < array.size(); i++ )
108         cin >> array[ i ];
109 } // end function inputVector
Size of vector integers1 is 7
vector after initialization:
      0          0          0          0
      0          0          0          0

Size of vector integers2 is 10
vector after initialization:
      0          0          0          0
      0          0          0          0
      0          0          0          0

Enter 17 integers:
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17

After input, the vectors contain:
integers1:
      1          2          3          4
      5          6          7
integers2:
      8          9          10         11
      12         13         14         15
      16         17
Evaluating: integers1 != integers2
integers1 and integeres2 are not equal

Size of vector integers3 is 7
vector after initialization:
      1          2          3          4
      5          6          7
Assigning integers2 to integers1:
inregers1:
      8          9          10         11
      12         13         14         15
      16         17
inregers2:
      8          9          10         11
      12         13         14         15
      16         17
Evaluating: integers1 == integers2
integers1 and integers2 are equal

integers1[5] is 13

assigning 1000 to integers[5]

```



```
integer1:  
 8          9          10         11  
 12         1000       14         15  
 16         17  
Attempt to assign 1000 to integers1.at( 15 )  
abnormal program termination
```

شکل ۷-۲۶ | کلاس استاندارد **vector**.

در خط 23 از تابع عضو بردار **size** برای آوردن سایز (یعنی تعداد عناصر) **integers1** استفاده شده است. در خط 25، 25 به تابع **integers1** به تابع **outputVector** (خطوط 102-88) ارسال شده که از براکت‌ها ([]) برای تهیه مقدار هر عنصر از **vector** **vector** بعنوان مقداری که بتوان برای خروجی استفاده کرد، استفاده کرده است. به شباخت این نشانه‌گذاری با نشانه‌گذاری بکار رفته برای دسترسی به مقدار یک عنصر آرایه دقت کنید. خطوط 28 و 30 همان وظایف را برای **integers2** انجام می‌دهند.

تابع عضو **size** از کلاس **vector** تعداد عناصر در یک بردار را بعنوان مقداری از نوع **size_t** برگشت می‌دهد (که نشاندهنده نوع **unsigned int** بر روی بسیاری از سیستم‌ها است). بعنوان نتیجه، خط 90 متغیر کنترلی **i** را از نوع **size_t** اعلام کرده است. زمانیکه شرط تکرار حلقه (خط 92) با یک مقدار **signed** (یعنی **i** (int) و یک مقدار **unsigned** (یعنی مقداری از نوع **size_t** برگشتی از تابع **size**) مقایسه شود و اعلام **i** بعنوان یک **int** سبب می‌شود برخی از کامپایلرها یک پیغام هشدار صادر نماید.

خطوط 34-35 مبادرت به ارسال **integers1** و **integers2** به تابع **inputVector** (خطوط 105-109) برای خواندن مقادیر برای عناصر هر **vector** از سوی کاربر می‌کنند. تابع **inputVector** از براکت‌ها ([]) برای بدست آوردن مقادیر سمت چپ (*lvalues*) که می‌توانند برای ذخیره سازی مقادیر ورودی در هر عنصر **vector** بکار گرفته شوند، استفاده کرده است.

خط 46 بیان می‌کند که شی‌های **vector** می‌توانند بطور مستقیم با عملگر **!=** مقایسه کردند. اگر محتویات دو بردار برابر نباشند، عملگر مقدار **true** و در غیر اینصورت **false** برگشت می‌دهد.

کلاس **vector** به برنامه‌نویسان اجازه می‌دهد تا یک شی جدید **vector** ایجاد کنند که با محتویات **vector** موجود مقداردهی اولیه شود. در خط 51 یک شی بردار (**integers3**) ایجاد و با کمی از **integers1** مقداردهی شده است. با اینکار سازنده کمی کننده بردار برای انجام عملیات کمی، فعل می‌شود. در فصل ۱۱ با این نوع از سازنده‌ها بیشتر آشنا خواهید شد. خطوط 53 و 55 سایز و محتویات **integers3** را برای نشان دادن اینکه عملیات مقداردهی بدرستی صورت گرفته، به نمایش در می‌آورند.

خط 59 برای توصیف اینکه می‌توان از عملگر تخصیص (=) در شی‌های برداری استفاده کرده، مبادرت به تخصیص **integers1** کرده است. خطوط 62 و 64 محتویات هر دو شی را برای نمایش اینکه هر دو دارای مقادیر یکسان هستند در خروجی ظاهر کرده‌اند. سپس خط 69 مبادرت به مقایسه



از **integers1** با **integers2** توسط عملگر تساوی (=) کرده تا تعیین کند که محتويات دو شی پس از انجام تخصیص در خط ۵۹ یکسان هستند. خطوط ۷۳ و ۷۷ نشان می‌دهند که برنامه می‌تواند از جفت برآکت‌ها ([]) برای بدست آوردن یک عنصر بردار بعنوان یک *lvalue* غیرقابل تغییر و بعنوان یک *lvalue* تغییرپذیر استفاده کند. یک *lvalue* تغییر نیافته عبارتی است که شی موجود در حافظه را نشان می‌دهد (همانند یک عنصر در بردار)، اما نمی‌تواند برای تغییر شی بکار گرفته شود. همچنین یک مقدار *lvalue* تغییرپذیر هم شناسه‌ای از یک شی در حافظه است، اما می‌تواند برای اصلاح و تغییر شی بکار گرفته شود. همانند آرایه‌های مبتنی بر اشاره گر سبک C، زبان C++ در مورد بررسی مرزها به هنگام دسترسی به عناصر بردار با استفاده از برآکت‌ها هیچ تستی انجام نمی‌دهد. از اینرو، بایستی برنامه‌نویس از عملکرد عباراتی که از برآکت‌ها استفاده می‌کند مطمئن گردد تا بطور تصادفی مبادرت به دستکاری عناصری خارج از مرزهای بردار نکند. با این همه، کلاس **الگوی استاندارد vector** دارای تابع عضو **at** است که به بررسی مرزها با به راه‌انداختن یک استثناء می‌پردازد (فصل شانزدهم). اگر آرگومان این تابع یک شاخص معتبر نباشد، یک استثناء بوجود می‌آورد. بطور پیش‌فرض این حالت سبب خاتمه یافتن برنامه C++ می‌شود. اگر شاخص معتبر باشد، تابع **at** مبادرت به برگشت دادن آن از آن مکان بعنوان یک *lvalue* تغییرپذیر یا غیرقابل تغییر براساس زمینه (غیرثابت یا ثابت) در فراخوانی صورت گرفته می‌کند. در خط ۸۳ نمونه‌ای از فراخوانی تابع با یک شاخص غیرمعتبر نشان داده شده است.

۷-۱۲ مبحث آموزشی مهندسی نرم‌افزار: همکاری مابین شی‌های سیستم ATM

در این بخش، به بررسی همکاری (تعاملات) صورت گرفته مابین شی‌های موجود در سیستم ATM می‌پردازیم. زمانیکه دو شی برای انجام دادن یک وظیفه با یکدیگر ارتباط برقرار می‌کنند، گفته می‌شود که باهم همکاری می‌نمایند و اینکار با فعال کردن عملیات‌ها در سمت مقابل صورت می‌گردد. همکاری متشکل از ارسال یک پیغام از سوی شی از یک کلاس به شی از یک کلاس دیگر است. پیغام‌ها در C++ از طریق فراخوانی تابع عضو ارسال می‌شوند.

در بخش ۶-۱۸ به بررسی تعدادی از عملیات‌ها که در کلاس‌های سیستم ATM رخ می‌دادند پرداختیم. در این بخش، تمرکز ما بر روی پیغام‌های است که این عملیات‌ها را فعال می‌سازند. برای شناسایی همکاری‌های صورت گرفته در سیستم به مستند نیازها در بخش ۲-۸ مراجعه می‌کنیم. بخاطر دارید که این مستند تعیین کننده محدوده فعالیت‌های است که در طول مدت زمان یک جلسه ATM اتفاق می‌افتد (همانند تایید کاربر، انجام تراکنش‌های لازم). اولین کار ما در شناسایی همکاری‌های صورت گرفته در



سیستم، بدست آوردن توصیفی از نحوه انجام این وظایف است. همانطوری که اینکار را ادامه می‌دهیم، در مابقی بخش‌های «مهندسی نرم‌افزار» به همکاری‌های بیشتری دست خواهیم یافت.

شناسایی همکاری‌ها در سیستم

برای شناسایی همکاری‌های موجود در سیستم بدقت شروع به مطالعه بخش‌های مربوط به مستند نیازها که تصریح کننده آنچه که با ATM در تایید کاربر و انجام هر نوع تراکنش صورت دهد می‌کنیم. برای هر عمل یا مرحله توضیح داده شده در مستند نیازها، تصمیم می‌گیریم که کدام شی‌ها در سیستم باید برای دست یافتن به نتایج دلخواه در تعامل قرار داده شوند. یک شی را بعنوان شی ارسال کننده (شی که پیغام ارسال می‌کند) و دیگری را بعنوان شی دریافت کننده مشخص می‌کنیم (شی که به سرویس‌گیرنده‌های کلاس پیشنهاد عملیاتی را می‌کند). سپس یکی از عملیات‌های شی دریافت کننده را انتخاب می‌کنیم (مشخص شده در بخش ۱۸-۶) که باید توسط شی ارسال کننده برای انجام یک رفتار مناسب فعال شود. برای مثال، ATM در زمان بیکاری پیغام خوش‌آمدگویی را به نمایش در می‌آورد. می‌دانیم که شی از کلاس Screen پیغامی را به کاربر از طریق عملیات displayMessage خود به نمایش در می‌آورد. از اینرو، تصمیم می‌گیریم که سیستم قادر به نمایش پیغام خوش‌آمدگویی از طریق یک همکاری مابین ATM و Screen باشد به نحوی که ATM پیغام displayMessage را به Screen از طریق فعال کردن عملیات displayMessage از کلاس Screen ارسال می‌کند.

در جدول ۷-۲۷ لیست همکاری‌های که می‌توان از مستند نیازها بدست آورد مشخص شده‌اند. برای هر شی ارسال کننده، براساس ترتیب موجود در مستند نیازها، اقدام به لیست همکاری‌ها کرده‌ایم. هر همکاری را با یک فرستنده منحصر بفرد، پیغام و دریافت کننده و فقط یکبار لیست کرده‌ایم، حتی اگر همکاری چندین بار در یک جلسه ATM رخ دهد. برای مثال در سطر اول جدول ۷-۲۷ مشخص است که همکاری با Screen ATM هر زمانیکه ATM نیاز بنمایش پیغامی به کاربر داشته باشد رخ می‌دهند.

باشی از کلاس	ارسال پیغام	شی از کلاس
ATM	displayMessage getInput authenticateUser execute execute execute	Screen keypad BankDatabase BalanceInquiry Withdrawal Deposit
BalanceInquiry	getAvailableBalance getTotalBalance displayMessage	BankDatabase BankDatabase Screen
Withdrawal	displayMessage getInput getAvailableBalance isSufficientCashAvailable debit dispenseCash	Screen Keypad BankDatabase CashDispenser BankDatabase CashDispenser



Deposit	<code>displayMessage</code> <code>getInput</code> <code>isEnvelopeReceived</code> <code>credit</code>	Screen Keypad DepositSlot BankDatabase
BankDatabase	<code>validatePIN</code> <code>getAvailableBalance</code> <code>getTotalBalance</code> <code>debit</code> <code>credit</code>	Account Account Account Account Account

شکل ۷-۲۷ | همکاری‌های موجود در سیستم ATM

اجازه دهید به بررسی همکاری‌های موجود در جدول ۷-۲۷ پردازیم. قبل از اینکه کاربر اجازه هر نوع تعاملی را بدست آورده باشد، بایستی ATM به کاربر اعلان کند تا شماره حساب، سپس PIN خود را وارد سازد. هر کدامیک از این وظایف با ارسال پیغام `displayMessage` به **Screen** انجام می‌شود. هر دو این اعمال اشاره به همان همکاری مابین ATM و Screen دارند که در جدول ۷-۲۷ لیست شده‌اند. ورودی کاربر را در واکنش به اعلان و از طریق ارسال پیغام `getInput` به صفحه کلید (Keypad) بدهست می‌آورد. سپس باید ATM تعیین کند که آیا شماره حساب و PIN مشخص شده از سوی کاربر مطابق با حسابی در پایگاه داده است یا خیر. اینکار با ارسال پیغام `authenticateUser` (اعتبارسنجی کاربر) به **BankDatabase** یا پایگاهداده صورت می‌گیرد. بخاطر دارید که **BankDatabase** بطور مستقیم قادر به اعتبارسنجی کاربر نیست و فقط حساب (Account) کاربر می‌تواند به PIN کاربر برای تایید وی دسترسی داشته باشد. بنابر این در جدول ۷-۲۷ یک همکاری لیست شده که در آن **BankDatabase** مبادرت به ارسال پیغام `validatePIN` به یک Account می‌کند.

پس از تایید کاربر، ATM منوی اصلی را با ارسال دنباله‌ای از پیغام‌های از `displayMessage` به Screen و `getInput` به Keypad بدهست آوردن ورودی از منوی انتخابی با ارسال پیغام `execute` به **BalanceInquiry** کاربر، ATM مبادرت به اجرای تراکنش با ارسال پیغام `execute` به شی مناسب می‌کند (مثلاً به یک `Deposit` یا یک `Withdrawal`). برای مثال، اگر کاربر تقاضای میزان موجودی را کند، ATM یک پیغام `execute` به **BalanceInquiry** ارسال خواهد کرد.

با بررسی دقیق‌تر مستند نیازهای همکاری‌های صورت گرفته در میان هر نوع تراکنش آشکارتر می‌شود. یک `BalanceInquiry` میزان پول موجود در حساب کاربر را با ارسال پیغام `getAvailableBalance` به `BalanceInquiry` که به ارسال پیغام `getAvailableBalance` به حساب کاربر (Account) واکنش نشان می‌دهد، بدست می‌آورد. به همین ترتیب `BalanceInquiry` میزان پول موجود در یک سپرده را با ارسال پیغام `getTotalBalance` به `BankDatabase` بدست می‌آورد، که همان پیغام را به Account



کاربر ارسال می‌کند. برای نمایش هر دو مقدار از میزان موجودی کاربر در یک زمان، **BalanceInquiry** پیغام **displayMessage** را به **Screen** ارسال می‌کند.

یک **Withdrawal** (برداشت پول) دنباله‌ای از پیغام‌های **displayMessage** را به **Screen** برای نمایش یک منو از میزان پرداخت‌های استاندارد (مثلًا $\$20$, $\$40$, $\$60$, $\$100$, $\$200$) ارسال می‌کند. **Withdrawal** اقدام به ارسال پیغام **Keypad getInput** به **Withdrawal** می‌کند تا انتخاب کاربر از منو را بدست آورد. سپس تعیین می‌کند که آیا تقاضای میزان برداشت کمتر یا معادل میزان موجودی کاربر است یا خیر. **Withdrawal** می‌تواند میزان پول موجود در حساب کاربر را با ارسال پیغام **getAvailableBalance** به **BankDatabase** بدست آورد. سپس **Withdrawal** تست می‌کند که آیا برداشت کننده پول (جعبه پول) به میزان کافی پول نقد در خود دارد یا خیر و اینکار را با ارسال پیغام **isSufficientCashAvailable** به **CashDispenser** انجام می‌دهد. **Withdrawal** پیغام **debit** را به **BankDatabase** ارسال می‌کند تا از میزان موجودی کاربر کاسته شود. بخاطر دارید که بدھکار کردن حساب هم در **totalBalance** و **availableBalance** اتفاق می‌افتد. برای برداشت میزان پول درخواستی، **Withdrawal** پیغام **displayMessage** را به **CashDispenser** ارسال می‌کند. سرانجام، **CashDispenser** پیغام **dispenseCash** را به **Screen** ارسال می‌کند تا به کاربر برداشت پول را یادآوری کند.

به پیغام **execute** ابتدا با ارسال یک پیغام **displayMessage** به **Screen** برای اعلان میزان سپرده‌گذاری از سوی کاربر واکنش نشان می‌دهد. **Deposit** پیغام **Deposit** را به **Keypad** برای تهیه ورودی کاربر ارسال می‌کند. سپس پیغام **displayMessage** را به **Screen** می‌فرستد تا به کاربر اعلان کند که پاکت سپرده‌گذاری را وارد سازد. برای تعیین اینکه آیا شکاف سپرده‌گذاری پاکت سپرده را دریافت کرده است یا خیر، **Deposit** پیغام **isEnvelopeReceived** را به **DepositSlot** ارسال می‌نماید. **Deposit** اقدام به، به روز کردن حساب کاربر با ارسال پیغام **credit** به **BankDatabase** می‌کند و متعاقب آن یک پیغام **credit** به حساب کاربر ارسال می‌شود. بخاطر دارید که اعتبار افزوده شده به حساب فقط موجب افزایش میزان **totalBalance** می‌شود و تأثیری در **availableBalance** ندارد.

دیاگرام‌های تراکنش

اکنون که مجموعه‌ای از همکاری‌های ممکن‌های مابین شی‌های موجود در سیستم ATM خود را شناسایی کرده‌ایم، اجازه دهید تا این تراکنش‌ها را با استفاده از UML بصورت گرافیکی مدل سازی کنیم. زیان UML دارای چندین نوع دیاگرام تراکنشی است که رفتار یک سیستم را با مدل کردن نحوه تعامل شی‌ها با شی دیگری مدل سازی می‌کنند. تاکید دیاگرام ارتباطی بر مشارکت شی‌ها در همکاری‌ها است [نکته در نسخه‌های اولیه UML به دیاگرام‌های ارتباطی، دیاگرام‌های همکاری گفته می‌شود]. همانند دیاگرام



ارتباطی، دیاگرام توالی نمایشی از همکاری‌ها در میان شی‌ها است، اما با تاکید بر زمان ارسال پیغام‌ها مابین شی‌ها.

دیاگرام‌های ارتباطی

شکل ۷-۲۸ نمایشی از یک دیاگرام ارتباطی است که اجرای **BalanceInquiry** توسط **ATM** را مدل کرده است. شی‌های مدل شده در UML بصورت مستطیل‌های حاوی اسمی بفرم **نام کلاس: نام شی** نشان داده می‌شوند. در این مثال، که فقط یک شی از هر نوع باهم درگیر شده‌اند، ما نام شی را نادیده گرفته‌ایم و فقط یک کولن قبل از نام کلاس قرار داده‌ایم. [نکته: مشخص کردن نام هر شی در یک دیاگرام ارتباطی در زمان مدل کردن چندین شی از یک نوع توصیه شده است]. شی‌های ارتباطی با خطوط یک پارچه به هم متصل شده، و جهت پیغام‌های ارسالی مابین شی‌ها در امتداد این خطوط با فلش نشان داده می‌شوند. نام پیغام که در کنار فلش ظاهر می‌شود، نام یک عملیات (تابع عضو) متعلق به شی دریافت‌کننده است. می‌توانید در مورد نام همانند سرویسی فکر کنید که شی دریافت‌کننده آنرا برای شی ارسال‌کننده تدارک می‌بیند (سرویس گیرنده‌های خود).

شکل ۷-۲۸ | دیاگرام ارتباطی از اجرای پرس‌وجوی میزان موجودی توسط ATM.

فلش یکپارچه بکار رفته در شکل ۷-۲۸ نشان‌دهنده یک پیغام یا فرآخوانی همزمان در UML و فرآخوانی یک تابع در C++ است. این فلش بر این نکته دلالت دارد که جریان کنترل از سوی شی فرستنده (در اینجا **ATM**) به شی دریافت‌کننده (**BalanceInquiry**) است. از آنجا که این یک فرآخوانی همزمان است، امکان ارسال یک پیغام دیگر توسط شی ارسال‌کننده یا انجام کاری دیگری وجود ندارد تا اینکه شی دریافت‌کننده این پیغام را پردازش کرده و کنترل به شی فرستنده برگشت داده شود. فرستنده فقط در انتظار باقی می‌ماند. برای مثال در شکل ۷-۲۸ ATM مبادرت به فرآخوانی تابع عضو **execute** از **BalanceInquiry** می‌کند و تا زمانیکه **execute** کار خود را تمام نکرده و کنترل را به ATM برگشت ندارد نمی‌تواند پیغام دیگری ارسال کند. [نکته: اگر فرآخوانی از نوع غیرهمزمان یا اسنکرون باشد، اینحالت با یک فلش دوطرفه نشان داده می‌شود و دیگر شی ارسال‌کننده مجبور نبود تا در انتظار برگشت کنترل از سوی شی دریافت‌کننده باقی بماند و می‌تواند به ارسال پیغام‌های دیگری بلافاصله پس از فرآخوانی اسنکرون ادامه دهد. غالباً فرآخوانی‌های اسنکرون در C++ با استفاده از پلات‌فرم خاصی از کتابخانه‌های تدارک دیده شده توسط کامپایلر پیاده‌سازی می‌شوند. بررسی چنین تکنیک‌های خارج از

قلمرو این کتاب هستند.]

توالی پیغام‌ها در دیاگرام ارتباطی



در شکل ۷-۲۹ یک دیاگرام ارتباطی نشان داده شده است که تراکنش‌های مابین شی‌های موجود در سیستم را در زمانیکه یک شی از کلاس **BalanceInquiry** اجرا می‌شود را مدل کرده است. فرض می‌کنیم که صفت **accountNumber** شی حاوی شماره حساب کاربر جاری است. همکاری‌های موجود در شکل ۷-۲۹ پس از ارسال یک پیغام **execute** از سوی **ATM** به یک **BalanceInquiry** (یعنی تراکنش مدل شده در شکل ۷-۲۸) شروع می‌شوند.

شکل ۷-۲۹ | دیاگرام ارتباطی اجرای پس وجوی میزان موجودی.

اعداد قرار گرفته در سمت چپ نام یک پیغام بر ترتیب ارسال و گذر پیغام دلالت دارند. توالی پیغام‌ها در یک دیاگرام ارتباطی دارای ترتیب عددی از کوچکترین به سمت بزرگترین عدد است. در این دیاگرام، عدد گذاری پیغام با ۱ شروع و با پیغام ۳ خاتمه یافته است. ابتدا **BalanceInquiry** یک پیغام **getTotalBalance** به **BankDatabase** ارسال می‌کند (پیغام ۱)، سپس پیغام **getAvailableBalance** به **BankDatabase** ارسال می‌کند (پیغام ۲). در درون پرانتزهای قرار گرفته در مقابل نام پیغام، می‌توانیم یک لیست جدا شده با کاما از اسمی پارامترهای ارسالی به همراه پیغام قرار دهیم (آرگومان در فراخوانی یک تابع C++). در این مورد **BalanceInquiry** صفت **accountNumber** را به همراه پیغام خود به **BankDatabase** ارسال می‌کند تا نشان دهد اطلاعات کدام حساب باید بازیابی شود. از شکل ۶-۳۳ بخاطر دارید که عملیات **BankDatabase** به **getTotalBalance** و **getAvailableBalance** از کلاس **BankDatadase** هر یک مستلزم یک پارامتر برای شناسایی حساب هستند. سپس **BalanceInquiry** مبادرت به نمایش موجودی قابل برداشت و کل موجودی با ارسال یک پیغام **displayMessage** به **Screen** (پیغام ۳) به کاربر می‌کند که شامل یک پارامتر به نشانه پیغام قابل نمایش است. دقت کنید با وجود اینکه در شکل ۷-۲۹ دو پیغام از **BankDatabase** به یک حساب (Account) ارسال شده (پیغام‌های ۱ و ۲)، بایستی **BankDatabase** یک پیغام **getAvailableBalance** و یک پیغام **getTotalBalance** به حساب کاربر ارسال کند. به چینی پیغام‌های که در درون پیغام دیگری ارسال می‌شوند، پیغام‌های تودرتو یا آشیانه‌ای گفته می‌شود. توصیه UML بر استفاده از شماره گذاری اعشاری برای نشان دادن پیغام‌های تودرتو است. برای مثال، پیغام ۱.۱ اولین پیغام تودرتو در پیغام ۱ است، **BankDatabase** یک پیغام **getAvailableBalance** در حال پردازش پیغام از همان نام است ارسال کرده است. [نکته: اگر **BankDatabase** نیازمند ارسال پیغام تودرتوی دومی باشد در حالیکه پیغام ۱ پردازش می‌شود، پیغام دوم بصورت ۱.۲ شماره گذاری می‌شود.] امکان دارد یک پیغام زمانی ارسال شود که کلیه پیغام‌های تودرتو از پیغام قبلي ارسال شده باشند. برای مثال، **BalanceInquiry** پیغام ۳ را فقط پس از پیغام ۲ و ۱ ارسال می‌کند.



طرح شماره‌گذاری تودرتوی بکار رفته در دیاگرام‌های ارتباطی سبب افزایش وضوح نحوه و ترتیب ارسال هر پیغام می‌شود. برای مثال، اگر پیغام‌های بکار رفته در شکل ۷-۲۹ را با استفاده از طرح عدد‌گذاری ساده (همانند ۱, ۲, ۳, ۴, ۵) شماره‌گذاری کنیم، احتمال دارد شخصی که به دیاگرام نگاه می‌کند قادر به تعیین اینکه **BankDatabase** مبادرت به ارسال پیغام **getAvailableBalance** (پیغام ۱.۱) به یک **Account** در زمانیکه **BankDatabase** در حال پردازش پیغام ۱ است یا پس از کامل شدن پردازش پیغام ۱ نباشد. اما شماره‌گذاری تودرتو کمک می‌کند که دومین پیغام **getAvailableBalance** (پیغام ۱.۱) به یک **Account** در درون اولین پیغام **getAvailableBalance** (پیغام ۱) توسط **BankDatabase** ارسال شده است.

دیاگرام‌های توالی

تاكيد دیاگرام‌های ارتباطی بر همکاری‌های موجود است، اما مدل‌سازی زمان در آنها چندان قوی نیست. دیاگرام توالی در مدل کردن زمانبندی همکاری‌ها از وضوح بیشتری برخوردار است. در شکل ۷-۳۰ نمایشی از یک دیاگرام توالی که تراکنش‌های رخ داده در زمان برداشت پول **Withdrawal** را مدل کرده، آورده شده است. خطوط خط چین که از مستطیل یک شی به سمت پایین امتداد یافته‌اند، نشاندهنده خط عمر و زمان آن شی هستند. ترتیب زمانی وقوع عمل یا فرآیندی در طول عمر یک شی از بالا به سمت پایین است، عملی که در مراتب بالا قرار دارد قبل از عملی که در پایین تراز آن جای گرفته اتفاق می‌افتد.

۷-۳۰ | دیاگرام توالی مدل کننده عملیات برداشت پول (*Withdrawal*)

ارسال پیغام در دیاگرام‌های توالی همانند ارسال پیغام در دیاگرام‌های ارتباطی است. فلش یک پارچه بسط یافته از سمت شی ارسال کننده به سمت شی دریافت کننده نشاندهنده یک پیغام مابین دو شی است. نوک فلش به سمت یک فعالیت در خط عمر شی دریافت کننده است. یک فعالیت به شکل یک مستطیل نازک عمودی نشان داده می‌شود، که نشاندهنده یک شی در حال اجرا است. زمانیکه یک شی کنترل را باز می‌گرداند، پیغام برگشتی توسط یک خط چین با فلش از سوی شی فعال که کنترل را به فرستنده پیغام بازمی‌گردد نشان داده می‌شود. برای حذف موارد سردرگم کننده، فلش‌های پیغام برگشتی را حذف کرده‌ایم و UML به منظور افزایش خوانایی دیاگرام چنین اجزاهای را می‌دهد. همانند دیاگرام‌های ارتباطی، دیاگرام توالی می‌تواند نشاندهنده پارامترهای پیغام در میان پرانتزها پس از نام پیغام باشند.

توالی پیغام در شکل ۷-۳۰ زمانی شروع می‌شود که یک **Withdrawal** به کاربر اعلان می‌کند تا میزان پول مورد نظر را انتخاب کند و اینکار با ارسال پیغام **displayMessage** به **Screen** صورت می‌گیرد. سپس **Withdrawal** پیغام **getInput** را به **Keypad** می‌فرستد تا ورودی کاربر را دریافت کند. در شکل ۵-۲۸ منطق اعمال شده در فعالیت برداشت پول مدل سازی شده است و از این‌رو این منطق را در این



دیاگرام توالی نشان نداده‌ایم و بجای آن بهترین سناریو را که در آن موجودی حساب کاربر بیشتر یا برابر میزان برداشتی است و پرداخت کننده پول، حاوی مقدار کافی پول نقد برای برآورده کردن تقاضا است را مدل کرده‌ایم.

پس از بدست آوردن میزان پول برای برداشت، **Withdrawal** پیغام **getAvailableBalance** را به **BankDatabase** ارسال می‌کند که آن هم در ادامه پیغام **getAvailableBalance** را به حساب کاربر (**Account**) ارسال می‌نماید. فرض کنید که حساب کاربر به میزان کافی پول دارد و می‌تواند تراکنش درخواست شده را انجام دهد، سپس **Withdrawal** پیغام **isSufficientCashAvailable** را به **CashDispenser** ارسال می‌نماید. فرض کنید که پول نقد به میزان کافی در اختیار است، **Withdrawal** موجودی کاربر را از حساب آن کم می‌کند (هم از **availableBalance** و هم از **totalBalance**) با ارسال پیغام **debit** به **BankDatabase**. پایگاه داده با ارسال پیغام **debit** به حساب کاربر (**Account**) از خود واکنش نشان می‌دهد. سرانجام **Withdrawal** پیغام **dispenseCash** را به **CashDispenser** و پیغام **displayMessage** را به **Screen** می‌کند تا به کاربر اعلان کند، پول خود را از ماشین بردارد.

تمرینات خودآزمایی مبحث مهندسی نرم‌افزار

۱-۷ متشکل از ارسال یک پیغام از سوی شی از یک کلاس به شی از کلاس دیگر است.

- (a) وابستگی
- (b) اجتماع
- (c) همکاری
- (d) ترکیب

۲-۲ کدام فرم از دیاگرام تراکنشی تاکید برای نوع همکاری دارد؟ و کدامیک تاکید بر زمان همکاری؟

۲-۳ یک دیاگرام توالی ایجاد کنید که تراکنش‌های موجود مابین در زمان اجرای موفقیت آمیزی سپرده‌گذاری (Deposit) را مدل‌سازی کند.

پاسخ خودآزمایی مبحث مهندسی نرم‌افزار

۷-۱

۷-۲ دیاگرام‌های ارتباطی بر نوع همکاری و دیاگرام‌های توالی بر زمان رخ دادن همکاری‌ها تاکید دارند.

شکل ۷-۳۱ دیاگرام توالی که اجرای سپرده‌گذاری (Deposit) را مدل کرده است.

خودآزمایی

۷-۱ جاهای خالی را در عبارات زیر با کلمات مناسب پر کنید:

(a) مقادیر لیست‌ها و جداول می‌توانند در و ذخیره شوند.

(b) عناصر یک آرایه دارای و یکسان هستند.

(c) عددی که به یک عنصر آرایه اشاره می‌کند، نام دارد.



(d) باید از یک در اعلان سایز آرایه استفاده کرد، چراکه با اینکار برنامه پایدارتر می‌شود.

(e) فرآیند قراردادن عناصر یک آرایه در یک ترتیب، آرایه نامیده می‌شود.

(f) عمل تعیین اینکه آیا آرایه‌ای حاوی یک مقدار مشخص است، نامیده می‌شود.

(g) به آرایه‌های که دو یا بیش از دو ساختار دارند، آرایه‌های، گفته می‌شود.

۷-۲ کدامیک از عبارات زیر صحیح و کدامیک اشتباه است. اگر عبارتی اشتباه است علت آنرا توضیح دهید.

(a) یک آرایه می‌تواند مقادیری از نوع‌های مختلف در خود ذخیره سازد.

(b) ساختار یک آرایه بایستی از نوع داده float باشد.

(c) اگر لیست مقداردهی کننده اولیه کمتر از تعداد عناصر در آرایه باشد، مابقی عناصر با آخرین مقدار در لیست مقداردهی اولیه، مقدار دریافت خواهند کرد.

(d) اگر لیست مقداردهی کننده اولیه حاوی مقادیری بیش از تعداد عناصر موجود در آرایه باشد، خطای خواهد داد.

(e) یک عنصر مجزا در آرایه که به تابعی ارسال شده و تغییر یافته پس از کامل شدن وظیفه تابع، حاوی مقدار تغییر یافته خواهد بود.

۷-۳ عبارتی بنویسید که موارد خواسته شده در زیر را بآورده سازد (برای آرایه‌ای بنام fractions):

(a) یک متغیر ثابت نام arraySize تعریف و با 10 مقداردهی کنید.

(b) آرایه‌ای با عناصر arraySize از نوع double اعلان، و آنها را با صفر مقداردهی کنید.

(c) چهارمین عنصر در آرایه.

(d) مراجعه به عنصر 4 آرایه.

(e) تخصیص مقدار 1.667 به عنصر 9 آرایه.

(f) تخصیص مقدار 3.333 به هفتمین عنصر آرایه.

(g) چاپ عناصر 6 و 9 آرایه با دقت دو رقم در سمت راست نقطه اعشار، و نمایش خروجی.

(h) چاپ تمام عناصر آرایه توسط عبارت for. متغیر i را عنوان متغیر کنترلی حلقه for تعریف کرده خروجی را بنمایش در آورید.

۷-۴ به سوالات زیر با توجه به آرایه‌ای بنام table پاسخ دهید:

(a) اعلان آرایه از نوع صحیح با 3 سطر و 3 ستون. فرض کنید که متغیر ثابت arraySize با مقدار 3 تعریف شده.

(b) برنامه‌ای بنویسید که تا مقادیر هر عنصر آرایه table را در فرمت جدولی 3 سطر و 3 ستون چاپ کند. با فرض اینکه آرایه بصورت زیر مقداردهی اولیه شده است:

```
int table[arraySize][arraySize]={{1,8},{2,4,6},{5}};
```

و متغیرهای کنترلی i, j تعریف شده باشند. خروجی را بنمایش در آورید.

۷-۵ خطای موجود در عبارات زیر را یافته و اصلاح کنید.

a) # include <iostream>;



b) `arraySize=10; // arraySize was declared const`

ب) فرض اینکه `b[10]={0};`

`for (int i=0; i<=10; i++)`

`b[i]=1;`

د) بافرض `a[2][2]={ {1,2},{3,4} };`

`a[i,j]=5;`

پاسخ خودآزمایی

۷-۱ (a) آرایه‌ها، بردارها. (b) نام، نوع. (c) شاخص یا ساب‌اسکریپت. (d) متغیر ثابت. (e) مرتب‌سازی. (f) جستجو. (g) دو بعدی.

`IndexOutOfRangeException (j const(i))` نامنظم.

۷-۲ (a) اشتباه. یک آرایه فقط قادر به نگهداری مقادیر از یک نوع است. (b) اشتباه. شاخص آرایه باید یک مقدار یا عبارت صحیح باشد. (c) اشتباه. عناصر باقیمانده با صفر مقداردهی اولیه می‌شوند. (d) صحیح. (e) اشتباه. عناصر مجزای آرایه به روش مقدار ارسال می‌شوند.

۷-۳

```
a) const int arraySize = 10;
b) double fractions[arraySize] = {0,0};
c) fractions[3]
d) fractions[4]
e) fractions[9] = 1.667'
f) fractions[6] = 3.333;
g) cout << fixed<<setprecision (2);
   cout << fractions[6] << ' ' << fractions[9] << endl;
3.33 1.67
```

```
h) for (int i=0; i< arraySize;i++)
   cout << " fractions["<<i<<"] = << fractions[i] << endl;
```

خروجی:

```
Fractions[0]=0.0
Fractions[1]=0.0
Fractions[2]=0.0
Fractions[3]=0.0
Fractions[4]=0.0
Fractions[5]=0.0
```



```
Fractions[6]=3.333
Fractions[7]=0.0
Fractions[8]=0.0
Fractions[9]=1.667
```

۷-۴

```
a) int table[arraySize][arraySize];
b) 4
c) for (i=0; i<< arraySize; i++)
    for (j=0; j< arraySize; j++)
        table[i][j]=i+j;
d) cout <<"[0] [1] [2]" <<endl;
for (int i=0; i< arraySize; i++){
    cout << '[' <<i<<"]";
    for (int j=0; j< arraySize; j++)
        cout <<setw(3)<<table[i][0]<< " ";
    cout << endl;
```

خروجی:

```
[0] [1] [2]
[0] 1 8 0
[1] 2 4 6
[2] 5 0 0
```

۷-۵

(a) خطأ: سیملوکن در انتهای #include قرار گرفته است.

اصلاح: حذف سیملوکن.

(b) خطأ: تخصیص مقداری به متغیر ثابت با استفاده از عبارت تخصیصی.

اصلاح: مقداردهی اولیه متغیر ثابت در .const int arraySize.

(c) خطأ: مراجعه به عنصر آرایه خارج از مرزهای آرایه (b)[10].

اصلاح: تغییر مقدار نهایی متغیر کنترلی به .9

(d) خطأ: شاخص عملکرد درستی ندارد.

اصلاح: تغییر عبارت به .a[1][1]=5;

تمرینات

۷-۶ جاهای خالی را با عبارت مناسب پر کنید.



- (a) اسمی چهار عنصر آرایه `p` (یعنی `int p[1];`) عبارتند از ، ، و
- (b) نامگذاری آرایه، نوع آرایه مشخص کردن تعداد عناصر و در آرایه، آرایه نامیده می‌شود.
- (c) بطور قرار دادی، اولین شاخص در آرایه دو بعدی بعنوان عنصر و دومین شاخص بعنوان عنصر شناخته می‌شود.
- (d) یک آرایه `m` در `n` حاوی سطر، ستون و عنصر است.
- (e) نام عنصر در سطر 3 و ستون 5 آرایه `d` عبارت است از
- 7-7 تعیین کنید کدامیک از موارد زیر صحیح و کدامیک اشتباه است.
- (a) برای مراجعه به یک موقعیت خاص یا عنصری در درون آرایه، نام آرایه و مقدار دقیق آن عنصر را مشخص می‌کیم.
- (b) با اعلان آرایه فضا برای آرایه رزرو می‌شود.
- (c) برای اینکه 100 مکان برای آرایه صحیحی بنام `p` رزرو شود؛ باید برنامه نویس در اعلان بنویسد؛ `[100]`.
- (d) برای مقداردهی اولیه یک آرایه 15 عنصری با صفر، باید از یک عبارت `for` استفاده کرد.
- (e) برای محاسبه مجموع عناصر یک آرایه دو بعدی باید از عبارت `for` تودرتو استفاده کرد.
- 7-8 عباراتی در `c++` بنویسید که موارد خواسته در زیر را برابر آورده سازند:
- (a) نمایش مقدار عنصر 6 از آرایه کاراکتری `f`.
- (b) وارد کردن مقداری به عنصر 4، آرایه یک بعدی اعشار، بنام `b`.
- (c) مقداردهی اولیه پنج عنصر آرایه یک بعدی `g` با 8.
- (d) محاسبه مجموع و چاپ عناصر آرایه `c` که 100 عنصر دارد.
- (e) کپی کردن آرایه `a` به اولین بخش آرایه `b`. با فرض `a[11], b[34]` .
- (f) کوچکترین و بزرگترین مقدار موجود در آرایه `w` با 99 عنصر را یافته و چاپ کنید.
- 7-9 با در نظر گرفتن آرایه 2 در 3 بنام `t` و از نوع صحیح موارد خواسته شده زیر را برابر آورده سازید.
- (a) اعلانی برای `t` بنویسید.
- (b) آرایه `t` دارای چند ستون است؟
- (c) آرایه `t` دارای چند سطر است؟
- (d) آرایه `t` چند عنصر دارد؟
- (e) اسمی تمام عناصر موجود در سطر 1 از `t` را بنویسید.
- (f) اسمی تمام عناصر موجود در ستون 2 از `t` را بنویسید.
- (g) عباراتی بنویسید که عنصر قرار گرفته در سطر 1 و ستون 2 را با صفر تنظیم کند.
- (h) دنباله ای از دستورات بنویسید که هر عنصر از `t` را با صفر مقداردهی اولیه کند. از حلقه استفاده نکنید.
- (i) یک عبارت `for` تودرتو بنویسید که هر عنصر `t` را با صفر مقداردهی کند.



ز) عبارتی بنویسید که مقادیری برای عناصر a از ترمیナル دریافت کنند.

ک) دنباله‌ای از عبارات بنویسید که کوچکترین مقدار در آرایه a را یافته و چاپ کنند.

ل) عبارتی بنویسید که عناصر موجود در سطر صفر a را بنمایش در آورد.

م) عبارتی بنویسید که مجموع عناصر در ستون سوم a را بنمایش در آورد.

ن) عبارتی بنویسید که آرایه a را بصورت عادی در فرمت جدولی چاپ کنند.

۷-۱۰ با استفاده از یک آرایه یک بعدی مسئله زیر را حل کنید: یک شرکت به فروشنده‌گان خود بر حسب کمیسیون مبالغی پرداخت می‌کند. هر فروشنده برای هر هفته 200 دلار به همراه 9 درصد از فروش خود در آن هفته دریافت می‌کند. برای مثال اگر فروشنده‌ای در یک هفته 5000 دلار فروش داشته باشد، مبلغ 200 دلار به همراه 9 درصد از 500 دلار یعنی 650 دلار یافت خواهد کرد. برنامه‌ای بنویسید (با استفاده از یک آرایه از شمارنده‌ها) که تعداد فروشنده‌گان را بر حسب محدوده‌های مشخص شده تعیین نماید:

\$200 - \$299 (a)

\$300 - \$399 (b)

\$400 - \$499 (c)

\$500 - \$599 (d)

\$600 - \$699 (e)

\$700 - \$799 (f)

\$800 - \$899 (g)

\$900 - \$999 (h)

و بالاتر. (i) \$1000

۷-۱۱ در الگوریتم مرتب‌سازی حبابی (bubble sort)، مقادیر کوچکتر همانند حباب خود را به بالای آرایه می‌رسانند، همانند حرکت حباب در درون آب، در حالیکه مقادیر بزرگتر به پایین آرایه فرستاده می‌شوند. در مرتب‌سازی حبابی، چندین بار کل آرایه پیمایش می‌شود. در هر بار، جفت عناصر با هم مقایسه می‌شوند. اگر جفتی در ترتیب صعودی قرار داشته باشند (یا مقادیر یکسان باشند)، مقادیر در سر جای خود رها می‌شوند. اگر جفتی در ترتیب نزولی قرار داشته باشند، این مقادیر جای خود را در آرایه عوض می‌کنند. برنامه‌ای بنویسید که آرایه‌ای با 10 مقدار صحیح، را به روش حبابی، مرتب کنند.

۷-۱۲ برنامه مرتب‌سازی حبابی ارائه شده، در آرایه‌های بزرگ قادر کارایی لازم است. با بکار بردن اصلاحات ساده ارائه شده در زیر کارایی این نوع از مرتب‌سازی را افزایش دهید:

(a) پس از اولین ارسال (گذار)، مطمئن هستیم بزرگترین عدد در بالاترین محل آرایه قرار دارد. پس از دومین گذار دو عدد بزرگ در آن مکان قرار دارند. و به همین ترتیب بجای انجام 9 مقایسه در هر بار گذار، مرتب‌سازی را طوری اصلاح کنید که در دومین گذار هشت مقایسه، در سومین گذار هفت مقایسه صورت گیرد و اینکار تا انتها انجام شود.



(b) داده‌های موجود در آرایه ممکن است بصورت مرتب قرار گرفته باشند یا تقریباً دارای حالت مرتب شده باشند. پس چرا باید 9 گذار انجام گیرد که کمترین تأثیر را بر روی مرتب‌سازی اعمال می‌کند. مرتب‌سازی حبای را به نحوی اصلاح کنید تا در پایان هر گذار تست کند که آیا عمل جابجایی صورت گرفته است یا خیر. اگر جابجایی صورت نگرفته باشد، پس داده‌ها در آرایه بصورت مرتب قرار گرفته‌اند پس برنامه باید خاتمه پذیرد. اگر جابجایی صورت گرفته باشد نیاز به یک گذار یا بیشتر داریم.

۷-۱۳ عباراتی بنویسید که موارد خواسته شده در زیر را بر روی یک آرایه تک بعدی اعمال کنند:

(a) مقداردهی اولیه 10 عنصر آرایه **counts** با صفر.

(b) افزودن 1 به هر 15 عنصر آرایه **.bonus**.

(c) حواندن 12 مقدار برای آرایه **monthlyTemperatures** از نوع **double** و از طریق صفحه کلید.

(d) چاپ 5 مقدار از آرایه صحیح بنام **bestScores** در فرمت جدولی.

۷-۱۴ خطای خطاهای موجود در عبارات زیر را پیدا کنید:

(a) با فرض اینکه: `char str[5];`

`cin>>str; //user types "hello"`

(b) با فرض اینکه: `int a[3];`

`cout<<a[1]<<" "<<a[2]<<" "<<a[3]<<endl;`

`double f[3]={1.1,10.01,100,001,1000,0001};`

`d[1,9]=2.345;`

(c)

(d) با فرض اینکه: `double d[2][10];`

۷-۱۵ با استفاده از یک آرایه یک بعدی مسئله زیر را حل کنید: برنامه 20 عدد دریافت می‌کند که هر عدد باید مابین 100 و 10 باشد. اگر هر عدد دریافتی، تکرار نشده باشد، آنرا به نمایش درآورد. سعی کنید از کوچکترین آرایه استفاده کنید.

۷-۱۶ برنامه‌ای بنویسید که پرتاب دو طاس را شبیه‌سازی کند. در این برنامه باید از تابع **rand** در پرتاب طاس اول و مجدداً در پرتاب طاس دوم استفاده شود. سپس مجموع دو مقدار محاسبه شود. [نکته: هر طاس می‌تواند یک مقدار صحیح از 1 تا 6 را نشان دهد، از این‌رو مجموع دو مقدار مابین 2 تا 12 متغیر خواهد بود] در شکل ۷-۳۲، ترکیبی از 36 حالت ممکنه در پرتاب دو طاس نشان داده شده است. برنامه شما باید دو طاس را به تعداد 36000 بار پرتاب کند. از یک آرایه تک بعدی استفاده کنید تا تعداد دفعات ممکنه از مجموع هر پرتاب را نشان دهد. نتایج را در فرمت جدولی چاپ کنید.

شکل ۷-۳۲ ۳۶ حالت ممکنه در پرتاب دو طاس.

۷-۱۸ برنامه زیر چه کاری انجام می‌دهد؟

```
// Ex. 7.18: ex07_18.cpp
// What does this program do?
#include <iostream>
using std::cout;
using std::endl;
int whatIsThis( int [], int ); // function prototype
```



```

int main()
{
    const int arraySize = 10;
    int a[ arraySize ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

    int result = whatIsThis( a, arraySize );

    cout << "Result is " << result << endl;

    return 0; // indicates successful termination
} // end main

// What does this function do?
int whatIsThis( int b[], int size )
{
    if ( size == 1 ) // base case
        return b[ 0 ];
    else // recursive step
        return b[ size - 1 ] + whatIsThis( b, size - 1 );
} // end function whatIsThis

```

۷-۱۹ برنامه شکل ۱۱-۶ را برای 6000 بار بازی craps تغییر دهید.

۷-۲۰ خط هوایی کوچکی مبادرت به خرید یک کامپیوتر برای سیستم رزرواسیون اتوماتیک خود کرده است. از شما خواسته شده تا این سیستم جدید را برنامه‌نویسی کنید. برنامه‌ای خواهید نوشت که صندلی‌های هر پرواز را تخصیص دهد (ظرفیت: 10 صندلی).

برنامه شما باید منوهای زیر را در اختیار کاربر قرار دهد:

Please type 1 for "first class"

و

Please type 2 for "Economy"

اگر کاربر، 1 را تایپ کند، برنامه یک صندلی در بخش First class (صندلی‌های ۱-۵) به وی تخصیص خواهد داد. اگر کاربر، 2 را تایپ کند، برنامه یک صندلی در بخش Economy (صندلی‌های ۶-۱۰) به وی تخصیص خواهد داد. برنامه باید اطلاعات صندلی از جمله شماره صندلی و نوع کلاس آن را چاپ کند.

برای نمایش صندلی‌ها در هوایپما از یک آرایه تک بعدی استفاده کنید. تمام عناصر آرایه را در ابتدای کار با صفر مقداردهی اولیه کنید تا نشان داده شود که همه صندلی‌ها خالی هستند. همانطوری که هر صندلی تخصیص داده می‌شود، موقعیت آن صندلی در آرایه با ۱ تنظیم شود تا مشخص گردد که دیگر این صندلی رانی توان به دیگری تخصیص داد. زمانیکه بخش First class پر شد، برنامه باید از کاربر سوال کند که آیا مایل به پذیرش صندلی در بخش Economy است یا خیر (بر عکس اینحالات را هم انجام دهد). اگر پاسخ مثبت باشد، صندلی مربوطه تخصیص داده شود و در غیر اینصورت پیغام "Next flight leaves in 3 hours" را چاپ کند.

۷-۲۱ برنامه زیر چه کاری انجام می‌دهد؟

```

// Ex. 7.21: ex07_21.cpp
// What does this program do?
#include <iostream>
using std::cout;
using std::endl;

void someFunction( int [], int, int ); // function prototype

int main()
{

```



```
const int arraySize = 10;
int a[ arraySize ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

cout << "The values in the array are:" << endl;
someFunction( a, 0, arraySize );
cout << endl;

return 0; // indicates successful termination
} // end main

// What does this function do?
void someFunction( int b[], int current, int size )
{
    if ( current < size )
    {
        someFunction( b, current + 1, size );
        cout << b[ current ] << " ";
    }
} // end function someFunction
```

۷-۲۲ با استفاده از یک آرایه دو بعدی، مسئله زیر را حل کنید. شرکتی دارای چهار فروشنده است (۱ تا ۴) که مسئول فروش پنج محصول مختلف هستند (۱ تا ۵) در پایان یک روز، هر فروشنده صورت فروش هر نوع محصول را ارائه می کند.

هر صورت حاوی اطلاعات زیر است:

- (a) شماره فروشنده
- (b) شماره محصول
- (c) مبلغ کل از آن محصول در روز

بنابراین، هر فروشنده صورت فروشی مابین ۰ تا ۵ را ارائه می کند. فرض کنید اطلاعات تمام صورت فروش‌های ماه قبل در دسترس است. برنامه‌ای بنویسید که تمام این اطلاعات را خوانده و کل فروش هر فروشنده و محصول فروخته شده را بطور خلاصه بنمایش در آورد. کل فروش باید در آرایه دو بعدی sales ذخیره شده باشد. پس از پردازش کل اطلاعات ماه قبل، نتایج در فرمت جدولی که هر ستون نشانده‌نده یک فروشنده خاص است و هر سطر نشانده‌نده یک محصول ویژه، چاپ گردد.

۷-۲۳ زیان logo که از محبوبیت خاصی در مدارس ابتدایی برخوردار است، از مفهوم لاکپشت گرافیکی استفاده کرده است. تصور کنید که یک لاکپشت مکانیکی در فضایی حرکت می کند که در کنترل یک برنامه C++ است. لاکپشت مدادی را به یکی از دو جهت، بالا یا پایین حرکت می دهد. زمانیکه مداد به طرف پایین کشیده می شود، لاکپشت اشکالی را ترسیم می کند. با حرکت مداد به سمت بالا، لاکپشت آزادانه بدون ترسیم چیزی حرکت می کند. در این مسئله، می خواهیم حرکت لاکپشت را شبیه‌سازی کرده و صفحه طراحی کامپیوتری شده‌ای را هم برای آن ایجاد کنید.

از آرایه 20 در 20 با میان floor استفاده کنید که در ابتدا با صفر مقداردهی اولیه شده است. دستورات از آرایه‌ای خوانده شود که حاوی آنها است. همیشه مسیر و موقعیت جاری لاکپشت را خواه مداد بالا باشد یا پایین، داشته باشید. فرض کنید لاکپشت از موقعیت (0,0) با مداد بالا حرکت خود را آغاز می کند. مجموعه دستورات لاکپشت که باید برنامه مبادرت به پردازش آنها کند در جدول ۷-۳۳ آورده شده‌اند.



فرض کنید که لاکپشت جای در نزدیکی مرکز صفحه قرار دارد. برنامه زیر باید یک مریع 12 در 12 ترسیم و چاپ کرده و بالا رفتن مداد به کار پایان دهد:

2
5,12
3
5,12
3
5,12
3
5,12
1
6
9

همانطوری که لاکپشت با مداد پایین حرکت می‌کند، عناصر مقتضی آرایه **floor** با 1 تنظیم می‌شوند. زمانیکه دستور 6 اعمال می‌شود (چاپ)، هر جا که 1 در آرایه وجود داشته باشد، یک کاراکتر ستاره یا کاراکتر دلخواه بنمایش در آید. هر جا که صفر در آرایه وجود داشته باشد، یک جای خالی بنمایش در آید. برنامه‌ای بنویسید که لاک گرافیکی را با قابلیت‌های فوق پیاده‌سازی کند.

دستور	مفهوم دستور
1	مداد بالا
2	مداد پایین
3	گردش به راست
4	گردش به چپ
5.10	حرکت به میزان space 10 (یا هر عددی به جای 10)
6	چاپ آرایه 20 در 20
9	پایان داده (مراقبتی)

شکل ۷-۳۳ | دستورات لاکپشت گرافیکی.

۷-۲۴ یکی از معماهای جالب در صفحه شطرنج، مسئله حرکت مهره اسب است. مسئله این است: آیا می‌توان مهره اسب را در صفحه خالی شطرنج به نحوی حرکت داد که کل 64 خانه صفحه را فقط یکبار لمس کرده باشد؟ در این تمرین به بررسی این مسئله می‌پردازیم. مهره اسب حرکتی بفرم L دارد. از اینرو، از یک خانه در میانه صفحه خالی شطرنج، این مهره می‌تواند هشت حرکت مختلف انجام دهد که در شکل ۷-۳۴ نشان داده شده‌اند (شماره گذاری شده از صفر تا هفت).



شکل ۷-۳۴ | هشت حرکت مختلف اسب.

(a) یک صفحه شطرنج ۸ در ۸ بر روی کاغذ ترسیم کرده و مبادرت به حرکت مهره اسب بر روی آن کنید. در اولین خانه که اسب در آن فرود آمده عدد ۱، در دومین خانه ۲، در سومین خانه عدد ۳، و الى آخر، قرار دهید. قبل از شروع حرکت، میزان و تعداد حرکت را تخمین بزنید. بخاطر داشته باشید که حرکت کامل برابر ۶۴ حرکت است. تا کجا توانسته اید پیش بروید؟ آیا به حدثی که زده اید، نزدیک هستید؟

(b) اکنون اجازه دهید، تا برنامه‌ای ایجاد کنیم که مهره اسب را در صفحات شطرنج به حرکت در آورد. صفحه شطرنج توسط یک آرایه دو بعدی ۸ در ۸ بنام board عرضه می‌شود. هر کدامیک از خانه‌های شطرنج با صفر مقداردهی اولیه می‌شوند. هر هشت حرکت ممکنه را در جهات افقی و عمودی بیان می‌کنیم. برای مثال، حرکت از نوع صفر، که در شکل ۷-۳۴ نشان داده شده است. مشکل از حرکت دو خانه افقی به سمت راست و یک خانه عمودی به سمت بالا می‌باشد. حرکت ۲ مشکل از حرکت یک خانه افقی به چپ و دو خانه عمودی به بالا می‌باشد. حرکت افقی به چپ و حرکت عمودی به بالا دلالت بر مقادیر منفی دارند. این هشت حرکت را می‌توان توسط دو آرایه یک بعدی، **vertical**, **horizontal** توصیف کرد، همانند:

```
horizontal[0] = 2
horizontal[1] = 1
horizontal[2] = -1
horizontal[3] = -2
horizontal[4] = -2
horizontal[5] = -1
horizontal[6] = 1
horizontal[7] = 2
```

```
vertical[0] = -1
vertical[1] = -2
vertical[2] = -2
vertical[3] = -1
vertical[4] = 1
vertical[5] = 2
vertical[6] = 2
vertical[7] = 1
```

متغیرهای **currentColumn**, **currentRow** می‌توانند نشاندهنده موقعیت جاری سطر و ستون اسب باشند. برای داشتن حرکتی از نوع **moveNumber** که در آن **moveNumber** عددی مابین ۰ و ۷ است، برنامه شما از عبارت زیر استفاده کند.

```
currentRow += vertical [moveNumber];
currentColumn += horizontal [moveNumber];
```

از یک شمارنده که از ۱ تا ۶۴ در حال شمارش است، استفاده کنید. آخرین شماره را که اسب در آن مربع فرود آمده است را ثبت نمایید. بخاطر داشته باشید که هر حرکتی را تست کنید تا متوجه شوید که آیا اسب قبلاً در آن خانه حضور داشته است یا خیر، همچنین تست کنید که اسب از صفحه شطرنج خارج نشود. اکنون برنامه‌ای بنویسید



که مهره اسب را در صفحه شطرنج به حرکت در آورد. برنامه را اجرا کنید. اسب چند حرکت انجام داده است؟
 (c) پس از نوشتن و اجرای برنامه حرکت مهره اسب، محتملأ دید ارزشمندی بدست آورده‌اید. ما از این بینش برای توسعه یک استراتژی یا روش غیرمستدل در حرکت مهره اسب، استفاده خواهیم کرد. استراتژی تضمین کننده موقفيت نیست، اما اگر این استراتژی بدقت توسعه یافته باشد، شناس موقفيت نیز افزایش خواهد یافت. شاید متوجه شده باشید که خانه‌های خارجی به نسبت خانه‌های نزدیک به مرکز صفحه شطرنج، پر دردسرتر هستند. در واقع، پر رحمت‌ترین یا غیر قابل دسترس‌ترین، خانه‌ها در چهار گوشه قرار دارند.

چنین بینشی به شما پیشنهاد می‌دهد که ابتدا مهره اسب را به خانه‌های پر دردسر حرکت داده و سپس به سراغ خانه‌های آسانتر بروید، از اینرو زمانیکه صفحه شطرنج در انتهای حرکات فشرده می‌شود، شناس زیادی برای موقفيت بدست می‌آید.

می‌توانید با طبقه‌بندی کردن هر خانه مطابق با نحوه دسترسی به آن و سپس حرکت اسب به خانه‌ای (البته به شکل L) که تقریباً غیر قابل دسترسی است یک "استراتژی دسترسی" طراحی کنید. از یک آرایه دو بعدی بنام accessibility با اعدادی استفاده می‌کنیم که دلالت بر تعداد دفعات دسترسی هر خانه را در عمل نشان می‌دهند. در یک صفحه خالی شطرنج، هر خانه مرکزی با ۸ هر خانه گوشه با ۲ و سایر خانه‌ها با اعداد دسترسی ۳، ۴ یا ۶ درجه‌بندی شده‌اند، همانند:

2	3	4	4	4	4	3	2
3	4	6	6	6	6	4	3
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
3	4	6	6	6	6	4	3
2	3	4	4	4	4	3	2

حال نسخه‌ای از برنامه حرکت مهره اسب را بنویسید که از استراتژی دسترسی در آن استفاده شده باشد.
 هر زمان، مهره اسب باید به خانه‌ای با پایین‌ترین عدد دسترسی منتقل شود. بنابر این حرکت را می‌توان از هر یک از چهار گوشه صفحه آغاز کرد. این نسخه از برنامه را اجرا کنید. آیا حرکت کامل را انجام داده‌اید؟ اکنون برنامه را برای 64 حرکت از یکی از گوشه‌ها آغاز کنید.

(d) نسخه‌ای از برنامه حرکت مهره اسب را بنویسید که در زمان مواجه شدن مابین دو یا چند خانه، تصمیم بگیرد که کدام خانه انتخاب شود، با توجه به خانه‌های که از آن محل در دسترس هستند.

۷-۲۴ در تمرین ۷-۲۵ راه حلی برای مسئله حرکت مهره اسب در شطرنج ارائه کردیم. از روش "استراتژی دسترسی" استفاده کردیم که از کارایی و قابلیت مناسبی برخوردار است.

همانطوری که هر روز بر قدرت کامپیوترها افزوده می‌شود، ما هم قادر می‌شویم تا مسائل پیچیده‌تر را با توجه به قدرت کامپیوتر و الگوریتم‌های نه چندان حرffe‌ای تر حل کنیم، که این روش "brute force" نامیده می‌شود.
 (a) با استفاده از تولید عدد تصادفی اقدام به حرکت دادن تصادفی مهره اسب بر روی صفحه شطرنج کنید. برنامه شما



آرایه‌ها و برد ارها

- باید یک تور انجام داده و صفحه پایانی را چاپ کند. مهره تا به کجا حرکت کرده است؟
- (b) به احتمال زیاد، برنامه قبلی تور کوتاهتری را تولید می‌کرد. اکنون برنامه خود را برای انجام 1000 تور تغییر دهد.
- از یک آرایه تک بعدی برای حفظ تعداد تورها در هر مسیر استفاده کنید. پس از اینکه برنامه مبادرت به انجام 1000 تور کرد، بایستی این اطلاعات را بصورت مرتب در یک فرمت جدولی چاپ کند. بهترین نتیجه کدام است؟
- (c) به احتمال زیاد، برنامه قبلی تورهای قابل قبولی عرضه می‌کند، اما این تورها کامل نیستند. اکنون تورهای ناقص را خارج کرده و به برنامه اجازه دهید تا تولید یک تور کامل به کار خود ادامه دهد. [هشدار: این نسخه از برنامه بر روی یک کامپیوتر قدرتمند در حدود چند ساعت زمان صرف خواهد کرد.]
- (d) نسخه "brute force" را با نسخه "استراتژی دسترسی" در حرکت مهره شترنج مقایسه کنید. کدامیک را ترجیح می‌دهید؟ توسعه کدام الگوریتم مشکل‌تر است؟ کدامیک به توان کامپیوتر بیشتر احتیاج دارد؟
- ۷-۲۶ یکی دیگر از مسائل صفحه شترنج، مسئله هشت ملکه است، به اینصورت: آیا امکان دارد که هشت ملکه را بر روی صفحه خالی شترنج به نحوی قرار داد که هیچ ملکه‌ای به ملکه دیگر حمله نکند، یعنی هیچ دو ملکه در یک سطر، ستون یا در امتداد یک خط مورب قرار نگیرد؟ با استفاده از تفکر ارائه شده در تمرین ۷-۲۴ یک استراتژی برای حل مسئله هشت ملکه بدست آورید. برنامه اجرا کنید.
- ۷-۲۷ در این تمرین، می‌خواهیم که به روش brute force مسئله هشت ملکه که در تمرین ۷-۲۷ گفته شده پردازید.
- (a) با استفاده از تکنیک تولید عدد تصادفی که در تمرین ۷-۲۵ بیان شده، به حل تمرین هشت ملکه بپردازید.
- (b) از یک تکنیک جامع استفاده کنید، یعنی تمام حالت ممکنه از ترکیب هشت ملکه بر روی صفحه شترنج را در نظر بگیرید.
- (c) چرا گمان می‌کنید روش جامع نمی‌تواند برای حل مسئله حرکت مهره اسب مناسب باشد؟
- (d) روش‌های b, a را با هم مقایسه کنید.
- ۷-۲۸ در مسئله حرکت مهره اسب، یک تور کامل زمانی اتفاق می‌افتد که مهره اسب در هر 64 خانه صفحه شترنج فقط یک بار فرود آمده باشد. حرکت closed tour زمانی رخ می‌دهد که شش و چهارمین حرکت یک خانه از مکان شروع حرکت مهره اسب فاصله داشته باشد. برنامه حرکت مهره اسب نوشته شده در تمرین ۷-۲۴ را برای تست closed tour اصلاح کنید، اگر یک تور کامل رخ داده باشد.
- ۷-۲۹ عدد اول، عددی است که همواره برخودش و 1 قابل تقسیم باشد. روش sieve of Eratosthene روشی برای یافتن اعداد اول است. این روش بصورت زیر عمل می‌کند:
- (a) ایجاد یک آرایه که تمام عناصر آن با 1 مقداردهی اولیه شده‌اند(true). عناصر آرایه با شاخص عدد اول به همان صورت یعنی 1 نگهداری می‌شوند. دیگر عناصر آرایه سرانجام با صفر تنظیم خواهند شد. شما می‌توانید عناصر 0 و 1 را در این تمرین نادیده بگیرید.
- (b) کار با شاخص 2 شروع می‌شود، هر زمان که یک عنصر آرایه یافته شد که مقدار آن 1 است، حلقه در مابقی آرایه حرکت کرده و هر عنصری را که شاخص آن مضربی از شاخص برای عنصری با مقدار 1 است، با صفر تنظیم



می‌کند. در آرایه با شاخص ۲، تمام عناصر که مضری از ۲ هستند. با صفر تنظیم می‌شوند (شاخص‌های ۱۰, ۸, ۶, ۴ و الی آخر)، در آرایه با شاخص ۳، تمام عناصر که مضری از ۳ هستند با صفر تنظیم می‌شوند (شاخص‌های ۱۲, ۹, ۶ و الی آخر) و همینطور تا آخر.

زمانیکه این فرآیند کامل شد، عناصر آرایه که هنوز ۱ باقی مانده‌اند نشان می‌دهند که شاخص یک عدد اول است. می‌توان این شاخص‌ها را چاپ کرد. برنامه‌ای بنویسید که از یک آرایه ۱۰۰ عنصری برای تعیین و چاپ اعداد اول مابین ۲ تا ۹۹۹ استفاده کند. عنصر صفر آرایه را در نظر نگیرید.

۷-۳۰ مرتب‌سازی باکت (bucket sort) بر روی یک آرایه تک بعدی از مقادیر مثبت صحیح شروع و آنرا مرتب می‌کند و بر روی یک آرایه دو بعدی از مقادیر صحیح با سطرهای شاخص‌گذاری شده از ۰ تا ۹ و ستوان‌های شاخص‌گذاری شده از صفر تا $n-1$ شروع می‌شود که n تعداد مقادیر در آرایه است که مرتب خواهد شد. به هر سطر آرایه دو بعدی، یک باکت (bucket) گفته می‌شود. تابعی بنام **bucketSort** بنویسید که یک آرایه صحیح و سایز آرایه را بعنوان آرگومان دریافت کرده و مراحل زیر را انجام دهد:

(a) مقدار هر آرایه یک بعدی را در سط्रی از باکت آرایه و بر مبنای رقم یکان مقدار قرار دهد. برای مثال، ۹۷ در سطر ۷، ۳ در سطر ۳ و ۱۰۰ در سطر صفر جای داده شود. به اینحالت "گذرا توزیعی" گفته می‌شود.

(b) سطر به سطر از میان باکت آرایه عبور کرده و مقادیر را به آرایه اصلی کپی کنید به اینحالت "گذرا تجمعی" گفته می‌شود. ترتیب جدید مقادیر فوق الذکر یک بعدی بصورت ۹۷, ۳, ۹۹, ۱۰۰ خواهد بود.

(c) این فرآیند را برای هر موقعیت مکانی رقم تکرار کنید (دهگان، صدگان، هزارگان، ...). در دومین گذرا، ۱۰۰ در سطر صفر، ۳ در سطر صفر (چرا که ۳ دارای رقم دهگان نیست) و ۹۷ در سطر ۹ جای داده می‌شود. پس از گذرا تجمعی، ترتیب مقادیر در آرایه یک بعدی بصورت ۹۷, ۳, ۹۹, ۱۰۰ خواهد بود. در سومین گذرا، ۱۰۰ در سطر ۱، ۳ در سطر صفر و ۹۷ در سطر صفر جای خواهد گرفت (پس از ۳). پس از آخرین گذرا تجمعی، آرایه اصلی مرتب شده خواهد بود و تکنیک بکار رفته در روش باکت به نسبت مرتب‌سازی درجی از کارایی بهتری برخوردار است، اما نیازمند حافظه بیشتر می‌باشد.

تمرینات بازگشته

۷-۳۱ در مرتب‌سازی انتخابی (selection sort) آرایه بدلبال کوچکترین عنصر جستجو می‌شود. سپس جای کوچکترین عنصر با اولین عنصر در آرایه عوض می‌شود. این فرآیند برای زیرآرایه که با دومین عنصر آرایه آغاز می‌شود، تکرار می‌گردد. در هر گذرا آرایه یک عنصر در مکان صحیح خود قرار داده می‌شود. زمانیکه زیرآرایه به یک عنصر ختم شود، پس آرایه مرتب شده است. تابع بازگشته **selectionSort** را برای انجام این الگوریتم بنویسید.

۷-۳۲ پالندروم رشته‌ای است که تلفظ آن از ابتداء و هم از انتهای یکسان است. برای مثال عبارات زیر همگی پالندروم هستند: "radar"، "able was i ere i saw elba" و اگر فاصله‌ها را نادیده بگیریم عبارت "a man a plan a canal panama". تابع بازگشته بنام **testPalindrome** بنویسید که اگر رشته ذخیره شده در آرایه، پالندروم باشد، مقدار true بازگرداند و در غیر اینصورت مقدار false. تابع باید فضاهای خالی را در نظر نگیرد.



۷-۳۳ (جستجوی خطی) برنامه ۱۹-۷ را با استفاده از تابع بازگشته **linearSearch** اصلاح کنید. این تابع باید یک آرایه از نوع صحیح، یک کلید جستجو، شاخص آغازین و شاخص پایانی را به عنوان آرگومان دریافت کند. اگر کلید جستجو یافت شود، شاخص آرایه بعنوان پاسخ برگشت داده شود و در غیر اینصورت مقدار ۱-چاپ گردد.

۷-۳۴ برنامه هشت ملکه ایجاد شده در تمرین ۷-۲۶ را بصورت بازگشته پیاده سازی کنید.

۷-۳۵ یک تابع بازگشته بنام **printArray** بنویسید که یک آرایه، شاخص شروع و شاخص پایانی را بعنوان آرگومان دریافت کرده و چیزی برگشت ندهد. تابع باید زمانی پردازش را متوقف و برگشت یابد که شاخص شروع معادل با شاخص پایانی باشد.

۷-۳۶ تابع بازگشته بنام **stringReverse** بنویسید که یک آرایه کاراکتری حاوی یک رشته و شاخص شروع را به عنوان آرگومان دریافت کرده، رشته را بصورت معکوس چاپ کرده و چیزی برگشت ندهد. تابع باید زمانی به پردازش خاتمه دهد که با کاراکتر null مواجه شده باشد.

۷-۳۷ تابع بازگشته بنام **recursiveMinimum** بنویسید که یک آرایه صحیح، شاخص شروع و پایان را بعنوان آرگومان دریافت و کوچکترین عنصر آرایه را برگشت دهد. تابع باید زمانی به پردازش خاتمه دهد که شاخص شروع معادل با شاخص پایان باشد.

تمرینات vector

۷-۳۸ از یک **vector** صحیح برای حل مسئله توضیح داده شده در تمرین ۱۰-۷ استفاده کنید.

۷-۳۹ برنامه پرتاب طاس مطرح شده در تمرین ۷-۱۷ را برای استفاده از **vector** به منظور ذخیره سازی تعداد دفعات مجموع پرتاب طاس ها اصلاح کنید.

۷-۴۰ راه حل مطرح شده در تمرین ۳۷-۷ را برای یافتن کوچکترین مقدار در یک **vector** بجای یک آرایه اصلاح کنید.