فصل هشتم

اشاره‌گرها و رشته‌های مبتنی بر اشاره‌گر

اهداف

- اشاره‌گرها چیستند.
- شباهت و تفاوت مابین اشاره‌گرها و مراجعه‌ها و زمان استفاده از آنها.
- کاربرد اشاره‌گرها برای ارسال آرگومان‌ها به توابع به روش مراجعه.
- استفاده از رشته‌های مبتنی بر اشاره‌گر.
- رابطه نزدیک مابین اشاره‌گرها، آرایه‌ها و رشته‌ها.
- کاربرد اشاره‌گرها در توابع.
- اعلان و استفاده از آرایه‌های رشته‌ای.

www.yourshop.4kia.ir

رئوس مطالب

۸-۱ مقدمه
۸-۲ اعلان و مقداردهی اولیه متغیرهای اشاره‌گر
۸-۳ عملگرهای اشاره‌گر
۸-٤ ارسال آرگومان به توابع به روش مراجعه با اشاره‌گرها
۸-۵ کاربرد const همراه با اشاره‌گرها
۸-٦ مرتب‌سازی انتخابی به روش مراجعه
۸-۷ عملگر sizeof
۸-۸ عبارات اشاره‌گر و محاسبات اشاره‌گر
۸-۹ رابطه مابین اشاره‌گرها و آرایه‌ها
۸-۱۰ آرایه‌ای از اشاره‌گرها
۸-۱۱ مبحث آموزشی: بازی کارت
۸-۱۲ اشاره‌گرهای تابع
۸-۱۳ پردازش رشته‌های مبتنی بر اشاره‌گر
۸-۱۳-۱ اصول کاراکترها و رشته‌های مبتنی بر اشاره‌گر
۸-۱۳-۲ توابع دستکاری کننده رشته

۸-۱ مقدمه

بحث این فصل در مورد یکی از مهمترین و قویترین ویژگیهای زبان برنامه نویسی C++، یعنی *اشاره‌گرها* (*pointer*) است. در فصل ششم مشاهده کردید که مراجعه‌ها می‌توانند برای انجام عملیات ارسال به روش *مراجعه* بکار گرفته شوند. اشاره‌گرها هم می‌توانند چنین عملیاتی انجام دهند و می‌توانند برای ایجاد و دستکاری ساختمان داده‌های دینامیکی (پویا) همانند لیست پیوندی، صف‌ها، پشته‌ها و درخت‌ها بکار گرفته شوند. در این فصل به توضیح مفاهیم پایه و بررسی رابطه موجود مابین آرایه‌ها و اشاره‌گرها خواهیم پرداخت. بررسی آرایه‌ها بعنوان اشاره‌گر از زبان برنامه‌نویسی C مشتق شده است. همانطوری‌که در فصل هفتم مشاهده کردید، کلاس **vector** مبادرت به پیاده‌سازی آرایه‌ها به روش خاصی می‌کند. به همین ترتیب، C++ دو نوع رشته عرضه می‌کند، شی‌های از کلاس **string** (که در فصل سوم از آنها استفاده کردیم) و رشته‌های مبتنی بر اشاره‌گر * **char** به سبک C. بحث این فصل بر روی رشته‌های اشاره‌گر * **char** متمرکز است تا دانش شما در ارتباط با اشاره‌گرها عمیق‌تر شود. در واقع، رشته‌های مطرح شده در بخش ۴-۷ و بکار رفته در برنامه ۱۲-۷ از نوع رشته‌های مبتنی بر اشاره‌گر * **char** بودند.

www.yourshop.4kia.ir

در فصل سیزدهم به استفاده از اشاره‌گرها در کلاس‌ها خواهیم پرداخت که به آن در برنامه‌نویسی شی‌گرا «*پردازش چند ریختی*» می‌گویم. در فصل بیست و یکم مثالهای در ارتباط با ایجاد و استفاده از ساختارهای داده پویا که با اشاره‌گرها پیاده‌سازی می‌شوند مطرح کرده‌ایم.

۸-۲ اعلان و مقداردهی اولیه متغیرهای اشاره‌گر

متغیرهای اشاره‌گر حاوی آدرس‌های حافظه بعنوان مقادیر خود هستند. معمولاً، یک متغیر حاوی یک مقدار مشخص است. با این وجود، یک اشاره‌گر حاوی آدرس حافظه از متغیری است که دارای یک مقدار مشخص می‌باشد. در چنین وضعیتی، نام متغیر بصورت مستقیم به یک مقدار مراجعه دارد و یک اشاره‌گر بصورت غیرمستقیم به یک مقدار مراجعه می‌کند (شکل ۱-۸). غالباً مراجعه به یک مقدار از طریق یک اشاره‌گر بصورت غیرمستقیم انجام می‌شود. توجه کنید که در تصاویر، اشاره‌گر بصورت یک فلش از متغیری که حاوی یک آدرس به سمت متغیری که در آن آدرس حافظه قرار دارد نشان داده می‌شود. اشاره‌گرها، همانند هر متغیر دیگری، بایستی قبل از استفاده اعلان شده باشند. برای مثال، برای اشاره‌گر به نمایش درآمده در شکل ۱-۸ اعلان

```
int *countPtr, count;
```

متغیر **countPtr** را از نوع ***int** (یعنی یک اشاره‌گر به یک مقدار **int**) اعلان کرده که به اینصورت خوانده می‌شود «**countPtr** اشاره‌گری به **int** است» یا «**countPtr** اشاره‌گر به یک شی از نوع **int** است.» همچنین در این اعلان، متغیر **count** از نوع **int** اعلان شده است و اشاره‌گری به یک **int** نمی‌باشد. علامت * فقط بر روی **countPtr** اعمال شده است. هر متغیری که می‌خواهد بصورت یک اشاره‌گر اعلان شود باید قبل از آن یک علامت * قرار گرفته باشد. برای مثال، در اعلان

```
double *xPtr, *yPtr;
```

مشخص است که **xPtr** و **yPtr** هر دو اشاره‌گرهای به مقادیری از نوع **double** هستند. زمانیکه * در اعلانی ظاهر می‌شود، دیگر بعنوان یک عملگر مطرح نشده بلکه نشاندهنده متغیری است که بصورت یک اشاره‌گر اعلان شده است. اشاره‌گرها می‌توانند برای اشاره به شی‌های از هر نوع داده اعلان شوند.

خطای برنامه‌نویسی

*فرض اینکه استفاده از یک * در اعلان یک اشاره‌گر بر روی تمام اسامی متغیرها در یک اعلان جدا شده با کاما از یکدیگر تاثیر خواهد گذاشت، تصور اشتباهی است. هر اشاره‌گر باید با یک پیشوند *قبل از نام اعلان شده باشد (خواه با فاصله یا بدون فاصله، کامپایلر فاصله‌ها را نادیده می‌گیرد). اعلان یک متغیر در هر خط کمک می‌کند تا جلوی رخ دادن چنین اشتباهاتی گرفته شود و خوانایی برنامه افزایش یابد.*

برنامه‌نویسی ایده‌ال

*اگر چه انجام این کار ضروری نیست، اما بهتر است حروف **Ptr** را برای نشان دادن اینکه متغیر از نوع اشاره‌گر است با نام متغیر همراه کنید.*

www.yourshop.4kia.ir

شکل ۱-۸ | مراجعه مستقیم و غیرمستقیم به یک متغیر.

بایستی اشاره‌گرها به هنگام اعلان یا توسط یک عبارت تخصیص‌دهنده مقداردهی اولیه شوند. یک اشاره‌گر می‌تواند با صفر، **NULL** یا یک آدرس مقداردهی اولیه شود. یک اشاره‌گر با مقدار صفر یا NULL به چیزی اشاره نمی‌کند و بعنوان یک *اشاره‌گر null* شناخته می‌شود. ثابت سمبولیک NULL در فایل سرآیند <**iostream**> و چند فایل سرآیند از کتابخانه استاندارد تعریف شده که نشاندهنده مقدار صفر است. مقداردهی اولیه یک اشاره‌گر با NULL معادل با مقداردهی اولیه یک اشاره‌گر با صفر است، اما در C++ بطور قراردادی از صفر استفاده می‌شود. زمانیکه صفر تخصیص داده می‌شود، به یک اشاره‌گر از نوع مقتضی برگردانده می‌شود. مقدار صفر تنها مقدار صحیحی است که می‌تواند مستقیماً به یک متغیر اشاره‌گر تخصیص داده شود بدون اینکه ابتدا نوع صحیح به نوع اشاره‌گر تبدیل شود.

اجتناب از خطا

برای اجتناب از اشاره اشاره‌گرها به مکان‌های ناشناخته و مقداردهی نشده حافظه، مبادرت به مقداردهی اولیه اشاره‌گرها کنید.

۳ـ ۸ عملگرهای اشاره‌گر

عملگر آدرس (**&**) یک عملگر غیربایری است که آدرس حافظه علموند خود را برگشت می‌دهد. برای مثال، با فرض اعلان‌های

```
int y = 5; //declare variable y
int *yPtr; // declare pointer variable yPtr
```

عبارت

```
yPtr = &y; // assign address of y to yPtr
```

آدرس متغیر **y** را به متغیر اشاره‌گر **yPtr** تخصیص می‌دهد. پس متغیر **yPtr** به **y** اشاره می‌کند. اکنون، **yPtr** بصورت غیرمستقیم به مقدار متغیر **y** مراجعه دارد. به نحوه استفاده از **&** در عبارت انتساب فوق دقت کنید که مشابه کاربرد **&** در اعلان متغیر مراجعه‌ای نیست.

شکل ۲-۸ نمایشی از حافظه پس از تخصیص فوق است. رابطه اشاره‌گر توسط یک فلش یا پیکان از جعبه‌ای که نشاندهنده اشاره‌گر **yPtr** در حافظه به جعبه‌ای که نشاندهنده متغیر **y** در حافظه است، عرضه شده است.

شکل ۳-۸ نمایش دیگری از اشاره‌گر در حافظه است، با فرض اینکه متغیر صحیح **y** در مکان 600000 حافظه و متغیر اشاره‌گر **yPtr** در مکان 500000 حافظه ذخیره شده است. عملوند، عملگر آدرس بایستی یک *lvalue* (یعنی چیزی که بتوان مقداری به آن تخصیص داد همانند نام یک متغیر یا مراجعه) باشد، عملگر آدرس نمی‌تواند با ثابت‌ها یا عباراتی که نمی‌توانند در مراجعه نتیجه‌ای بدست دهند بکار گرفته شود.

www.yourshop.4kia.ir

شکل ۲-۸ | نمایش گرافیکی از اشاره یک اشاره‌گر به یک متغیر در حافظه.

شکل ۳-۸ | نمایش y و yPtr در حافظه.

معمولاً از عملگر * بعنوان یک عملگر غیرمستقیم یا عملگر غیرمراجعه‌ای یاد می‌شد که یک مترداف برای شی که عملوند اشاره‌گر به آن اشاره می‌کند، برگشت می‌دهد. برای مثال (به شکل ۲-۸ توجه کنید) عبارت

```
cout << *yPtr << endl;
```

مقدار متغیر **y**، یعنی 5 را همانند عبارت زیر چاپ می‌کند

```
cout << y << endl;
```

به استفاده از عملگر * به این روش، مراجعه غیرمستقیم اشاره‌گر گفته می‌شود. توجه کنید که مراجعه غیرمستقیم اشاره‌گر می‌تواند در سمت چپ یک عبارت تخصیصی بکار گرفته شود، همانند

```
*yPtr = 9;
```

که مقدار 9 را به **y** در شکل ۳-۸ تخصیص می‌دهد. همچنین می‌توان از این عملگر برای بازیابی مقدار ورودی استفاده کرد، برای مثال

```
cin >> *yPtr;
```

مقدار ورودی را در y قرار می‌دهد. اشاره‌گر غیرمراجعه‌ای یک *lvalue* است.

در برنامه شکل ۴-۸ به توضیح عملگرهای اشاره‌گر **&** و * پرداخته شده است. مکان‌های حافظه چاپ شده توسط >> در این مثال بصورت مقادیر هگزادسیمال (پایه 16) هستند. توجه کنید که آدرس‌های حافظه هگزادسیمال بدست آمده در این برنامه وابسته به کامپایلر و سیستم عامل هستند و امکان دارد برنامه شما نتایج متفاوتی از این برنامه تولید کند.

```
// Fig. 8.4: fig08_04.cpp
// Using the & and * operators.
#include <iostream>
using std::cout;
using std::endl;

int main()
{
   int a; // a is an integer
   int *aPtr; // aPtr is an int * -- pointer to an integer

   a = 7; // assigned 7 to a
   aPtr = &a; // assign the address of a to aPtr

   cout << "The address of a is " << &a
      << "\nThe value of aPtr is " << aPtr;
   cout << "\n\nThe value of a is " << a
      << "\nThe value of *aPtr is " << *aPtr;
   cout << "\n\nShowing that * and & are inverses of "
      << "each other.\n&*aPtr = " << &*aPtr
      << "\n*&aPtr = " << *&aPtr << endl;
   return 0; // indicates successful termination
} // end main
```

```
The address of a is 0012F580
The value of aPtr is 0012F580

The value of a is 7
The value of *aPtr is 7
```

www.yourshop.4kia.ir

```
Showing that * and & are inverses of each other.
&*aPtr = 0012F580
*&aPtr = 0012F580
```

شکل ٤-٨ | عملگرهای & و *.

دقت کنید که آدرس **a** (خط 15) و مقدار **aPtr** (خط 16) در خروجی یکسان هستند و تایید می‌کنند که آدرس **a** براستی به متغیر اشاره‌گر **aPtr** تخصیص یافته است. عملگرهای **&** و ***** وارون همدیگر هستند. زمانیکه هر دو آنها بصورت متوالی بر روی **aPtr** بکار گرفته شوند به هر ترتیب، سبب «لغو دیگری شده» و مقدار یکسانی (مقدار **aPtr**) چاپ می‌شود.

در جدول شکل ٥-٨ لیستی از تقدم و شرکت پذیری عملگرهای مطرح شده تا بدین جا آورده شده است. توجه کنید که عملگر آدرس (**&**) و عملگر غیرمراجعه‌ای (*****) از جمله عملگرهای غیربایزی در سومین سطح از جدول تقدم عملگرها هستند.

عملگر	ارتباط	نوع
()	**چپ به راست**	**پرانتز**
++ -- static_cast< *type* **>()**	**چپ به راست**	**غیربایزی**
++ -- + -	**راست به چپ**	**غیربایزی**
*** / %**	**چپ به راست**	**تعددی**
+ -	**چپ به راست**	**افزاینده کاهنده**
<< >>	**چپ به راست**	**درج/استخراج**
< <= > >=	**چپ به راست**	**رابطه‌ای**
== !=	**چپ به راست**	**برابری**
&&	**چپ به راست**	**ANDمنطقی**
||	**چپ به راست**	**ORمنطقی**
?:	**راست به چپ**	**شرطی**
= += -= *= /= %=	**راست به چپ**	**تخصیصی**
,	**چپ به راست**	**کاما**

شکل ٥-٨ | تقدم و شرکت پذیری عملگرها.

٤-٨ ارسال آرگومان به توابع به روش مراجعه با اشاره‌گرها

در C++ سه روش برای ارسال آرگومان به یک تابع وجود دارد، ارسال با مقدار، *ارسال با مراجعه* با *آرگومان‌های مراجعه‌ای* و *ارسال با مراجعه* با *آرگومان‌های اشاره‌گر*.

در فصل ششم به مقایسه ارسال آرگومان مراجعه‌ای به روش مقدار و مراجعه پرداختیم. در این بخش به توضیح ارسال آرگومان‌های اشاره‌گر به روش مراجعه می‌پردازیم. همانطوری که در فصل ششم مشاهده کردید، **return** می‌توانست برای بازگرداندن یک مقدار از تابع فراخوانی شده به فراخوان (یا بازگرداندن

www.yourshop.4kia.ir

کنترل از تابع فراخوانی شده بدون برگشت دادن مقداری) بکار گرفته شود. همچنین شاهد بودید که آرگومان‌ها می‌توانستند به تابع با استفاده از آرگومان‌های مراجعه‌ای ارسال شوند. چنین آرگومان‌های به تابع فراخوانی شده امکان تغییر در مقادیر اصلی آرگومان‌ها در فراخوان را می‌دهند. همچنین آرگومان‌های مراجعه‌ای به برنامه‌ها امکان ارسال شی‌ها با داده‌های بزرگ به یک تابع را فراهم می‌آورند و از سربارگذاری ارسال شی به روش مقدار اجتناب می‌شود (که مستلزم ایجاد یک کپی از شی است). از اشاره‌گرها همانند مراجعه‌ها، می‌توان برای اصلاح یک یا چندین متغیر در فراخوان یا ارسال اشاره‌گرها به شی‌های با داده‌های بزرگ استفاده کرد تا از سربارگذاری ارسال شی‌ها به روش مقدار جلوگیری شود.

در C++، برنامه‌نویسان می‌توانند از اشاره‌گرها و عملگر غیرمستقیم (*) برای انجام ارسال به روش مراجعه استفاده کنند. در زمان فراخوانی تابعی با آرگومانی که باید اصلاح شود، آدرس آرگومان ارسال می‌شود. معمولاً اینکار با اعمال عملگر آدرس (&) بر روی نام متغیری که مقدار آن تغییر خواهد یافت صورت می‌گیرد.

همانطوری که در فصل هفتم مشاهده کردید، آرایه‌ها با استفاده از عملگر & قادر به ارسال نیستند، چرا که نام آرایه مکان شروع در حافظه برای آن آرایه است (یعنی نام آرایه خود یک اشاره‌گر است). نام یک آرایه همانند **arrayName**، معادل با **&arrayName[0]** است. زمانیکه آدرس یک متغیر به یک تابع ارسال می‌شود، عملگر غیرمستقیم (*) می‌تواند در تابع بفرم مترادفی از نام متغیر بکار گرفته شود، در اینحالت می‌تواند مقدار متغیر در آن مکان از حافظه فراخوان را تغییر دهد.

برنامه‌های شکل ۶-۸ و ۷-۸ دو نسخه از یک تابع را عرضه می‌کنند که مکعب دو مقدار صحیح **cubeByReference** و **cubeByValue** را محاسبه می‌کنند. در برنامه شکل ۶-۸ متغیر **number** به روش مقدار به تابع **cubeByValue** (خط 15) ارسال می‌شود. تابع **cubeByValue** (خطوط 21-24) مکعب آرگومان خود را محاسبه مقدار جدید را به **main** و با استفاده از دستور **return** برگشت می‌دهد (خط 23). مقدار جدید به **nubmer** در **main** تخصیص می‌یابد (خط 15). توجه کنید که تابع فراخوان فرصت بررسی نتیجه از فراخوان تابع را قبل از اصلاح مقدار متغیر **number** را دارد. برای مثال، در این برنامه، می‌توانیم نتیجه **cubeByValue** را در متغیر دیگری ذخیره کرده و به بررسی مقدار آن پرداخته و فقط در صورتیکه تشخیص دهیم که مقدار برگشتی قابل قبول است آنرا به **number** تخصیص دهیم.

```
1  Fig. 8.6: fig08_06.cpp
2  // Cube a variable using pass-by-value.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  int cubeByValue( int ); // prototype
8
9  int main()
10 {
```

www.yourshop.4kia.ir

```
11    int number = 5;
12
13    cout << "The original value of number is " << number;
14
15    number = cubeByValue( number ); // pass number by value to cubeByValue
16    cout << "\nThe new value of number is " << number << endl;
17    return 0; // indicates successful termination
18 } // end main
19
20 // calculate and return cube of integer argument
21 int cubeByValue( int n )
22 {
23    return n * n * n; // cube local variable n and return result
24 } // end function cubeByValue
```

```
The original value of number is 5
The new value of number is 125
```

شکل ٦-٨| ارسال به روش مقدار برای محاسبه مکعب مقدار یک متغیر.

در برنامه شکل ۷-۸ متغیر **number** به تابع **cubeByReference** با استفاده از روش مراجعه با یک آرگومان اشاره‌گر ارسال شده است (خط 15). آدرس **number** به تابع ارسال می‌شود. تابع **cubeByReference** در خطوط 25-22 تصریح کننده پارامتر **nPtr** (یک اشاره‌گر به **int**) برای دریافت آرگومان خود است. تابع، اشاره‌گر را دنبال و مقدار مکعب که **nPtr** به آن اشاره می‌کند را محاسبه می‌نماید (خط 24). در اینحالت مقدار **nubmer** در **main** مستقیماً تغییر می‌یابد.

تابع دریافت کننده آدرس بعنوان یک آرگومان بایستی یک پارامتر اشاره‌گر به آدرس دریافتی تعریف کرده باشد. برای مثال در سرآیند تابع **cubeByReference** در خط 12 مشخص شده است که **cubeByReference** آدرس یک متغیر **int** (یعنی یک اشاره‌گر به یک **int**) را بعنوان آرگومان دریافت می‌کند، آدرس را بصورت محلی در **nPtr** ذخیره کرده و مقداری برگشت نمی‌دهد.

نمونه اولیه تابع برای **cubeByReference** در خط 7 حاوی **int *** در درون پرانتز است. همانند سایر انواع متغیرها، نیازی به شامل کردن اسامی پارامترهای اشاره‌گر در نمونه اولیه تابع نیست. هدف از قرار دادن اسامی فقط تهیه مستندات بوده و توسط کامپایلر نادیده گرفته می‌شوند. تصاویر ۸-۸ و ۹-۸ یک تحلیل گرافیکی به ترتیب از اجرای برنامه‌های ۶-۸ و ۷-۸ ارائه می‌کنند.

```
1  // Fig. 8.7: fig08_07.cpp
2  // Cube a variable using pass-by-reference with a pointer argument.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  void cubeByReference( int * ); // prototype
8
9  int main()
10 {
11    int number = 5;
12
13    cout << "The original value of number is " << number;
14
15    cubeByReference( &number ); // pass number address to cubeByReference
16
17    cout << "\nThe new value of number is " << number << endl;
18    return 0; // indicates successful termination
19 } // end main
20
```

www.yourshop.4kia.ir

```
// calculate cube of *nPtr; modifies variable number in main
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr; // cube *nPtr
} // end function cubeByReference
```

```
The original value of number is 5
The new value of number is 125
```

شکل ۷-۸ | ارسال به روش مراجعه با یک آرگومان اشاره‌گر برای محاسبه مکعب یک متغیر.

شکل ۸-۸ | تحلیل ارسال به روش مقدار در برنامه شکل ٦-۸.

شکل ۹-۸ | تحلیل ارسال به روش مراجعه (با یک آرگومان اشاره‌گر) در برنامه شکل ۷-۸.

۵-۸ کاربرد const همراه با اشاره‌گرها

بخاطر دارید که توصیف‌کننده **const** به برنامه‌نویس امکان می‌دهد تا به کامپایلر اطلاع دهد که مقدار یک متغیر خاص نبایستی تغییر یابد.

قابلیت حمل

*اگرچه **const** بخوبی در **ANSI C** و **++C** تعریف شده است، اما برخی از کامپایلر آنرا بطور کامل رعایت نمی‌کنند. بهتر است به بررسی کامپایلر خود بپردازید.*

سالها بخش اعظمی از کدهای نوشته شده و به یادگار مانده از نسخه‌های اولیه C از وجود **const** تهی بودند چرا که وجود نداشت. به همین دلیل مجال مناسبی برای بهبود کد قدیمی C بوجود آمده است. همچنین هنوز هم برخی از برنامه‌نویسان استفاده کننده از ANSI C و ++C از **const** در برنامه‌های خود استفاده نمی‌کنند، چراکه آنها برنامه‌نویسی را با نسخه‌های اولیه C شروع کرده و یاد گرفته‌اند. چنین برنامه‌نویسانی از فرصت‌های مناسبی که یک مهندسی نرم‌افزار ایده‌ال می‌تواند در اختیار آنها قرار دهد، محروم هستند.

دلایل مختلفی برای استفاده (یا عدم استفاده) از **const** در کنار پارامترهای تابع وجود دارد. چگونه شرایط انتخاب یا عدم انتخاب را تشخیص می‌دهید؟ اجازه دهید تا به بررسی یک روش و قاعده علمی بپردازیم. همیشه برای یک تابع، آن میزان دسترسی به داده پارامترهای خود اعطاء کنید که بتواند وظیفه خود را به انجام برساند، نه بیشتر. بحث این بخش در ارتباط با نحوه ترکیب **const** با اعلان اشاره‌گر است تا حداقل قواعد علمی رعایت شود.

در فصل ششم توضیح داده شد که به هنگام فراخوانی یک تابع به روش ارسال با مقدار، یکی کپی از آرگومان (یا آرگومان‌ها) در فراخوان تابع ایجاد و به تابع ارسال می‌شود. اگر کپی در تابع تغییر یابد، مقدار اصلی همچنان در فراخوان بدون تغییر باقی می‌ماند. در برخی از موارد، یک مقدار ارسالی به تابع تغییر داده می‌شود تا اینکه تابع بتواند وظیفه خود را انجام دهد. با این همه، در برخی از شرایط نبایستی مقدار در تابع فراخوانی شده تغییر پیدا کند، حتی اگر تابع فراخوانی شده مبادرت به دستکاری کردن کپی از مقدار اصلی نماید.

www.yourshop.4kia.ir

برای مثال فرض کنید تابعی یک آرایه تک بعدی و سایز آنرا بعنوان آرگومان دریافت کرده و متعاقب آن آرایه را چاپ می‌کند. چنین تابعی باید با استفاده از یک حلقه در میان آرایه حرکت کرده و هر عنصر آرایه را بصورت مجزا در خروجی قرار دهد. از سایز در بدنه تابع برای تعیین بالاترین شاخص آرایه استفاده می‌شود، از اینروست که حلقه می‌تواند زمان کامل شدن عملیات چاپ را تشخیص دهد. سایز آرایه نمی‌تواند در بدنه تابع تغییر یابد، بنابر این بایستی بعنوان **const** اعلان شود. البته، بدلیل اینکه آرایه فقط چاپ می‌شود، بایستی خود آنرا را هم بصورت **const** اعلان کرده باشیم. اینکار از اهمیت خاصی برخوردار است چراکه کل آرایه همیشه بصورت مراجعه ارسال می‌شود و می‌تواند به آسانی در تابع فراخوانی شده تغییر داده شود.

مهندسی نرم‌افزار

*اگر مقداری در بدنه تابع که به آن ارسال شده تغییر نمی‌یابد (یا نبایستی تغییر داده شود)، باید آن پارامتر بصورت **const** اعلان شود تا از تغییر ناخواسته آن اجتناب گردد.*

اگر مبادرت به تغییر مقدار یک **const** شود، پیغام هشدار یا خطا با توجه به نوع کامپایلر صادر خواهد شد.

اجتناب از خطا

قبل از استفاده از یک تابع، به بررسی نمونه اولیه تابع بپردازید تا مشخص شود که پارامترها می‌توانند تغییر یابند یا خیر.

چهار روش برای ارسال یک اشاره‌گر به یک تابع وجود دارد: اشاره‌گر غیرثابت به داده غیرثابت (شکل ۱۰-۸)، اشاره‌گر غیرثابت به داده ثابت (شکل ۱۱-۸ و ۱۲-۸)، اشاره‌گر ثابت به داده غیرثابت (شکل ۱۳-۸) و اشاره‌گر ثابت به داده ثابت (شکل ۱۴-۸). هر یک از این حالت‌ها مجوزهای دسترسی متفاوتی بوجود می‌آورند.

اشاره‌گر غیرثابت به داده غیرثابت

بالاترین سطح دسترسی توسط یک اشاره‌گر غیرثابت به یک داده غیرثابت اهداء می‌شود. داده می‌تواند از طریق دسترسی به مقدار اشاره‌گر تغییر داده شود و اشاره‌گر می‌تواند برای اشاره به داده دیگری اصلاح یا تغییر داده شود. در اعلان یک اشاره‌گر غیرثابت به داده غیرثابت نیازی به حضور **const** نیست. چنین اشاره‌گری می‌تواند برای بازیابی یک رشته پایان یافته با **null** در یک تابع که مقدار اشاره‌گر را در ضمن پردازش هر کاراکتر در رشته تغییر می‌دهد، بکار گرفته شود. از بخش ۴-۷ بخاطر دارید که چنین رشته‌ای می‌تواند در یک آرایه کاراکتری که حاوی کاراکترهای آن رشته و یک کاراکتر **null** بعنوان نشاندهنده انتهای رشته است، جای داده شود.

www.yourshop.4kia.ir

در برنامه شکل ۱۰-۸، تابع **convertToUppercase** در خطوط 34-25 پارامتر **sPtr** (خط 25) را بصورت یک اشاره‌گر غیرثابت به یک داده غیرثابت اعلان کرده است (بدون حضور **const**). تابع مبادرت به پردازش یک کاراکتر در هر زمان از یک رشته خاتمه یافته با **null** و ذخیره شده در آرایه کاراکتری **phrase** می‌کند (خطوط 33-27). بخاطر داریدکه نام آرایه کاراکتری معادل یک اشاره‌گر به اولین کاراکتر آرایه است، از اینرو ارسال **phrase** بعنوان یک آرگومان به **convertToUppercase** ممکن است. تابع **islower** در خط 29 یک آرگومان کاراکتری دریافت و اگر کاراکتر یک حرف کوچک باشد مقدار **true** و در غیراینصورت **false** برگشت می‌دهد.

```
1  // Fig. 8.10: fig08_10.cpp
2  // Converting lowercase letters to uppercase letters
3  // using a non-constant pointer to non-constant data.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  #include <cctype> // prototypes for islower and toupper
9  using std::islower;
10 using std::toupper;
11
12 void convertToUppercase( char * );
13
14 int main()
15 {
16    char phrase[] = "characters and $32.98";
17
18    cout << "The phrase before conversion is: " << phrase;
19    convertToUppercase( phrase );
20    cout << "\nThe phrase after conversion is:  " << phrase << endl;
21    return 0; // indicates successful termination
22 } // end main
23
24 // convert string to uppercase letters
25 void convertToUppercase( char *sPtr )
26 {
27    while ( *sPtr != '\0' ) // loop while current character is not '\0'
28    {
29       if ( islower( *sPtr ) ) // if character is lowercase,
30          *sPtr = toupper( *sPtr ); // convert to uppercase
31
32       sPtr++; // move sPtr to next character in string
33    } // end while
34 } // end function convertToUppercase
```

```
the phrase before conversion is: characters and $32.98
the phrase after conversion is:  CHARACTERS AND $32.98
```

شکل ۱۰-۸ | تبدیل رشته به حرف بزرگ.

کاراکترهای قرار گرفته در محدودهٔ **'a'** تا **'z'** به حروف متناظر و بزرگ خود توسط تابع **toupper** تبدیل می‌شوند (خط 30) و کاراکترهای دیگر بلاتغییر باقی می‌مانند. تابع **toupper** یک کاراکتر بعنوان آرگومان دریافت می‌کند. اگر کاراکتر یک حرف کوچک باشد، حرف بزرگ متناظر با آن برگشت داده می‌شود، در غیر اینصورت کاراکتر اصلی برگردانده خواهد شد. تابع **toupper** و تابع **islower** بخشی از کتابخانه رسیدگی‌کننده به کاراکتر **<cctype>** هستند. پس از پردازش یک کاراکتر، خط 32 مقدار **sPtr**

www.yourshop.4kia.ir

را یک واحد افزایش می‌دهد (اگر **sPtr** بصورت **const** اعلان شده بود انجام اینکار ممکن نبود). به هنگام اعمال عملگر + به اشاره‌گری که به یک آرایه اشاره می‌کند، آدرس حافظه ذخیره شده در اشاره‌گر برای اشاره به عنصر بعدی آرایه تغییر می‌یابد (در این مورد کاراکتر بعدی در رشته). افزودن یک واحد به اشاره‌گر یک عملیات معتبر در حساب اشاره‌گر است که در بخش ۸–۸ و ۹–۸ با جزئیات آنها بیشتر آشنا خواهید شد.

اشاره‌گر غیرثابت به داده ثابت

یک اشاره‌گر غیرثابت به داده ثابت اشاره‌گری است که می‌تواند برای اشاره به هر ایتم داده از نوع مقتضی تغییر داده شود، اما داده‌ی که به آن اشاره می‌کند در مدت زمان اشاره به آن قابل تغییر نمی‌باشد. می‌توان از چنین اشاره‌گری در دریافت یک آرگون آرایه به تابعی که هر عنصر آرایه را پردازش خواهد کرد استفاده کرد، اما نبایستی اجازه تغییر در داده صادر شود. برای مثال تابع **printCharacters** (خطوط 22-26 از برنامه شکل ۱۱–۸) پارامتر **sPtr** در خط 22 را از نوع **const char *** اعلان کرده است، از اینروست که می‌تواند رشته خاتمه یافته با **null** و بر پایه اشاره‌گر را دریافت کند. مفهوم این اعلان از سمت چپ به این مضمون است «**sPtr** اشاره‌گری به یک ثابت کاراکتری است.» بدنه تابع از یک عبارت **for** (خط 24-25) برای چاپ هر کاراکتر موجود در رشته تا رسیدن به کاراکتر **null** استفاده کرده است. پس از چاپ هر کاراکتر، اشاره‌گر **sPtr** برای اشاره به کاراکتر بعدی در رشته افزایش داده می‌شود (انجام اینکار ممکن است چرا که اشاره‌گر **const** نیست). تابع **main** آرایه **phrase** از نوع **char** را برای ارسال به **printCharacters** ایجاد می‌کند. مجدداً، می‌توانیم آرایه **phrase** را به **printCharacters** ارسال کنیم چرا که نام آرایه واقعاً یک اشاره‌گر به اولین کاراکتر در آرایه است.

```
1  // Fig. 8.11: fig08_11.cpp
2  // Printing a string one character at a time using
3  // a non-constant pointer to constant data.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  void printCharacters( const char * ); // print using pointer to const data
9
10 int main()
11 {
12    const char phrase[] = "print characters of a string";
13
14    cout << "The string is:\n";
15    printCharacters( phrase ); // print characters in phrase
16    cout << endl;
17    return 0; // indicates successful termination
18 } // end main
19
20 // sPtr can be modified, but it cannot modify the character to which
21 // it points, i.e., sPtr is a "read-only" pointer
22 void printCharacters( const char *sPtr )
23 {
24    for ( ; *sPtr != '\0'; sPtr++ ) // no initialization
25       cout << *sPtr; // display character without modification
26 } // end function printCharacters
```

www.yourshop.4kia.ir

```
The string is:
print characters of a string
```

شکل ۱۱-۸ | چاپ یک کاراکتر در هر زمان با استفاده از یک اشاره‌گر غیرثابت به داده ثابت.

برنامه شکل ۱۲-۸ به توصیف پیغام خطاهای کامپایلر در زمانیکه مبادرت به کامپایل تابعی می‌کند که یک اشاره‌گر غیرثابت به داده ثابت دریافت می‌کند و سپس سعی می‌نماید تا با استفاده از آن اشاره‌گر مبادرت به تغییر دادن داده کند. [نکته: توجه نمائید که پیغام خطای کامپایلر در میان کامپایلرها باهم تفاوت دارد.]

```
1  // Fig. 8.12: fig08_12.cpp
2  // Attempting to modify data through a
3  // non-constant pointer to constant data.
4
5  void f( const int * ); // prototype
6
7  int main()
8  {
9     int y;
10
11    f( &y ); // f attempts illegal modification
12    return 0; // indicates successful termination
13 } // end main
14
15 // xPtr cannot modify the value of constant variable to which it points
16 void f( const int *xPtr )
17 {
18    *xPtr = 100; // error: cannot modify a const object
19 } // end function f
```

Borland C++ command-line compiler error message:

```
Error E2024 fig08_12.cpp 18:
   Cannot modify a const object in function f(const int *)
```

Microsoft Visual C++ .NET compiler error message:

```
c:\cpphtp5_examples\ch08\Fig08_12\fig08_12.cpp(18) :
   error C2166: 1-value specifies const object
```

GNU C++ compiler error message:

```
Fig08_12.cpp: In function void f(const int*)':
Fig08_12.cpp:18: error: assignment of read-only location
```

شکل ۱۲-۸ | مبادرت به تغییر داده از طریق یک اشاره‌گر غیرثابت به داده ثابت.

همانطوری‌که می‌دانید، آرایه‌ها نوع داده‌های به‌هم پیوسته هستند که ایتم‌های داده مرتبط از نوع یکسان را تحت یک نام ذخیره می‌سازند. زمانیکه تابعی با یک آرایه بعنوان آرگومان فراخوانی می‌شود، آرایه به روش مراجعه به تابع ارسال می‌گردد. با این وجود، شی‌ها همیشه به روش مقدار ارسال می‌گردند. یک کپی از کل شی ارسال می‌شود. انجام اینکار مستلزم صرف زمان برای تهیه کپی از هر ایتم داده در شی و ذخیره‌سازی آن در پشته فراخوانی تابع است. زمانیکه باید یک شی به تابعی ارسال شود، می‌توانیم از یک اشاره‌گر به ثابت داده (یا مراجعه به یک ثابت داده) برای بدست آوردن کارایی ارسال به روش مراجعه استفاده کرده و مانع از ارسال به روش مقدار شویم. زمانیکه یک اشاره‌گر به یک شی ارسال می‌گردد، فقط یک کپی از آدرس آن شی تهیه می‌شود و خود شی کپی نمی‌شود. در یک ماشین با آدرس‌های چهار بایتی، یک کپی از چهار بایت حافظه بسیار سریعتر از کپی کردن یک شی بزرگ صورت می‌گیرد.

اشاره‌گر ثابت به داده غیرثابت

www.yourshop.4kia.ir

یک اشاره‌گر ثابت به داده غیرثابت، اشاره‌گری است که همیشه به همان مکان حافظه اشاره دارد، داده موجود در آن مکان می‌تواند از طریق اشاره‌گر تغییر یابد. اینحالت برای نام آرایه حالت پیش فرض است. نام آرایه یک اشاره‌گر ثابت برای نشان دادن ابتدای آرایه است. به کل داده‌های آرایه می‌توان با نام آرایه و شاخص دسترسی پیدا کرده و آنها را تغییر داد از یک اشاره‌گر ثابت به یک داده غیرثابت می‌توان برای بازیابی یک آرایه بعنوان یک آرگومان به تابعی که به عناصر آرایه با استفاده از شاخص دسترسی پیدا می‌کند، استفاده کرد. باید اشاره‌گرهای که بصورت **const** اعلان می‌شوند در زمان اعلان مقداردهی اولیه گردند. برنامه شکل ۱۳-۸ مبادرت تغییر در یک اشاره‌گر ثابت می‌کند. خط 11 اشاره‌گر **ptr** را بصورت **int * const** اعلان کرده است. این اعلان از سمت راست به چپ به اینصورت خوانده می‌شود «**ptr** یک اشاره‌گر ثابت به مقدار صحیح غیرثابت است.» اشاره‌گر با آدرس متغیر صحیح **x** مقداردهی اولیه شده است. در خط 14 مبادرت به تخصیص آدرس **y** به **ptr** می‌شود، اما کامپایلر یک پیغام خطا تولید می‌کند. توجه کنید که تا رسیدن به خط 13 و تخصیص مقدار 7 به **ptr** * خطا رخ نمی‌دهد.

```
1  // Fig. 8.13: fig08_13.cpp
2  // Attempting to modify a constant pointer to non-constant data.
3
4  int main()
5  {
6     int x, y;
7
8     // ptr is a constant pointer to an integer that can
9     // be modified through ptr, but ptr always points to the
10    // same memory location.
11    int * const ptr = &x; // const pointer must be initialized
12
13    *ptr = 7; // allowed: *ptr is not const
14    ptr = &y; // error: ptr is const; cannot assign to it a new address
15    return 0; // indicates successful termination
16 } // end main
```

Borland C++ command-line compiler error message:

```
Error E2024 fig08_13.cpp 14:Cannot modify a const object in function
main()s
```

Microsoft Visual C++ .NET compiler error message:

```
c:\cpphtp5e_examples\ch08\Fig08_13\fig08_13.cpp(14) : error C2166:
   1-value specifies const object
```

GNU C++ compiler error message:

```
Fig08_13.cpp: In function int main()':
Fig08_13.cpp:14: error: assignment of read-only variable ptr'
```

شکل ۱۳-۸ | اقدام به تغییر یک اشاره‌گر ثابت به داده غیرثابت.

خطای برنامه‌نویسی

*عدم مقداردهی اولیه اشاره‌گری که بصورت **const** اعلان شده، خطای کامپایلر بدنبال خواهد داشت.*

اشاره‌گر ثابت به داده ثابت

آخرین مجوز دسترسی توسط یک اشاره‌گر ثابت به داده ثابت اعطا می‌شود. همیشه چنین اشاره‌گری به همان موقعیت حافظه اشاره می‌کند و داده موجود در آن مکان از حافظه با استفاده از اشاره‌گر قادر به تغییر نمی‌باشد. در اینحالت است که آرایه به تابعی ارسال می‌شود و آن تابع فقط می‌تواند با استفاده از شاخص،

www.yourshop.4kia.ir

آرایه را خوانده و قادر به تغییر آرایه نخواهد بود. در برنامه شکل ۱۴-۸ یک متغیر اشاره‌گر بنام ptr از نوع const int * const اعلان شده است (خط 14). اگر این اعلان از سمت راست به چپ خوانده شود به اینصورت خواهد بود «ptr یک اشاره‌گر ثابت به یک ثابت صحیح است.» در خروجی برنامه پیغام‌های خطای تولید شده به هنگام مبادرت به اعمال تغییر در داده‌ای که ptr به آن اشاره می‌کند و (خط 18) در زمان اعمال تغییر در آدرس ذخیره شده در متغیر اشاره‌گر (خط 19) دیده می‌شود.

```
1  // Fig. 8.14: fig08_14.cpp
2  // Attempting to modify a constant pointer to constant data.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  int main()
8  {
9     int x = 5, y;
10
11    // ptr is a constant pointer to a constant integer.
12    // ptr always points to the same location; the integer
13    // at that location cannot be modified.
14    const int *const ptr = &x;
15
16    cout << *ptr << endl;
17
18    *ptr = 7; // error: *ptr is const; cannot assign new value
19    ptr = &y; // error: ptr is const; cannot assign new address
20    return 0; // indicates successful termination
21 } // end main
```

Borland C++ command-line compiler error message:

```
Error E2024 fig08_14.cpp 18: Cannot modify a const object in function main()
Error E2024 fig08_14.cpp 19: Cannot modify a const object in function main()
```

Microsoft Visual C++ .NET compiler error message:

```
c:\cpphtp5e_examples\ch08\Fig08_13\fig08_14.cpp(18) : error  C2166:
   l-value specifies const object
c:\cpphtp5e_examples\ch08\Fig08_13\fig08_14.cpp(19) : error  C2166:
   l-value specifies const object
```

GNU C++ compiler error message:

```
Fig08_14.cpp: In function 'int main()':
Fig08_14.cpp:18: error: assignment of read-only location
Fig08_14.cpp:19: error: assignment of read-only variable 'ptr'
```

شکل ۱۴-۸ | اقدام به تغییر یک اشاره‌گر ثابت به داده ثابت.

۶-۸ مرتب سازی انتخابی به روش مراجعه

در این بخش، اقدام به تعریف برنامه‌ای می‌کنیم که به توصیف نحوه ارسال آرایه‌ها و عناصر جداگانه آن به روش مراجعه می‌پردازد. از الگویتم *مرتب‌سازی انتخابی* استفاده می‌کنیم که برنامه آسانی است، اما الگوریتم آن از کارایی پایانی برخوردار است. در اولین تکرار، الگوریتم کوچکترین عنصر در آرایه را انتخاب کرده و آنرا با اولین عنصر جابجا می‌کند. در دومین تکرار، دومین عنصر کوچکتر (که کوچکترین در میان باقیمانده عناصر است) انتخاب شده و با دومین عنصر تعویض می‌شود. الگوریتم به کار خود ادامه می‌دهد تا اینکه در آخرین تکرار دومین عنصر بزرگ را انتخاب و با آنرا با شاخص دومین عنصر کوچک

www.yourshop.4kia.ir

عوض کرده، عنصر بزرگ را در آخرین شاخص نگه می‌دارد. پس از i^{th} تکرار، کوچکترین ایتم‌های i از آرایه به ترتیب صعودی مرتب خواهند شد. برای مثال به آرایه زیر توجه کنید

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

برنامه‌ای که مرتب‌سازی انتخابی را پیاده‌سازی می‌کند ابتدا مبادرت به تعیین کوچکترین عنصر (4) در این آرایه می‌کند که در عنصر دوم جای دارد. برنامه جای 4 را با عنصر صفر (34) عوض می‌کند و در نتیجه لیست زیر خواهد بود

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

سپس برنامه کوچکترین مقدار باقیمانده در میان سایر عناصر (همه عناصر بجز 4) را که 5 باشد پیدا می‌کند که در عنصر هشتم جای دارد. برنامه جای 5 را با عنصر یک (56) عوض می‌کند و نتیجه کار لیست زیر است

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

در تکرار سوم، برنامه کوچکترین مقدار بعدی را تعیین (10) و جای آنرا با عنصر دوم (34) عوض می‌کند

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

این فرآیند تا مرتب شدن کامل آرایه ادامه می‌یابد.

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

دقت کنید که پس از اولین تکرار، کوچکترین عنصر در اولین مکان جای خواهد گرفت. پس از دومین تکرار، دو عنصر کوچکتر به ترتیب در دو موقعیت اول جای خواهند گرفت. پس از سومین تکرار، سه عنصر کوچکتر به ترتیب در سه موقعیت اول جای خواهند گرفت.

برنامه شکل ۱۵-۸ مبادرت به پیاده‌سازی الگوریتم مرتب‌سازی انتخابی با استفاده از دو تابع **selectionSort** و **swap** کرده است. تابع **selectionSort** در خطوط 53-36 آرایه را مرتب می‌کند. در خط 38 متغیر **smallest** اعلان شده که شاخص کوچکترین عنصر در آرایه باقیمانده را در خود ذخیره می‌کند. خطوط 52-41 حلقه‌ای به تعداد **size - 1** بوجود می‌آورند. خط 43 مبادرت به تنظیم شاخص کوچکترین عنصر با شاخص جاری می‌کند. خطوط 49-46 حلقه‌ای بر روی باقیمانده عناصر در آرایه تدارک می‌بینند. برای هر کدامیک از این عناصر، خط 48 مقدار خود را با مقدار کوچکترین عنصر مقایسه می‌کند. اگر عنصر جاری کوچکتر از کوچکترین عنصر باشد، خط 49 مبادرت به تخصیص شاخص عنصر جاری با **smallest** می‌کند. زمانیکه این حلقه پایان پذیرد، **smallest** حاوی شاخص کوچکترین عنصر در باقیمانده آرایه خواهد بود. خط 51 تابع **swap** را برای قرار دادن کوچکترین عنصر باقیمانده در نقطه بعدی آرایه فراخوانی می‌کند، یعنی مبادله عناصر آرایه **array[smallest]**و**array[i]** .

```
1  // Fig. 8.15: fig08_15.cpp
2  // This program puts values into an array, sorts the values into
3  // ascending order and prints the resulting array.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
```

www.yourshop.4kia.ir

```
8  #include <iomanip>
9  using std::setw;
10
11 void selectionSort( int * const, const int ); // prototype
12 void swap( int * const, int * const ); // prototype
13
14 int main()
15 {
16    const int arraySize = 10;
17    int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
18
19    cout << "Data items in original order\n";
20
21    for ( int i = 0; i < arraySize; i++ )
22       cout << setw( 4 ) << a[ i ];
23
24    selectionSort( a, arraySize ); // sort the array
25
26    cout << "\nData items in ascending order\n";
27
28    for ( int j = 0; j < arraySize; j++ )
29       cout << setw( 4 ) << a[ j ];
30
31    cout << endl;
32    return 0; // indicates successful termination
33 } // end main
34
35 // function to sort an array
36 void selectionSort( int * const array, const int size )
37 {
38    int smallest; // index of smallest element
39
40    // loop over size - 1 elements
41    for ( int i = 0; i < size - 1; i++ )
42    {
43       smallest = i; // first index of remaining array
44
45       // loop to find index of smallest element
46       for ( int index = i + 1; index < size; index++ )
47
48          if ( array[ index ] < array[ smallest ] )
49             smallest = index;
50
51       swap( &array[ i ], &array[ smallest ] );
52    } // end if
53 } // end function selectionSort
54
55 // swap values at memory locations to which
56 // element1Ptr and element2Ptr point
57 void swap( int * const element1Ptr, int * const element2Ptr )
58 {
59    int hold = *element1Ptr;
60    *element1Ptr = *element2Ptr;
61    *element2Ptr = hold;
62 } // end function swap
```

```
Data item in original order
   2    6    4    8   10   12   89   68   45   37
Data item in ascending order
   2    6    4    8   10   12   37   45   68   89
```

شکل ۱۵-۸ | مرتب سازی انتخابی به روش مراجعه.

اجازه دهید نگاهی دقیق‌تر به تابع **swap** داشته باشیم. بخاطر دارید که C++ تاکید بر پنهان سازی اطلاعات مابین توابع دارد، از اینرو **swap** مجبور به دسترسی به عناصر جداگانه آرایه در **selectionSort** ندارد. چرا که **selectionSort** می‌خواهد **swap** به عناصر آرایه که جابجا یا تعویض خواهند شد دسترسی داشته

www.yourshop.4kia.ir

باشد. تابع **selectionSort** هر کدامیک از این عناصر را به **swap** به روش مراجعه ارسال می‌کند، آدرس هر عنصر آرایه بصورت صریح ارسال می‌شود. اگر چه کل آرایه‌ها به روش مراجعه ارسال می‌شوند، عناصر مجزای آرایه حالت اسکالر دارند و معمولاً به روش مقدار ارسال می‌شوند. از اینرو، تابع **selectionSort** از عملگر آدرس (&) بر روی هر یک از عناصر آرایه در فراخوانی **swap** استفاده کرده (خط 51) تا تاثیر ارسال با مراجعه بوجود آید. تابع **swap** در خطوط 62-57 مبادرت به دریافت **&array[i]** بشکل متغیر اشاره‌گر **element1Ptr** می‌کند. پنهان‌سازی اطلاعات از دانستن نام **array[i]** جلوگیری می‌کند، اما swap می‌تواند با استفاده از **element1Ptr*** بعنوان مترادفی برای **array[i]** استفاده کند.

از اینرو، زمانیکه **swap** به **element1Ptr*** مراجعه می‌کند در واقع به **array[i]** در **selectionSort** مراجعه می‌نماید. به همین ترتیب، هنگامی‌که **swap** به **element2Ptr*** مراجعه می‌کند در واقع به **array[smallest]** در **selectionSort** مراجعه می‌نماید.

ولو اینکه **swap** اجازه استفاده از عبارات

```
hold = array[i];
array[i] = array[smallest];
array[smallest] = hold;
```

را ندارد، می‌توان همان کارها را با عبارت زیر در تابع **swap** برنامه ۱۵-۸ انجام داد:

```
int hold = *element1Ptr;
*element1Ptr = *element2Ptr;
*element2Ptr = hold;
```

باید به چندین ویژگی تابع **selectionSort** توجه کنید. سرآیند تابع (خط 36) مبادرت به اعلان **array** بصورت **int *const array** بجای **int array[ ]** کرده تا نشان دهد که تابع **selectionSort** یک آرایه تک بعدی بعنوان آرگومان دریافت می‌کند. هر دو پارامتر اشاره‌گر **array** و پارامتر **size** بصورت **const** اعلان شده‌اند تا آخرین قاعده مجوز دسترسی اعمال شود. اگر چه پارامتر **size** یک کپی از مقدار در **main** را دریافت می‌کند و تغییر در کپی نمی‌تواند مقدار را در **main** تغییر دهد، **selectionSort** نیاز به تغییر دادن **size** برای انجام وظیفه خود ندارد. سایز آرایه در مدت زمان اجرای تابع **selectionSort** بلاتغییر باقی می‌ماند. از اینرو، **size** بصورت **const** اعلان شده تا مطمئن گردیم که تغییر پیدا نخواهد کرد. اگر سایز آرایه در زمان اجرای فرآیند مرتب سازی تغییر یابد، الگوریتم مرتب‌سازی بدرستی کار نخواهد کرد. توجه نمائید که تابع **selectionSort** سایز آرایه را بعنوان یک پارامتر دریافت می‌کند چرا که تابع بایستی اطلاعاتی برای مرتب‌سازی آرایه در اختیار داشته باشد. زمانیکه یک آرایه مبتنی بر اشاره‌گر به تابع ارسال می‌شود، فقط آدرس حافظه اولین عنصر آرایه توسط تابع دریافت می‌شود، و باید سایز آرایه هم بصورت مجزا به تابع ارسال گردد.

www.yourshop.4kia.ir

با تعریف تابع **selectionSort** برای دریافت سایز آرایه بعنوان یک پارامتر، تابعی خواهیم داشت که می‌توان از آن در هر برنامه‌ای که آرایه‌های **int** یک بعدی با هر سایز را مرتب می‌کند، استفاده کنیم. می‌توان سایز آرایه را مستقیماً در تابعی بصورت برنامه‌نویسی شده بدست آورد، اما اینکار می‌تواند تابع را محدود به پردازش آرایه‌ای با سایز مشخص کرده و از کارایی و استفاده مجدداً آن کم کند.

مهندسی نرم‌افزار

به هنگام ارسال آرایه به یک تابع، سایز آرایه را هم ارسال کنید. تا کارایی و استفاده مجدد از تابع افزایش یابد.

۸-۷ عملگر sizeof

زبان C++ دارای عملگر غیربایتری ***sizeof*** برای تعیین سایز یک آرایه (یا نوع داده‌های دیگر، متغیر یا ثابت) برحسب بایت در زمان کامپایل برنامه است. زمانیکه بر روی نام یک آرایه همانند شکل ۱۶-۸ (خط 14) اعمال می‌شود، این عملگر مجموع تعداد بایت‌های موجود در آرایه را بعنوان مقداری از نوع **size_t** (نام مستعار برای **unsigned int** بر روی اکثر کامپایلرها) برگشت می‌دهد. دقت کنید که این نوع متفاوت از **size** در **vector<int>** است که تعداد عناصر صحیح در بردار را مشخص می‌کند. کامپیوتری که ما از آن برای کامپایل کردن این برنامه استفاده کرده‌ایم، متغیرها از نوع **double** را در 8 بایت حافظه ذخیره می‌سازد و آرایه اعلان شده با 20 عنصر (خط 12) از 160 بایت حافظه استفاده می‌کند. زمانیکه یک پارامتر اشاره‌گر (خط 24) را در تابعی استفاده می‌کنیم که یک آرایه را بعنوان آرگومان دریافت می‌کند، عملگر **sizeof** سایز اشاره‌گر را برحسب بایت (4) و نه سایز آرایه برگشت می‌دهد.

```
1  // Fig. 8.16: fig08_16.cpp
2  // Sizeof operator when used on an array name
3  // returns the number of bytes in the array.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  size_t getSize( double * ); // prototype
9
10 int main()
11 {
12    double array[ 20 ]; // 20 doubles; occupies 160 bytes on our system
13
14    cout << "The number of bytes in the array is " << sizeof( array );
15
16    cout << "\nThe number of bytes returned by getSize is "
17       << getSize( array ) << endl;
18    return 0; // indicates successful termination
19 } // end main
20
21 // return size of ptr
22 size_t getSize( double *ptr )
23 {
24    return sizeof( ptr );
25 } // end function getSize
```

www.yourshop.4kia.ir

```
The number of bytes in the array is 160
The number of bytes returned by getSize is 4
```

شکل ١٦-٨ | اعمال عملگر sizeof بر روی نام آرایه برای برگشت دادن تعداد بایت‌ها در آرایه.

البته می‌توان تعداد عناصر در آرایه را با استفاده از نتایج بدست آمد از دوبار اعمال **sizeof** تعیین کرد. برای مثال، به اعلان آرایه زیر توجه کنید:

```
double realArray[22];
```

اگر متغیرها از نوع داده **double** در هشت بایت حافظه ذخیره شوند، آرایه **realArray** در مجموع 76 بایت خواهد بود. برای تعیین تعداد عناصر در آرایه می‌توان از عبارت زیر استفاده کرد:

```
sizeof realArray / sizeof(double) // calculate number of elements
```

این عبارت تعیین کننده تعداد بایت‌ها در آرایه **realArray** بوده و آن مقدار را بر تعداد بایت‌های بکار رفته در حافظه برای ذخیره یک مقدار **double** تقسیم کرده و نتیجه کار تعداد عناصر در **realArray** یعنی 22 خواهد بود.

تعیین سایز داده‌های بنیادین، آرایه و اشاره‌گر

در برنامه شکل ١٧-٨ از عملگر **sizeof** برای محاسبه تعداد بایت‌های بکار رفته در ذخیره انواع داده‌های استاندارد استفاده شده است. توجه کنید که، در خروجی برنامه نوع‌های **double** و **long double** دارای سایز یکسان هستند. انواع داده براساس نوع سیستم دارای سایزهای مختلفی می‌باشند.

```
// Fig. 8.17: fig08_17.cpp
// Demonstrating the sizeof operator.
#include <iostream>
using std::cout;
using std::endl;

int main()
{
   char c; // variable of type char
   short s; // variable of type short
   int i; // variable of type int
   long l; // variable of type long
   float f; // variable of type float
   double d; // variable of type double
   long double ld; // variable of type long double
   int array[ 20 ]; // array of int
   int *ptr = array; // variable of type int *

   cout << "sizeof c = " << sizeof c
      << "\tsizeof(char) = " << sizeof( char )
      << "\nsizeof s = " << sizeof s
      << "\tsizeof(short) = " << sizeof( short )
      << "\nsizeof i = " << sizeof i
      << "\tsizeof(int) = " << sizeof( int )
      << "\nsizeof l = " << sizeof l
      << "\tsizeof(long) = " << sizeof( long )
      << "\nsizeof f = " << sizeof f
      << "\tsizeof(float) = " << sizeof( float )
      << "\nsizeof d = " << sizeof d
      << "\tsizeof(double) = " << sizeof( double )
      << "\nsizeof ld = " << sizeof ld
      << "\tsizeof(long double) = " << sizeof( long double )
      << "\nsizeof array = " << sizeof array
      << "\nsizeof ptr = " << sizeof ptr << endl;
   return 0; // indicates successful termination
} // end main
```

```
sizeof  c = 1        sizeof(char) = 1
```

www.yourshop.4kia.ir

```
sizeof  s = 1        sizeof(short) = 2
sizeof  i = 4        sizeof(int) = 4
sizeof  l = 4        sizeof(long) = 4
sizeof  f = 4        sizeof(float) = 4
sizeof  d = 8        sizeof(double) = 8
sizeof  ld = 8       sizeof(long double) = 8
sizeof  array = 80
sizeof  ptr - 4
```

شکل ۱۷-۸ | عملگر sizeof در تعیین سایز انواع داده استاندارد.

می‌توان عملگر sizeof را بر روی نام هر متغیر، نام نوع یا مقدار ثابت اعمال کرد. زمانیکه sizeof بر روی نام یک متغیر یا مقدار ثابت بکار برده می‌شود (که نام یک آرایه نیست) تعداد بایت‌های بکار رفته در ذخیره سازی آن نوع متغیر یا ثابت برگشت داده می‌شود. دقت کنید که استفاده از پرانتز در sizeof در صورتی لازم خواهد بود که نام نوع بعنوان عملوند بکار رفته باشد (همانند int). استفاده از پرانتز در کنار sizeof زمانیکه عملوند sizeof نام یک متغیر یا ثابت باشد، ضروری نیست. بخاطر دارید که sizeof یک عملگر است نه تابع و به همین دلیل تاثیر آن در زمان کامپایل و نه اجرا مشخص می‌شود.

خطای برنامه‌نویسی

*نادیده گرفتن پرانتزها در **sizeof** زمانیکه عملوند نام نوع است، خطای کامپایل بدنبال خواهد داشت.*

کارائی

*بدلیل اینکه **sizeof** یک عملگر غیربایری زمان کامپایل است نه عملگر زمان اجرا، استفاده صحیح از **sizeof** تاثیر منفی در کارایی اجرا نخواهد گذاشت.*

اجتناب از خطا

*برای دوری کردن از خطای مرتبط با حذف پرانتزها در اطراف علموند عملگر **sizeof** بسیاری از برنامه نویسان ترجیح می‌دهند در اطراف هر عملوند **sizeof** پرانتز قرار دهند.*

۸-۸ عبارات اشاره‌گر و محاسبات اشاره‌گر

در عبارات محاسباتی، عبارات تخصیصی و مقایسه‌ای، حضور اشاره‌گرها بعنوان عملوند معتبر است. با این همه معمولاً حضور تمام عملگرها در چنین عباراتی با متغیرهای اشاره‌گر معتبر نمی‌باشد. در این بخش به توصیف عملگرهای می‌پردازیم که می‌توانند حضور اشاره‌گرها را بعنوان عملوند قبول کرده و همچنین به بررسی نحوه استفاده از این عملگرها در کنار اشاره‌گرها خواهیم پرداخت.

چندین عملیات محاسباتی وجود دارد که می‌توان بر روی اشاره‌گرها اعمال کرد. می‌توان یک اشاره‌گر را افزایش (++) یا کاهش (--) داد، یک مقدار صحیح به اشاره‌گر افزود (+= یا +) ، یا مقدار صحیح را از اشاره‌گر کم‌کرد (-= یا -) یا یک اشاره‌گر را از دیگری کم کرد.

فرض کنید آرایه **int v[5]** اعلان شده باشد و اولین عنصر آن در حافظه‌ی با موقعیت 3000 جای گرفته باشد. فرض کنید که اشاره‌گر **vPtr** برای اشاره به **v[0]** مقداردهی اولیه شده باشد (یعنی مقدار **vPtr** برابر

www.yourshop.4kia.ir

3000 باشد). این وضعیت در دیاگرام شکل ۱۸–۸ بر روی ماشینی نشان داده شده که مقادیر صحیح را در چهار بایت نگهداری می‌کند. توجه کنید که **vPtr** می‌تواند برای اشاره به آرایه **v** با عبارات زیر اشاره کند (چرا که نام آرایه معادل با آدرس اولین عنصر آن است):

```
int * vPtr = v;
int *vPtr = &v [0];
```

شکل ۱۸–۸ | آرایه v و متغیر اشاره‌گر vPtr که به v اشاره می‌کند.

در یک محاسبه عادی نتیجه جمع 2 + 3000 مقدار 3002 است. اما چنین جمعی در مورد محاسبات اشاره‌گر صادق نیست. زمانیکه یک مقدار صحیح به اشاره‌گر افزوده یا از آن کاسته می‌شود، اشاره‌گر به سادگی فقط به آن میزان افزایش یا کاهش نمی‌یابد بلکه آن مقدار صحیح در سایزی که اشاره‌گر به آن مراجعه دارد ضرب می‌شود. برای مثال عبارت

```
vPtr += 2;
```

مقدار 3008 را تولید می‌کند (4 * 2 + 3000) با فرض اینکه یک **int** در چهار بایت ذخیره می‌شود. هم‌اکنون در آرایه v، اشاره‌گر **vPtr** به v[2] اشاره خواهد کرد (شکل ۱۹–۸). اگر یک مقدار صحیح در دو بایت از حافظه ذخیره شود، پس محاسبه فوق حافظه‌ای با موقعیت 3004 را بدست خواهد داد (3000 + 2 * 2). اگر **vPtr** به 3016 افزایش داده شده باشد که به v[4] اشاره کند، عبارت

```
vPtr -= 4;
```

مبادرت به تنظیم **vPtr** برای برگشت به 3000 می‌کند، یعنی ابتدای آرایه. اگر اشاره‌گر در هر بار یک واحد افزایش یا کاهش یابد، می‌توان از عملگرهای افزایش دهنده (++) و کاهش‌دهنده (--) استفاده کرد.

شکل ۱۹–۸ | اشاره‌گر vPtr پس از محاسبات اشاره‌گر.

هر کدام از عبارات

```
++vPtr;
vPtr++;
```

سبب افزایش اشاره‌گر برای اشاره به عنصر بعدی در آرایه می‌شوند. هر یک از عبارات زیر

```
--vPtr;
vPtr--;
```

سبب کاهش اشاره‌گر برای اشاره به عنصر قبلی در آرایه می‌شوند.

متغیرهای اشاره‌گر، اشاره کننده به یک آرایه می‌توانند از یکدیگر کم شوند. برای مثال اگر **vPtr** حاوی موقعیت 3000 و **v2Ptr** حاوی آدرس 3008 باشد، عبارت

```
x = v2Ptr - vPtr;
```

تعداد عناصر آرایه از **vPtr** به **v2Ptr** را که در این مورد 2 است به x تخصیص می‌دهد. محاسبات اشاره‌گر در صورتیکه بر روی اشاره‌گری که به یک آرایه اشاره می‌کند اعمال نشود، بی‌معنی خواهد بود.

www.yourshop.4kia.ir

نمی‌توانیم تصور کنیم که دو متغیر از یک نوع دقیقاً در پشت سرهم در حافظه ذخیره شده باشند مگر اینکه آن دو عناصری از یک آرایه باشند.

خطای برنامه‌نویسی

استفاده از محاسبات اشاره‌گر بر روی اشاره‌گری که به مقادیر یک آرایه اشاره نمی‌کند، خطای منطقی است.

خطای برنامه‌نویسی

کاهش یا مقایسه دو اشاره‌گر که به عناصر یک آرایه اشاره نمی‌کند خطای منطقی است.

اگر هر دو اشاره‌گر از نوع یکسانی باشند، می‌توان یک اشاره‌گر را به دیگری تخصیص داد. در غیر اینصورت باید از یک عملگر **cast** برای تبدیل مقدار اشاره‌گر قرار گرفته در سمت راست تخصیص به نوع اشاره‌گر در سمت راست عملگر تخصیص استفاده کرد. استثنائی که در قانون وجود دارد در ارتباط با اشاره‌گر به **void** است (یعنی **void ***) که یک اشاره‌گر عام بوده و قادر به عرضه هر نوع اشاره‌گر می‌باشد. تمام انواع اشاره‌گر را می‌توان به اشاره‌گری از نوع **void *** تخصیص داد بدون اینکه نیازی به تبدیل باشد. با این همه، یک اشاره‌گر از نوع **void *** نمی‌تواند مستقیماً به یک اشاره‌گر از نوع دیگر تخصیص داده شود. بایستی ابتدا اشاره‌گر از نوع **void *** تبدیل به نوع اشاره‌گر مناسب و صحیح شود.

یک اشاره‌گر از نوع **void *** قادر به ردگیری نیست. برای مثال، کامپایلر می‌داند که یک اشاره‌گر به **int** به چهار بایت از حافظه بر روی ماشینی با مقادیر صحیح چهاربایتی مراجعه دارد، اما یک اشاره‌گر به **void** فقط حاوی یک آدرس حافظه برای یک نوع داده ناشناخته است. کامپایلر برای تعیین تعداد بایت نیاز به دانستن نوع داده دارد تا بتواند یک اشاره‌گر خاص را ردگیری کند.

خطای برنامه‌نویسی

*تخصیص یک اشاره‌گر از یک نوع به اشاره‌گری از نوع دیگر (بجز **void ***) بدون اینکه عمل تبدیل اشاره‌گر اول به نوع اشاره‌گر دوم صورت گرفته باشد، خطای زمان کامپایل بدنبال خواهد داشت.*

با استفاده از عملگرهای تساوی و رابطه‌ای می‌توان به مقایسه اشاره‌گرها پرداخت. مقایسه با استفاده از عملگرهای رابطه‌ای فرآیند بی‌معنی خواهد بود، مگر اینکه اشاره‌گرها در حال اشاره به عضوهای یک آرایه باشند. در مقایسه اشاره‌گرها فرآیند مقایسه آدرسهای ذخیره شده در اشاره‌گرها صورت می‌گیرد. برای مثال در مقایسه دو اشاره‌گر می‌توان تعیین کرد که آیا اشاره‌گری به عنصر بالاتری در آرایه به نسبت اشاره‌گر دیگری اشاره می‌کند یا خیر. یکی از استفاده‌های رایج از عملیات مقایسه اشاره‌گر، تعیین اشاره اشاره‌گر به صفر است (یعنی، اشاره‌گر یک اشاره‌گر ***null*** است و به چیزی اشاره نمی‌کند).

۹-۸ رابطه مابین اشاره‌گرها و آرایه‌ها

www.yourshop.4kia.ir

آرایه‌ها و اشاره‌گرها رابطه نزدیکی در C++ دارند و تقریباً می‌توان از آنها بجای یکدیگر استفاده کرد. می‌توان در مورد نام آرایه همانند یک اشاره‌گر ثابت فکر کرد. می‌توان از اشاره‌گرها برای انجام هر کاری که توسط شاخص آرایه می‌توان انجام داد، بکار گرفت. اعلان‌های زیر را در نظر بگیرید:

```
int b[5];  // create 5-element int array b
int *bPtr; // create int pointer bPtr
```

بدلیل اینکه نام آرایه (بدون شاخص) یک اشاره‌گر (ثابت) به اولین عنصر آرایه است، می‌توانیم **bPtr** را با آدرس اولین عنصر در آرایه **b** و با عبارت زیر تنظیم کنیم

```
bPtr = b; // assign address of array b to bPtr
```

این عبارت معادل دریافت آدرس اولین عنصر آرایه بصورت زیر است:

```
bPtr = &b[0]; // also assigns address of array b to bPtr
```

به عنصر **b[3]** آرایه می‌توان بطور جایگزین با عبارت اشاره‌گر زیر مراجعه کرد:

```
*(bPtr + 3)
```

عدد 3 در عبارت فوق افست به اشاره‌گر است. زمانیکه اشاره‌گر به ابتدای آرایه اشاره می‌کند، افست بر این نکته دلالت دارد که کدام عنصر آرایه باید مورد مراجعه واقع شود و مقدار افست معادل با شاخص آرایه است. از جمله فوق بعنوان *نشان‌گذاری اشاره‌گر/افست* یاد می‌شود. وجود پرانتزها ضروری است، چرا که اولویت * از + بالاتر است. بدون حضور پرانتز عبارت فوق مبادرت به افزودن 3 به مقدار **bPtr*** خواهدکرد (یعنی 3 به **b[0]** افزوده می‌شود، با فرض اینکه **bPtr** به ابتدای آرایه اشاره می‌کند). همانطوری که عنصر آرایه می‌تواند با یک عبارت اشاره‌گر مورد مراجعه قرار گیرد، آدرس

```
&b[3]
```

می‌تواند با عبارت اشاره‌گر و بصورت زیر نوشته شود

```
bPtr + 3
```

می‌توان با نام آرایه همانند یک اشاره‌گر رفتار کرد و در محاسبات اشاره‌گر بکار گرفت. برای مثال، عبارت

```
*(b + 3)
```

به عنصر **b[3]** آرایه مراجعه دارد. بطور کلی، می‌توان تمام عبارات آرایه که با شاخص انجام می‌شود را با نوشتن اشاره‌گر و یک افست انجام داد. در این مورد، نشانه‌گذاری اشاره‌گر/افست با نام آرایه بعنوان اشاره‌گر بکار گرفته شده است. دقت کنید که عبارت قبلی مبادرت به تغییر نام آرایه نمی‌کند، **b** هنوز هم به اولین عنصر در آرایه اشاره می‌کند.

همانند آرایه‌ها می‌توان اشاره‌گرها را هم شاخص‌دار کرد. برای مثال، عبارت

```
bPtr[1]
```

به عنصر **b[1]** آرایه مراجعه می‌کند، این عبارت از نشان‌گذاری اشاره‌گر/شاخص استفاده کرده است. بخاطر دارید که نام یک آرایه یک اشاره‌گر ثابت است که همیشه به ابتدای آرایه اشاره می‌کند. از اینرو عبارت

www.yourshop.4kia.ir

```
b += 3
```

سبب بوجود آمدن خطای کامپایل خواهد شد، چرا که این عبارت مبادرت به تغییر مقدار نام آرایه (یک ثابت) با محاسبات اشاره‌گر می‌کند.

خطای برنامه‌نویسی

با اینکه اسامی آرایه‌ها، اشاره‌گرهای به ابتدای آرایه هستند و اشاره‌گرها می‌توانند در عبارات محاسباتی تغییر داده شوند، اما اسامی آرایه‌ها را نمی‌توان در عبارات محاسباتی دچار تغییر کرد، چرا که اسامی آرایه‌ها از جمله اشاره‌گرهای ثابت هستند.

در برنامه شکل ۲۰-۸ از چهار نشان‌گذاری بحث شده در این بخش برای مراجعه به عناصر آرایه برای انجام یک وظیفه که چاپ چهار عناصر آرایه b از نوع صحیح باشد، استفاده شده است.

```
1  // Fig. 8.20: fig08_20.cpp
2  // Using subscripting and pointer notations with arrays.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  int main()
8  {
9     int b[] = { 10, 20, 30, 40 }; // create 4-element array b
10    int *bPtr = b; // set bPtr to point to array b
11
12    // output array b using array subscript notation
13    cout << "Array b printed with:\n\nArray subscript notation\n";
14
15    for ( int i = 0; i < 4; i++ )
16       cout << "b[" << i << "] = " << b[ i ] << '\n';
17
18    // output array b using the array name and pointer/offset notation
19    cout << "\nPointer/offset notation where "
20       << "the pointer is the array name\n";
21
22    for ( int offset1 = 0; offset1 < 4; offset1++ )
23       cout << "*(b + " << offset1 << ") = "<< *( b + offset1 )<< '\n';
24
25    // output array b using bPtr and array subscript notation
26    cout << "\nPointer subscript notation\n";
27
28    for ( int j = 0; j < 4; j++ )
29       cout << "bPtr[" << j << "] = " << bPtr[ j ] << '\n';
30
31    cout << "\nPointer/offset notation\n";
32
33    // output array b using bPtr and pointer/offset notation
34    for ( int offset2 = 0; offset2 < 4; offset2++ )
35       cout << "*(bPtr + " << offset2 << ") = "
36          << *( bPtr + offset2 ) << '\n';
37
38    return 0; // indicates successful termination
39 } // end main
```

```
Array b printed with:

Array subscript notation
b[0] =  10
b[1] =  20
b[2] =  30
b[3] =  40
Pointer/offset notation where the pointer is the array name
*(b + 0) = 10
```

www.yourshop.4kia.ir

```
*(b + 1) = 20
*(b + 2) = 30
*(b + 3) = 40
Pointer subscript notation
bPtr [0] =  10
bPtr [1] =  20
bPtr [2] =  30
bPtr [3] =  40
Pointer/offset notation
*(bPtr + 0) = 10
*(bPtr + 1) = 20
*(bPtr + 2) = 30
*(bPtr + 3) = 40
```

شکل ۲۰-۸ | مراجعه به عناصر آرایه با نام آرایه و اشاره‌گرها.

برای اینکه بهتر متوجه قابلیت تعویض آرایه‌ها و اشاره‌گرها شوید، اجازه دهید تا نگاهی به دو تابع کپی کننده رشته، **copy1** و **copy2** در برنامه ۲۱-۸ داشته باشیم. هر دو تابع مبادرت به کپی یک رشته به یک آرایه کاراکتری می‌کنند. پس از مقایسه نمونه اولیه تابع **copy1** و **copy2** توابع یکسان هستند. این توابع وظیفه یکسانی را انجام می‌دهند اما از پیاده‌سازی متفاوتی برخوردار می‌باشند.

```
1  // Fig. 8.21: fig08_21.cpp
2  // Copying a string using array notation and pointer notation.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  void copy1( char *, const char * ); // prototype
8  void copy2( char *, const char * ); // prototype
9
10 int main()
11 {
12    char string1[ 10 ];
13    char *string2 = "Hello";
14    char string3[ 10 ];
15    char string4[] = "Good Bye";
16
17    copy1( string1, string2 ); // copy string2 into string1
18    cout << "string1 = " << string1 << endl;
19
20    copy2( string3, string4 ); // copy string4 into string3
21    cout << "string3 = " << string3 << endl;
22    return 0; // indicates successful termination
23 } // end main
24
25 // copy s2 to s1 using array notation
26 void copy1( char * s1, const char * s2 )
27 {
28    // copying occurs in the for header
29    for ( int i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ )
30       ; // do nothing in body
31 } // end function copy1
32
33 // copy s2 to s1 using pointer notation
34 void copy2( char *s1, const char *s2 )
35 {
36    // copying occurs in the for header
37    for ( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ )
38       ; // do nothing in body
39 } // end function copy2
```

```
string1 = Hello
string3 = Good Bye
```

شکل ۲۱-۸ | کپی رشته با استفاده از نشان‌گذاری آرایه و اشاره‌گر.

www.yourshop.4kia.ir

تابع **copy1** (خطوط 31-26) از نشان‌گذاری شاخص آرایه برای کپی رشته در **s2** به آرایه کاراکتری **s1** استفاده کرده است. در این تابع از متغیر شمارنده **i** بعنوان شاخص آرایه استفاده شده است. سرآیند عبارت **for** (خط 29) کل عملیات کپی را انجام می‌دهد، بدنه این عبارت تهی است. سرآیند مشخص می‌کند که **i** با صفر مقداردهی اولیه شده و در هر بار تکرار حلقه یک واحد افزایش می‌یابد. شرط موجود در **for** یعنی **(s1[i] = s2[i]) != '\0'** عملیات کپی کاراکتر به کاراکتر را از **s2** به **s1** انجام می‌دهد. زمانیکه به کاراکتر *null* در **s2** برسد آنرا به **s1** تخصیص داده و حلقه خاتمه می‌پذیرد، چرا که کاراکتر **null** معادل با **'\0'** است. بخاطر دارید که مقدار یک عبارت تخصیصی مقدار تخصیص یافته به عملوند سمت چپ آن است.

تابع **copy2** در خطوط 39-34 از اشاره‌گرها و عبارت محاسباتی اشاره‌گر برای کپی رشته در **s2** به آرایه کاراکتری **s1** استفاده کرده است. مجدداً، سرآیند عبارت **for** در خط 37 کل عملیات کپی را انجام می‌دهد. سرآیند شامل مقداردهی اولیه متغیر نیست. همانند تابع **copy1** شرط، **(*s1 = *s2)! = '\0'** عملیات کپی را انجام می‌دهد. اشاره‌گر **s2** ردگیری شده و کاراکتر بدست آمده به اشاره‌گر ردگیری شده **s1** تخصیص داده می‌شود. پس شرط در تخصیص، حلقه هر دو اشاره‌گر را افزایش می‌دهد و از اینرو هر دو بترتیب به عنصر بعدی در آرایه **s1** و کاراکتری بعدی در رشته **s2** اشاره می‌کنند. زمانیکه حلقه با کاراکتر null در **s2** مواجه می‌شود، کاراکتر null به اشاره‌گر ردگیری شده s1 تخصیص داده شده و حلقه خاتمه می‌پذیرد.

آرگومان اول در هر دو تابع **copy1** و **copy2** بایستی برای نگهداری رشته موجود در آرگومان دوم به اندازه کافی بزرگ در نظر گرفته شده باشد. در غیر اینصورت احتمالاً به هنگام اقدام به نوشتن در حافظه‌ای خارج از مرزهای آرایه با خطا مواجه خواهید شد.

۱۰-۸ آرایه‌ای از اشاره‌گرها

آرایه‌ها می‌توانند حاوی اشاره‌گرها هم باشند. استفاده رایج از چنین ساختمان داده‌های بفرم آرایه‌ای از رشته‌های مبتنی بر اشاره‌گر است که گاهاً از آنها بعنوان *آرایه رشته‌ای* یاد می‌شود. هر موجودیت در آرایه یک رشته است، اما در C++ یک رشته ضرورتاً یک اشاره‌گر به اولین کاراکتر خود می‌باشد، از اینرو هر موجودیت در یک آرایه از رشته‌ها یک اشاره‌گر به اولین کاراکتر رشته می‌باشد. به اعلان آرایه رشته‌ای **suit** توجه اینکه که می‌تواند در نمایش مجموعه‌ای از کارت‌های بازی کاربرد داشته باشد:

```
const char *suit[4] =
   {"Hearts" , "Diamonds" , "Clubs" , "Spades"};
```

بخش **suit[4]** از اعلان نشان می‌دهد که آرایه چهار عنصر دارد. بخش **const char *** از اعلان بر این نکته تاکید دارد که هر عنصر از آرایه suit از نوع «اشاره‌گر به داده ثابت char است.» چهار مقدار جای گرفته در آرایه عبارتند از "Hearts"، "Diamonds"، "Clubs" و "Spades". هر کدامیک از آنها در آرایه

www.yourshop.4kia.ir

بعنوان یک رشته کاراکتری خاتمه یافته با null ذخیره شده‌اند که یک کاراکتر بزرگتر از تعداد کاراکترهای مابین گوتیشن‌ها است. چهار رشته به‌ترتیب دارای طولهای هفت، نه، شش و هفت کاراکتر (با احتساب کاراکتر خاتمه دهنده null) هستند. اگرچه بنظر می‌رسد که این رشته‌ها در آرایه **suit** جای گرفته‌اند، اما در واقع اشاره‌گرها همانند شکل ۲۲-۸ در آرایه ذخیره شده‌اند. هر اشاره‌گر به اولین کاراکتر از رشته متناظر خود اشاره می‌کند. از اینرو، حتی اگر آرایه **suit** دارای سایز ثابت باشد، دسترسی به رشته‌های کاراکتر به هر طول را ممکن می‌سازد. قابلیت انعطاف پذیری در ساختمان‌های داده یکی از توانایی‌های زبان C++ می‌باشد.

شکل ۲۲-۸ | نمایش گرافیکی از آرایه suit .

رشته‌های suit می‌توانند توسط یک آرایه دو بعدی عرضه شوند که در آن هر سطر نشاندهنده یک suit و هر ستون نشاندهنده یکی از حروف نام suit باشد. چنین ساختمان داده‌ای بایستی تعداد ثابت ستون برای هر سطر داشته باشد و آن عدد بایستی بقدر کافی از بزرگترین رشته بزرگ‌تر باشد. بنابر این به هنگام ذخیره تعداد زیادی رشته که اکثر آنها کوتاهتر از بزرگترین رشته هستند، حافظه قابل ملاحظه‌ای به هدر می‌رود. در بخش بعدی با استفاده از آرایه‌ای از رشته‌ها مبادرت به پیاده سازی بازی کارت خواهیم کرد.

معمولاً آرایه‌ای از رشته‌ها به همراه آرگومان‌های خط فرمان بکار گرفته می‌شوند تا در زمان شروع برنامه به تابع **main** ارسال گردند. چنین آرگومان‌های بدنبال اجرای یک برنامه از خط فرمان آورده می‌شوند. غالباً از آرگومان‌های خط فرمان برای ارسال گزینه‌ها به یک برنامه استفاده می‌شود. برای مثال می‌توانیم از آرگومان خط فرمان زیر در ویندوز یاد کنیم:

```
dir /p
```

که سبب لیست شدن محتویات موجود در شاخه جاری شده و پس از پر شدن صفحه، مکث می‌کند. زمانیکه دستور **dir** اجرا می‌شود گزینه **p** به **dir** به عنوان یک آرگومان خط دستور ارسال می‌شود. چنین آرگومان‌های در یک آرایه رشته‌ای که **main** بعنوان آرگومان دریافت می‌کند جای داده می‌شوند.

۱۱-۸ مبحث آموزشی: بازی کارت

در این بخش از روش تولید اعداد تصادفی برای ایجاد یک بازی برزدن کارت و برنامه شبیه‌سازی تقسیم کارت استفاده خواهیم کرد. سپس می‌توان از چنین برنامه‌ای در ایجاد برنامه‌های بازی خاص کارت استفاده کرد. به منظور آشکار شدن برخی از مشکلات کارایی، بطور دانسته از الگوریتم‌های بهینه شده بر زدن و تقسیم کارت استفاده کرده‌ایم.

با استفاده از روش *بالا به پایین، اصلاح گام به گام*، برنامه‌ای ایجاد خواهیم کرد که یک دسته کارت 52 تایی را بر زده و سپس آنها را تقسیم می‌کند. از یک آرایه 4 در 13 دو بعدی بنام **deck** برای نمایش یک دسته کارت بازی استفاده می‌کنیم (شکل ۲۳-۸). سطرها متناظر با مجموعه و به ترتیب سطر صفر با

www.yourshop.4kia.ir

Hearts، سطر یک با Diamond، سطر دو با Club و سطر سوم با Spade هستند. ستون‌ها متناظر با مقادیر موجود بر روی کارت‌ها هستند، ستون‌ها از 0 تا 9 نشاندهنده مقادیر از 1 (Ace یا تک خال) تا 10 بوده و ستون‌ها از 10 الی 12 به‌ترتیب نشاندهنده Jack (سرباز)، Queen (ملکه) و King (شاه) می‌باشند.

شکل ۲۳-۸ | آرایه دوبعدی برای عرضه یک دسته کارت.

بایستی آرایه رشته‌ای **suit** را با رشته‌های کاراکتری که نشاندهنده چهار مجموعه (همانند شکل ۲۲-۸) می‌باشند پر کرده و آرایه رشته‌ای **face** را با رشته‌های کاراکتری که نشاندهنده 13 مقدار کارت‌ها می‌باشند پر نمائیم.

شبیه‌سازی برزدن یک مجموعه کارت می‌تواند بصورت زیر دنبال شود. ابتدا آرایه **deck** با صفر مقداردهی اولیه می‌شود. سپس، یک سطر از 3-0 و یک ستون از 12-0 بطور تصادفی انتخاب می‌شوند. عدد 1 در عنصر ستون [ستون] [سطر] **deck** وارد می‌شود تا نشان دهد که این کارت در اولین توزیع از برخوردن کارت پخش شده است. این فرآیند با اعداد 2,3,...,52 ادامه یافته و بصورت تصادفی در آرایه **deck** جای می‌گیرند تا نشان دهند که کدام کارت در بر دوم، سوم، ... و پنجاه‌ودوم پخش شده‌اند. همانطوری که آرایه **deck** شروع به پر شدن با اعداد کارت‌ها می‌شودد، امکان دارد که کارتی دو بار انتخاب شود (یعنی [ستون] [سطر] **deck** به هنگام انتخاب شدن صفر نباشد). چنین انتخابی نادیده گرفته می‌شود و سطرها و ستون‌ها به تکرار و بصورت تصادفی انتخاب می‌شوند تا اینکه یک کارت انتخاب نشده پیدا شود. سرانجام، اعداد 1 الی 52 مبادرت به اشغال 52 شکاف آرایه **deck** می‌کنند. در این نقطه، دسته کارت‌ها کاملاً برخورده‌اند.

اگر کارت‌های که هم اکنون بر زده شده‌اند بطور تصادفی و مکرراً انتخاب شوند، این الگوریتم بر زدن می‌تواند برای یک مدت نامحدود بکار گرفته شود. این پدیده بعنوان تعویق نامعین هم شناخته می‌شود.

پس از توزیع اولین کارت، آرایه را برای یافتن عنصر [ستون] [سطر] **deck** که با 1 مطابقت نماید جستجو می‌کنیم. اینکار با یک دستور **for** تودرتو که سطر آن از 0 تا 3 و ستون آن از 0 تا 12 تغییر می‌کند، قابل انجام است. آرایه **suit** قبلاً با چهار مجموعه پر شده است، از اینرو پس از پیدا کردن کارت، آن مجموعه را گرفته و رشته کاراکتری در [سطر] **suit** را چاپ می‌کنیم. به همین ترتیب، مقدار روی کارت را بدست آورده و رشته کاراکتری در [ستون] **face** را چاپ می‌نمائیم. همچنین رشته "of" را چاپ می‌کنیم. چاپ صحیح و به ترتیب این اطلاعات می‌تواند ما را در چاپ هر کارت بصورت "king of dubs" و "Ace of Diomads" و غیره کمک کند.

اجازه به روش از بالا به پایین، اصلاح گام به گام این برنامه را پردازش کنیم. در بالاترین سطح عبارت زیر قرار دارد

www.yourshop.4kia.ir

shuffle and deal 52 cards

اولین اصلاح صورت گرفته حاصل زیر را بدست خواهد داد:

Initialize the suit array
Initialize the face array
Initialize the deck array
Shuffle the deck
Deal 52 cards

می‌توانیم عبارت "Shuffle the deck" را بصورت زیر بسط دهیم:

For each of the 52 cards
Place card number in randemly selected unoccupied slot of deck

می‌توانیم عبارت "Deal 52 Cards" را بصورت زیر بسط دهیم:

For each of 52 cards
  Find card number in deck array and print face and suit of card

با در کنار هم قرار دادن این تحلیل‌ها مرحله دوم کامل می‌شود:

Initialize the suit array
Initialize the face array
Initialize the dack array
For each of 52 cards
  Place and number in randomly selected unoccupied slot of deck
For each of the 52 cards
  Find card number in deck array and print face and suit of card

عبارت "Place card number in randomly selected unoccupied slot of deck" می‌تواند به عبارات زیر بسط داده شود:

choose slot of deck randomly
while chosen slot of deck has been previously chosen
  choose slot of deck randomly
place card number in chosen slot of deck

عبارت "Find card number in deck array and print face and suit of card" می‌تواند به عبارات زیر بسط داده شود:

For each of the 52 cards
  If slot contaius card number
    Print the face and suit of the card

با همکاری این عبارات بسط یافته، سومین مرحله اصلاح همانند شکل ۲۴–۸ بدست می‌آید، که فرآیند اصلاح را کامل می‌کند. برنامه‌های شکل ۲۵–۸ الی ۲۷–۸ حاوی برنامه بر زدن و توزیع کارت و یک اجرای نمونه هستند. خطوط 61-67 از تابع **deal** (شکل ۲۶–۸) پیاده سازی کننده خطوط 1-2 از برنامه ۲۴–۸ می‌باشند. سازنده در خطوط 22-35 از شکل ۲۶–۸ پیاده‌کننده خطوط 1-3 از شکل ۲۴–۸ می‌باشد. تابع **shuffle** (خطوط 38-55 از شکل ۲۶–۸) پیاده‌کننده خطوط 5-11 از شکل ۲۴–۸ می‌باشد. تابع **deal** (خطوط 58-88 از شکل ۲۶–۸) پیاده کننده خطوط 13-16 از شکل ۲۴–۸ می‌باشد. به فرمت خروجی بکار رفته در تابع **deal** دقت کنید (خطوط 81-83 از شکل ۲۶–۸).

Initialize the suit array
Initialize the face array
Initialize the deck array

For each of 52 cards

www.yourshop.4kia.ir

Choose slot of deck randomly

While slot of deck has been previously chosen
Choose slot of deck randomly

Place card number in chosen slot of deck

For each of the 52 cards
For each of the 52 cards
If slot contaius card number
Print the face and suit of the card

شکل ۲٤-۸ | الگوریتم شبکه متعلق به برنامه توزیع و بر زدن کارت.

```
1  // Fig. 8.25: DeckOfCards.h
2  // Definition of class DeckOfCards that
3  // represents a deck of playing cards.
4
5  // DeckOfCards class definition
6  class DeckOfCards
7  {
8  public:
9     DeckOfCards(); // constructor initializes deck
10    void shuffle(); // shuffles cards in deck
11    void deal(); // deals cards in deck
12 private:
13    int deck[ 4 ][ 13 ]; // represents deck of cards
14 }; // end class DeckOfCards
```

شکل ۲۵-۸ | فایل سرآیند DeckOfCards.

```
1  // Fig. 8.26: DeckOfCards.cpp
2  // Member-function definitions for class DeckOfCards that simulates
3  // the shuffling and dealing of a deck of playing cards.
4  #include <iostream>
5  using std::cout;
6  using std::left;
7  using std::right;
8
9  #include <iomanip>
10 using std::setw;
11
12 #include <cstdlib> // prototypes for rand and srand
13 using std::rand;
14 using std::srand;
15
16 #include <ctime> // prototype for time
17 using std::time;
18
19 #include "DeckOfCards.h" // DeckOfCards class definition
20
21 // DeckOfCards default constructor initializes deck
22 DeckOfCards::DeckOfCards()
23 {
24    // loop through rows of deck
25    for ( int row = 0; row <= 3; row++ )
26    {
27       // loop through columns of deck for current row
28       for ( int column = 0; column <= 12; column++ )
29       {
30          deck[ row ][ column ] = 0; // initialize slot of deck to 0
31       } // end inner for
32    } // end outer for
33
34    srand( time( 0 ) ); // seed random number generator
35 } // end DeckOfCards default constructor
36
37 // shuffle cards in deck
38 void DeckOfCards::shuffle()
39 {
```

www.yourshop.4kia.ir

```
   int row; // represents suit value of card
   int column; // represents face value of card

   // for each of the 52 cards, choose a slot of the deck randomly
   for ( int card = 1; card <= 52; card++ )
   {
      do // choose a new random location until unoccupied slot is found
      {
         row = rand() % 4; // randomly select the row
         column = rand() % 13; // randomly select the column
      } while( deck[ row ][ column ] != 0 ); // end do...while

      // place card number in chosen slot of deck
      deck[ row ][ column ] = card;
   } // end for
} // end function shuffle

// deal cards in deck
void DeckOfCards::deal()
{
   // initialize suit array
   static const char *suit[ 4 ] =
      { "Hearts", "Diamonds", "Clubs", "Spades" };

   // initialize face array
   static const char *face[ 13 ] =
      { "Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven",
      "Eight", "Nine", "Ten", "Jack", "Queen", "King" };

   // for each of the 52 cards
   for ( int card = 1; card <= 52; card++ )
   {
      // loop through rows of deck
      for ( int row = 0; row <= 3; row++ )
      {
         // loop through columns of deck for current row
         for ( int column = 0; column <= 12; column++ )
         {
            // if slot contains current card, display card
            if ( deck[ row ][ column ] == card )
            {
               cout << setw( 5 ) << right << face[ column ]
                  << " of " << setw( 8 ) << left << suit[ row ]
                  << ( card % 2 == 0 ? '\n' : '\t' );
            } // end if
         } // end innermost for
      } // end inner for
   } // end outer for
} // end function deal
```

شکل ۲۶-۸ | تعریف توابع عضو برای برزدن و توزیع کارت.

عبارت خروجی، کارت‌های چهره را با ترازبندی از راست در میدانی به طول پنج کاراکتر و کارت‌های suit را با ترازبندی از چپ در میدانی به طول هشت کاراکتر چاپ می‌کند (شکل ۲۷-۸). خروجی در فرمت دو ستونی چاپ می‌شود.

الگوریتم توزیع کارت دارای یک ضعف است. زمانیکه مطابقتی پیدا می‌شود، حتی اگر این مطابقت در اولین سعی پیدا شود، عبارت **for** تودرتو به جستجو در میان عناصر باقیمانده از کارت‌ها ادامه می‌دهد تا مطابقتی پیدا کند.

```
// Fig. 8.27: fig08_27.cpp
// Card shuffling and dealing program.
#include "DeckOfCards.h" // DeckOfCards class definition

```

www.yourshop.4kia.ir

```
5  int main()
6  {
7     DeckOfCards deckOfCards; // create DeckOfCards object
8
9     deckOfCards.shuffle(); // shuffle the cards in the deck
10    deckOfCards.deal(); // deal the cards in the deck
11    return 0; // indicates successful termination
12 } // end main
```

```
Nine of Spades          Seven of Clubs
Five of Spades          Eight of Clubs
Queen of Diamonds       three of Hearts
Jack of Spades          Five of Diamonds
Jack of Diamonds        Three of Diamonds
Three of Clubs          Six of Clubs
Ten of Clubs            Nine of Diamonds
Ace of Hearts           Queen of Heart
Seven of Spades         Deuce of Spades
Six of Hearts           Deuce of Clubs
Ace of Clubs            Deuce of Diamonds
Nine of Hearts          Seven of Diamonds
Six of Spades           Eight of Diamonds
Ten of Spades           King of Hearts
Four of Clubs           Ace of Spades
Ten of Hearts           Four of Spades
Eight of Hearts         Eight of Spades
Jack of Hearts          Ten of Diamonds
Four of Diamonds        king of Diamonds
Seven of Hearts         King of Spades
Queen of Spades         Four of Hearts
Nine of Clubs           Six of Diamonds
Deuce of Hearts         Jack of Diamonds
King of Clubs           Three of Spades
Queen of Clubs          Five of Clubs
Five of Hearts          Ace of Diamonds
```

شکل ۲۷-۸ | برنامه برزدن و توزیع کارت.

۱۲-۸ اشاره‌گرهای تابع

یک اشاره‌گر به تابع حاوی آدرس تابع در حافظه است. در فصل هفتم، مشاهده کردید که نام یک آرایه در واقع آدرسی در حافظه است که نشاندهنده اولین عنصر آرایه می‌باشد. به همین ترتیب، نام یک تابع در واقع آدرس شروع کدی در حافظه است که وظیفه تابع را به انجام می‌رساند. اشاره‌گرها به توابع می‌توانند به توابع ارسال، از توابع برگشت داده شوند، در آرایه‌ها ذخیره شده و به دیگر اشاره‌گرهای توابع تخصیص داده شوند.

مرتب سازی انتخابی با استفاده از اشاره‌گرهای تابع

برای توضیح نحوه استفاده از اشاره‌گرهای تابع، برنامه شکل ۲۸-۸ را که تغییر یافته برنامه مرتب‌سازی انتخابی است را بکار گرفته‌ایم. برنامه ۲۸-۸ حاوی **main** (خطوط 55-17) و توابع **selectionSort** در خطوط 76-59، **swap** در خطوط 85-80، **ascending** در خطوط 92-89 و **descending** در خطوط 96-99 است. تابع **selectionSort** یک اشاره‌گر به تابع دریافت می‌کند (خواه تابع **ascending** باشد یا تابع **descending**). توابع **ascending** و **descending** تعیین‌کننده ترتیب مرتب‌سازی هستند (صعودی یا نزولی). برنامه به کاربر اعلان می‌کند تا ترتیب مرتب‌سازی آرایه را مشخص کند (خطوط 26-24). اگر کاربر عدد 1 را وارد سازد، اشاره‌گر به تابع **ascending** به تابع **selectionSort** (خط 37) ارسال می‌شود و سبب می‌گردد تا آرایه بصورت صعودی مرتب شود. اگر کاربر عدد 2 را وارد سازد، اشاره‌گر به تابع

www.yourshop.4kia.ir

descending به تابع **selectionSort** (خط 45) ارسال می‌شود و سبب می‌شود تا آرایه بصورت نزولی مرتب گردد.

پارامتر زیر در خط 60 از سرآیند تابع **selectionSort** ظاهر شده است:

```
bool ( *compare )( int, int )
```

این پارامتر تصریح‌کننده یک اشاره‌گر به تابع است. کلمه کلیدی **bool** بر این نکته دلالت دارد که تابع اشاره کننده یک مقدار بولی برگشت خواهد داد. کلمه **(*compare)** نشاندهنده نام اشاره‌گر به تابع است (* نشان می‌دهد که پارامتر **compare** یک اشاره‌گر است). عبارت **(int , int)** نشان می‌دهد که تابع اشاره شده برای مقایسه، دو آرگومان صحیح دریافت می‌کند. وجود پرانتزها در اطراف **compare*** لازم بوده و نشان می‌دهند که **compare** یک اشاره‌گر به تابع است. اگر پرانتزها را حذف کنیم، اعلان بصورت زیر در می‌آید:

```
bool *compare( int, int )
```

تابعی اعلان می‌شود که دو آرگومان صحیح بعنوان پارامتر دریافت و یک اشاره‌گر به یک مقدار بولی برگشت خواهد داد.

```
1  // Fig. 8.28: fig08_28.cpp
2  // Multipurpose sorting program using function pointers.
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 // prototypes
12 void selectionSort( int [], const int, bool (*)( int, int ) );
13 void swap( int * const, int * const );
14 bool ascending( int, int ); // implements ascending order
15 bool descending( int, int ); // implements descending order
16
17 int main()
18 {
19    const int arraySize = 10;
20    int order; // 1 = ascending, 2 = descending
21    int counter; // array index
22    int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
23
24    cout << "Enter 1 to sort in ascending order,\n"
25       << "Enter 2 to sort in descending order: ";
26    cin >> order;
27    cout << "\nData items in original order\n";
28
29    // output original array
30    for ( counter = 0; counter < arraySize; counter++ )
31       cout << setw( 4 ) << a[ counter ];
32
33    // sort array in ascending order; pass function ascending
34    // as an argument to specify ascending sorting order
35    if ( order == 1 )
36    {
37       selectionSort( a, arraySize, ascending );
38       cout << "\nData items in ascending order\n";
39    } // end if
40
41    // sort array in descending order; pass function descending
```

www.yourshop.4kia.ir

```
   // as an argument to specify descending sorting order
   else
   {
      selectionSort( a, arraySize, descending );
      cout << "\nData items in descending order\n";
   } // end else part of if...else

   // output sorted array
   for ( counter = 0; counter < arraySize; counter++ )
      cout << setw( 4 ) << a[ counter ];

   cout << endl;
   return 0; // indicates successful termination
} // end main

// multipurpose selection sort; the parameter compare is a pointer to
// the comparison function that determines the sorting order
void selectionSort( int work[], const int size,
                    bool (*compare)( int, int ) )
{
   int smallestOrLargest; // index of smallest (or largest) element

   // loop over size - 1 elements
   for ( int i = 0; i < size - 1; i++ )
   {
      smallestOrLargest = i; // first index of remaining vector

      // loop to find index of smallest (or largest) element
      for ( int index = i + 1; index < size; index++ )
         if ( !(*compare)( work[ smallestOrLargest ], work[ index ] ) )
            smallestOrLargest = index;

      swap( &work[ smallestOrLargest ], &work[ i ] );
   } // end if
} // end function selectionSort

// swap values at memory locations to which
// element1Ptr and element2Ptr point
void swap( int * const element1Ptr, int * const element2Ptr )
{
   int hold = *element1Ptr;
   *element1Ptr = *element2Ptr;
   *element2Ptr = hold;
} // end function swap

// determine whether element a is less than
// element b for an ascending order sort
bool ascending( int a, int b )
{
   return a < b; // returns true if a is less than b
} // end function ascending

// determine whether element a is greater than
// element b for a descending order sort
bool descending( int a, int b )
{
   return a > b; // returns true if a is greater than b
} // end function descending
```

```
Enter1 to sort in ascending order,
Enter 2to sort in descending order: 1

Data item in original order
   2    6    4    8    10    12    89    68    45    37
Data item in ascending order
   2    4    6    8    10    12    37    45    68    89
```

```
Enter1 to sort in ascending order,
Enter 2to sort in descending order: 2
```

www.yourshop.4kia.ir

```
Data item in original order
    2    6    4    8   10    12    89    68    45    37
Data item in descending order
   89   68   45   37   12    10     8     6     4     2
```

شکل ۲۸-۸ | برنامه مرتب‌سازی با استفاده از اشاره‌گرهای تابع.

پارامتر متناظر در نمونه اولیه تابع **selectionSort** بصورت زیر است

```
bool (*)( int, int)
```

دقت کنید که فقط نوع‌ها در نظر گرفته شده‌اند. مانند همیشه و فقط با هدف مستندسازی، برنامه‌نویس می‌تواند اسامی که توسط کامپایلر نادیده گرفته می‌شوند را هم وارد کند.

تابع ارسالی به **selectionSort** در خط 71 و با عبارت زیر فراخوانی می‌شود:

```
( *compare )( work[ smallestOrLargest ], work[ index ])
```

همانطوری‌که یک اشاره‌گر به یک متغیر ردگیری می‌شود تا به مقدار متغیر دسترسی پیدا شود، یک اشاره‌گر به یک تابع هم ردگیری می‌شود تا تابع را به اجرا در آورد. وجود پرانتزها در اطراف **compare*** ضروری است. اگر این پرانتزها بکار گرفته نشوند، عملگر* مبادرت به ردگیری مقدار برگشتی از فراخوان تابع خواهد کرد.

آرایه‌ای از اشاره‌گرها به توابع

یکی از کاربردهای اشاره‌گر در سیستم‌های متکی بر منو است. برای مثال، برنامه‌ای می‌تواند به کاربر اعلان کند تا گزینه مورد نظر خود را از طریق یک منو با وارد ساختن یک مقدار صحیح انتخاب نماید. از انتخاب کاربر می‌توان بعنوان شاخص در آرایه‌ای از اشاره‌گرهای تابع استفاده کرده و از آن اشاره‌گر برای فراخوانی تابع استفاده کرد.

برنامه شکل ۲۹-۸ یک مثال عادی است که به توصیف اعلان و استفاده از یک آرایه اشاره‌گرها به توابع می‌پردازد. در این برنامه سه تابع بنام‌های **function0**، **function1** و **function2** تعریف شده است که هر یک آرگومان صحیح دریافت و مقداری برگشت نمی‌دهند. خط 17 مبادرت به ذخیره اشاره‌گرهای به این سه تابع در آرایه **f** می‌کند. در این مورد هر سه تابع که آرایه به آنها اشاره می‌کند بایستی دارای نوع برگشتی و پارامترهای یکسان باشند. اگر اعلان بکار رفته در خط 17 را از سمت چپ‌ترین پرانتز بخوانیم به اینصورت خواهد بود «**f** یک آرایه از سه اشاره‌گر به توابعی است که هر یک آرگومانی از نوع **int** دریافت و **void** برگشت می‌دهند.»

```
1  // Fig. 8.29: fig08_29.cpp
2  // Demonstrating an array of pointers to functions.
3  #include <iostream>
4  using std::cout;
5  using std::cin;
6  using std::endl;
7
8  // function prototypes -- each function performs similar actions
9  void function0( int );
10 void function1( int );
11 void function2( int );
```

www.yourshop.4kia.ir

```
12
13 int main()
14 {
15    // initialize array of 3 pointers to functions that each
16    // take an int argument and return void
17    void (*f[ 3 ])( int ) = { function0, function1, function2 };
18
19    int choice;
20
21    cout << "Enter a number between 0 and 2, 3 to end: ";
22    cin >> choice;
23
24    // process user's choice
25    while ( ( choice >= 0 ) && ( choice < 3 ) )
26    {
27       // invoke the function at location choice in
28       // the array f and pass choice as an argument
29       (*f[ choice ])( choice );
30
31       cout << "Enter a number between 0 and 2, 3 to end: ";
32       cin >> choice;
33    } // end while
34
35    cout << "Program execution completed." << endl;
36    return 0; // indicates successful termination
37 } // end main
38
39 void function0( int a )
40 {
41    cout << "You entered " << a << " so function0 was called\n\n";
42 } // end function function0
43
44 void function1( int b )
45 {
46    cout << "You entered " << b << " so function1 was called\n\n";
47 } // end function function1
48
49 void function2( int c )
50 {
51    cout << "You entered " << c << " so function2 was called\n\n";
52 } // end function function2
```

```
Enter a number between 0 and 2, 3 to end: 0
You entered 0 so function0 was called

Enter a number between 0 and 2, 3 to end: 1
You entered 1 so function1 was called

Enter a number between 0 and 2, 3 to end: 2
You entered 2 so function2 was called

Enter a number between 0 and 2, 3 to end: 3
Program execution completed.
```

شکل ۲۹-۸ | آرایه‌ای از اشاره‌گرها به توابع.

آرایه با اسامی سه تابع (که اشاره‌گر هستند) مقداردهی اولیه شده است. برنامه به کاربر اعلان می‌کند تا عددی مابین 0 و 2 یا برای خاتمه برنامه عدد 3 را وارد سازد. زمانیکه کاربر مقداری مابین 0 و 2 وارد سازد، از این مقدار بعنوان شاخص در آرایه‌ای از اشاره‌گرها به توابع استفاده خواهد شد. خط 29 یکی از توابع موجود در آرایه **f** را فعال می‌سازد. در فرآیند فراخوانی **f[choice]** اشاره‌گری در مکان **choice** آرایه را انتخاب می‌کند. اشاره‌گر برای فراخوانی تابع ردگیری شده و **choice** بعنوان آرگومان به تابع

www.yourshop.4kia.ir

ارسال می‌گردد. هر تابع مبادرت به چاپ مقدار آرگومان و نام خود می‌کند تا نشان دهد که تابع بدرستی فراخوانی شده است.

۸-۱۳ پردازش رشته‌های مبتنی بر اشاره‌گر

در این بخش، به توضیح برخی از توابع کتابخانه استاندارد C++ که در ارتباط با پردازش رشته هستند، می‌پردازیم. تکنیک‌های مطرح شده در این بخش مناسب ایجاد برنامه‌های ویرایشگر متن، لغت، نرم‌افزار صفحه‌بندی، کامپیوتری کردن سیستم‌های تایپ و انواع نرم‌افزارهای مرتبط با پردازش متن می‌باشد. در حال حاضر از کلاس **string** کتابخانه استاندارد C++ در چندین مثال استفاده کرده‌ایم. برای نمونه، کلاس **GradeBook** در مبحث آموزشی فصل‌های ۳ الی ۷ با استفاده از شی **string** نام دوره‌ای را عرضه می‌کردند. در فصل هیجدهم با جزئیات کلاس **string** آشنا خواهید شد. اگرچه استفاده از شی‌های **string** معمولاً سرراست است، در این بخش از رشته‌های مبتنی بر اشاره‌گر و خاتمه یافته با **null** استفاده خواهیم کرد. تعدادی از توابع کتابخانه استاندارد C++ فقط با رشته‌های خاتمه یافته با **null** و مبتنی بر اشاره‌گر عمل می‌کنند که به نسبت شی‌های **string** پیچیده‌تر هستند. همچنین اگر با برنامه‌های قدیمی C++ کار می‌کنید، احتمالاً نیاز به دستکاری کردن این رشته‌های مبتنی بر اشاره‌گر خواهید داشت.

۸-۱۳-۱ اصول کاراکترها و رشته‌های مبتنی بر اشاره‌گر

کاراکترها بلوک‌های سازنده بنیادین در برنامه‌های منبع C++ هستند. هر برنامه‌ای متشکل از دنباله‌ای از کاراکترها است که وقتی در کنار هم قرار داده می‌شوند مفهوم و معنی پیدا می کنند و می‌توانند توسط کامپایلر بعنوان مجموعه‌ای از دستورات برای انجام وظیفه‌ای، تفسیر گردند. یک برنامه می‌تواند حاوی کاراکترهای ثابت باشد. یک کاراکتر ثابت یک مقدار صحیح است که بعنوان یک کاراکتر در میان علامت نقل قول جای می‌گیرد. مقدار یک کاراکتر ثابت یک مقدار صحیح از کاراکتر در مجموعه کاراکتری کامپیوتر می‌باشد. برای مثال، '2' نشاندهنده مقدار صحیح 2 (عدد 122 در مجموعه کاراکتری ASCII) و '\n' نشاندهنده مقدار خط جدید (عدد 10 در مجموعه کاراکتری ASCII) است.

یک رشته دنباله‌ای از کاراکترها است که با آنها همانند یک یونیت واحد رفتار می‌شود. رشته می‌تواند حاوی حروف، ارقام و کاراکترهای خاص همانند + ، - ، * ، / و $ باشد. رشته‌های لیترال یا رشته‌های ثابت در C++ در میان جفت گوتیشن نوشته می‌شوند، همانند

```
"John Q.Doe"   (نام)
"999 Main Street"   (آدرس خیابان)
"Maynard, Massachusetts" (شهر و ایالت)
"(201) 555-1212"   (شماره تلفن)
```

یک رشته مبتنی بر اشاره‌گر در C++ آرایه‌ای از کاراکترها است که با کاراکتر null خاتمه می‌پذیرند (**'\n'**) ، که انتهای رشته در حافظه را مشخص می‌سازد. در دسترسی به یک رشته توسط اشاره‌گر، به اولین

www.yourshop.4kia.ir

کاراکتر آن دست یافته می‌شود. مقدار یک رشته، آدرس اولین کاراکتر آن است. از اینرو در C++، می‌توان گفت که یک رشته یک اشاره‌گر ثابت است، در واقع یک اشاره‌گر به اولین کاراکتر رشته می‌باشد. در چنین حالتی، رشته‌ها همانند آرایه‌ها هستند چرا که نام آرایه هم یک اشاره‌گر به اولین عنصر آن است.

می‌توان از یک رشته لیترال بعنوان یک مقداردهی کننده در اعلان یک آرایه کاراکتری یا متغیری از نوع **char *** استفاده کرده در اعلان‌های

```
char color[] = "blue";
const char *colorPtr = "blue";
```

هر کدامیک اقدام به مقداردهی یک متغیر با رشته"blue" می‌کند. در اعلان اول، یک آرایه پنج عنصری بنام **color** حاوی کاراکترهای 'e'، 'a'، 'l'، 'b' و'\n' ایجاد می‌شود. اعلان دوم یک متغیر اشاره‌گر بنام **colorPtr** ایجاد می‌کند که به حرف b از رشته "blue" در جائی از حافظه اشاره دارد (که با '**\n**' خاتمه می‌یابد). رشته‌های لیترال دارای کلاس ذخیره سازی **static** هستند (در مدت زمان اجرای برنامه وجود خواهند داشت) و اگر همان رشته لیترال از مکان‌های مختلف برنامه مورد مراجعه قرار گیرد هم می‌تواند و هم نمی‌تواند به اشتراک گذاشته شود. همچنین رشته‌های لیترال، در C++ ثابت هستند و کاراکترهای آنها قابل تغییر نمی‌باشد.

اعلان **char color[ ] = "blue";** می‌تواند بصورت زیر هم نوشته شود

```
char color[] = {'b', 'l', 'u', 'e', '\n'};
```

در زمان اعلان یک آرایه کاراکتری که حاوی رشته خواهد بود، بایستی آرایه به میزان کافی برای ذخیره سازی رشته و کاراکتر null آن، بزرگ باشد. در اعلان فوق سایز آرایه براساس تعداد موجود در لیست مقداردهی اولیه مشخص است.

خطای برنامه‌نویسی

*عدم تخصیص فضای کافی در یک آرایه کاراکتری برای ذخیره کاراکتر **null** که پایان دهنده یک رشته است، خطا خواهد بود.*

خطای برنامه‌نویسی

*ایجاد یا استفاده از یک رشته مبتنی بر C یا C-style که حاوی کاراکتر **null** نمی‌باشد، می‌تواند خطاهای منطقی بدنبال داشته باشد.*

اجتناب از خطا

در زمان ذخیره سازی رشته‌ای از کاراکترها در یک آرایه کاراکتری، مطمئن شوید که آرایه به میزان کافی فضا برای نگهداری بزرگترین رشته‌ای که در آن ذخیره خواهد شد، در اختیار دارد. C++ به رشته‌ها به هر طولی اجازه ذخیره شدن می‌دهد. اگر رشته‌ای طولانی‌تر از آرایه کاراکتری باشد که در آن ذخیره خواهد شد،

www.yourshop.4kia.ir

کاراکترهای قرار گرفته در آن سوی مرز آرایه بر روی داده‌های موجود در حافظه پس از آرایه بازنویسی می‌شوند و این عمل می‌تواند خطاهای منطقی بوجود آورد.

یک رشته می‌تواند بدون یک آرایه کاراکتری با استفاده از **cin** خوانده شود. برای مثال، از عبارت زیر می‌توان برای خواندن یک رشته بدرون آرایه **word[20]** استفاده کرد:

```
cin >> word;
```

رشته وارد شده توسط کاربر در **word** ذخیره می‌شود. عبارت فوق مبادرت به خواندن کاراکترها تا رسیدن به یک کاراکتر فاصله یا شاخص انتهای فایل می‌کند. توجه کنید که رشته نبایستی بیش از 19 کاراکتر طول داشته باشد تا مکانی هم برای کاراکتر null بکار گرفته شود، از دستور **setw** می‌توان برای مطمئن شدن از اینکه رشته خوانده شده به **word** از سایز آرایه تجاوز نکرده استفاده کرد. برای مثال، عبارت

```
cin >> setw(20) >> word;
```

مشخص می‌کند که **cin** بایستی حداکثر 19 کاراکتر بدرون آرایه **word** بخواند و در مکان بیستم آرایه، کاراکتر خاتمه‌دهنده null را برای رشته ذخیره نماید. دستور **setw** فقط پس از ورودی مقدار عمل می‌کند. اگر بیش از 19 کاراکتر وارد شده باشد، مابقی کاراکترها در آرایه **word** ذخیره نمی‌شوند، اما خوانده شده و می‌توانند در متغیر دیگری ذخیره گردند.

در برخی از مواقع مناسب خواهد بود تا کل یک خط بعنوان ورودی در آرایه وارد گردد. به همین منظور C++ تابع **cin.getline** در سرآیند فایل <**iostream**> را در نظر گرفته است. در فصل سوم به معرفی تابع مشابهی بنام **getline** از سرآیند فایل <**string**> پرداختیم، که ورودی را تا مواجه شدن با یک کاراکتر خط جدید خوانده و آنرا در یک رشته مشخص شده بعنوان آرگومان ذخیره می‌سازد (البته بدون کاراکتر خط جدید). تابع **cin.getline** سه آرگومان دریافت می‌کند، یک آرایه کاراکتری که خطی از متن در آن ذخیره می‌شود، طول و کاراکتر نشاندهنده انتهای داده یا حائل. برای مثال، در بخشی از برنامه زیر

```
char sentence[80];
cin.getline( sentence, 80, '\n' );
```

آرایه **sentence** هشتاد کاراکتری را اعلان و یک خط از متن را از صفحه کلید بدرون آرایه می‌خواند. تابع تا رسیدن به کاراکتر حائل '**\n**' به خواندن کاراکترها ادامه می‌دهد و زمانیکه تعیین کننده انتهای فایل وارد شد یا تعداد کاراکترها خوانده شده یک‌ی بیش از طول تعیین شده در آرگومان دوم گردید، خواندن متوقف می‌شود (کاراکتر آخر در آرایه برای کاراکتر null رزرو شده است). اگر با کاراکتر حائل برخورد کند آنرا خوانده و حذف می‌نماید. آرگومان سوم در **cin.getline** آرگومان '**\n**' است که مقدار پیش‌فرض است، از اینرو فراخوانی تابع فوق می‌توانست بصورت زیر نوشته شود:

```
cin.getline( sentence, 80 );
```

در فصل پانزدهم با جزئیات عملکرد **cin.getline** و دیگر توابع ورودی/خروجی آشنا خواهید شد.

www.yourshop.4kia.ir

خطای برنامه‌نویسی

ارسال یک رشته بعنوان آرگومان به تابعی که انتظار یک کاراکتر را دارد، خطای کامپایل است.

۸-۱۳-۲ توابع دستکاری کننده رشته

توابع مرتبط با رشته در کتابخانه توابع، توابع مناسبی برای دستکاری داده‌های رشته‌ای، مقایسه رشته‌ها، جستجو رشته‌ها برای یافتن کاراکترهای خاص و دیگر رشته‌ها، نشانه‌گذاری رشته‌ها (مجزا کردن رشته‌ها به قسمتهای منطقی بعنوان کلمات مجزا در یک جمله) و تعیین طول رشته‌ها، هستند. در این بخش به معرفی تعدادی از توابع پرکاربرد در زمینه دستکاری رشته‌ها از کتابخانه استاندارد C++ خواهیم پرداخت. در جدول شکل ۳۰-۸ این توابع آورده شده‌اند و از آنها در مثال‌هایی استفاده کرده‌ایم. نمونه اولیه این توابع در فایل سرآیند <**cstring**> قرار دارند.

نمونه اولیه تابع	توضیح
`char *strcpy( char *S1, const char *S2 );`	رشته s2 را بدرون آرایه کاراکتری s1 کپی می‌کند. مقدار s1 برگشت داده می‌شود.
`char *strncpy( char *s1, const char *s2, size_t n);`	حداکثر n کاراکتر از رشته s2 را بدرون آرایه کارکتری s1 کپی می‌کند. مقدار s1 برگشت داده می‌شود.
`char *strcat ( char *s1, const char *s2 )`	رشته s2 را به s1 الصاق می‌کند. اولین کاراکتر s2 مبادرت به بازنویسی کاراکتر null از s1 می‌نماید. مقدار s1 برگشت داده می‌شود.
`char *strncat( char *s1, const char *s2, size_t n);`	حداکثر n کاراکتر از رشته s2 را به s1 الصاق می‌کند. اولین کاراکتر s2 مبادرت به بازنویسی کاراکتر null از s1 می‌نماید. مقدار s1 برگشت داده می‌شود.
`int strcmp( const char *s1, const char *s2 );`	رشته s1 را با رشته s2 مقایسه می‌کند. تابع مقدار صفر، کمتر از صفر (معمولاً 1-) یا بزرگتر از صفر (معمولاً 1) در صورتیکه به ترتیب s1 برابر، کوچکتر یا بزرگتر از s2 باشد، برگشت می‌دهد.
`int strncmp( const char *s1, const char *s2, size_t n );`	تا n کاراکتر مبادرت به مقایسه رشته s1 با رشته s2 می‌کند. تابع مقدار صفر، کمتر از صفر یا بزرگتر از صفر برگشت می‌دهد اگر بخش n کاراکتری از s1 برابر، کمتر یا بزرگتر از n کاراکتر متناظر با رشته s2 باشد (به‌ترتیب).
`char *strtok( char *s1, const char *s2 );`	با فراخوانی دنباله‌ای از strtok، رشته s1 به نشانه یا توکن‌های تقسیم می‌شود. تقسیم رشته برپایه کاراکترهای موجود در رشته s2 است. برای نمونه، اگر رشته "this: is: a: string" را برپایه کاراکتر ':' تقسیم کنیم، نتیجه نشانه‌گذاری "a" ، "is" ، "this" و "string" خواهد بود. تابع strtok در هر بار یک نشانه برگشت می‌دهد. اولین فراخوانی حاوی s1 بعنوان اولین آرگومان و فراخوانی‌ها متعاقب آن مبادرت به نشانه‌گذاری همان رشته حاوی NULL بعنوان اولین آرگومان ادامه خواهد داد. زمانیکه دیگر چیزی باقی نمانده باشد، تابع NULL برگشت خواهد داد.
`size_t strlen( const char *s );`	تعیین کننده طول رشته s است.

شکل ۳۰-۸ | توابع دستکاری کننده رشته از کتابخانه استاندارد رشته.

کپی رشته‌ها با strcpy و strncpy

www.yourshop.4kia.ir

تابع **strcpy** مبادرت به کپی دومین آرگومان خود (رشته) بدرون اولین آرگومان خود می‌کند که یک آرایه کاراکتری است که باید به میزان کافی برای ذخیره‌سازی رشته و کاراکتر null بزرگ باشد. تابع **strncpy** عملکردی همانند **strcpy** دارد، بجز اینکه این تابع به تعداد مشخص کاراکتر را از رشته به آرایه کپی می‌کند. توجه کنید که تابع **strncpy** ضرورتاً مبادرت به کپی کاراکتر خاتمه‌دهنده null در آرگومان دوم خود نمی‌کند، فقط در صورتی کاراکتر null کپی خواهد شد که تعداد کاراکترهای کپی شونده حداقل یک کاراکتر بیش از رشته طول داشته باشند. برای مثال، اگر "test" آرگومان دوم باشد، کاراکتر null فقط در صورتیکه آرگومان سوم در **strcnpy** حداقل 5 باشد کپی خواهد شد (چهار کاراکتر در "test" به اضافه یک کاراکتر null). اگر آرگومان سوم بزرگتر از 5 باشد، کاراکترهای null به آرایه الصاق می‌شوند تا اینکه مجموع تعداد کاراکترها مشخص شده توسط آرگومان سوم نوشته شود.

در برنامه شکل ۳۱-۸ از تابع **strcpy** در خط 17 برای کپی کل رشته موجود در آرایه **x** بدرون آرایه **y** و از تابع **strncpy** (خط 23) برای کپی 14 کاراکتر اول از آرایه **x** به آرایه **z** استفاده شده است. خط 24 مبادرت به الصاق کاراکتر null ('**\n**') به آرایه **z** می‌کند، چرا که فراخوانی **strncpy** در برنامه مبادرت به نوشتن کاراکتر null نمی‌کند (آرگومان سوم کمتر از طول آرگومان دوم به اضافه یک است).

```
1  // Fig. 8.31: fig08_31.cpp
2  // Using strcpy and strncpy.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <cstring> // prototypes for strcpy and strncpy
8  using std::strcpy;
9  using std::strncpy;
10
11 int main()
12 {
13    char x[] = "Happy Birthday to You"; // string length 21
14    char y[ 25 ];
15    char z[ 15 ];
16
17    strcpy( y, x ); // copy contents of x into y
18
19    cout << "The string in array x is: " << x
20       << "\nThe string in array y is: " << y << '\n';
21
22    // copy first 14 characters of x into z
23    strncpy( z, x, 14 ); // does not copy null character
24    z[ 14 ] = '\0'; // append '\0' to z's contents
25
26    cout << "The string in array z is: " << z << endl;
27    return 0; // indicates successful termination
28 } // end main
```

```
The string in array x is: Happy Birthday to You
The string in array y is: Happy Birthday to You
The string in array z is: Happy Birthday
```

شکل ۳۱-۸ | توابع strcpy و strncpy.

الصاق رشته‌ها با strcat و strncat

www.yourshop.4kia.ir

تابع **strcat** مبادرت به الصاق آرگومان دوم خود (یک رشته) به آرگومان اول خود (یک آرایه کاراکتری حاوی رشته) می‌کند. کاراکتر اول از آرگومان دوم جایگزین کاراکتر null می‌شود ('\n') که خاتمه‌دهنده رشته در آرگومان اول است. برنامه‌نویس باید مطمئن شود که آرایه بکار رفته برای ذخیره رشته اول به اندازه کافی برای ذخیره‌سازی ترکیب رشته اول، رشته دوم و کاراکتر null بزرگ است. تابع **strncat** مبادرت به الصاق تعداد مشخصی از کاراکترها از رشته دوم به رشته اول کرده و یک کاراکتر null به نتیجه اضافه می‌کند. برنامه شکل ۲۴-۸ به توصیف عملکرد توابع **strcat** (خطوط 19 و 29) و **strncat** (خط 24) می‌پردازد.

```
1  // Fig. 8.32: fig08_32.cpp
2  // Using strcat and strncat.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <cstring> // prototypes for strcat and strncat
8  using std::strcat;
9  using std::strncat;
10
11 int main()
12 {
13    char s1[ 20 ] = "Happy "; // length 6
14    char s2[] = "New Year "; // length 9
15    char s3[ 40 ] = "";
16
17    cout << "s1 = " << s1 << "\ns2 = " << s2;
18
19    strcat( s1, s2 ); // concatenate s2 to s1 (length 15)
20
21    cout << "\n\nAfter strcat(s1, s2):\ns1 = " << s1 << "\ns2 = " << s2;
22
23    // concatenate first 6 characters of s1 to s3
24    strncat( s3, s1, 6 ); // places '\0' after last character
25
26    cout << "\n\nAfter strncat(s3, s1, 6):\ns1 = " << s1
27       << "\ns3 = " << s3;
28
29    strcat( s3, s1 ); // concatenate s1 to s3
30    cout << "\n\nAfter strcat(s3, s1):\ns1 = " << s1
31       << "\ns3 = " << s3 << endl;
32    return 0; // indicates successful termination
33 } // end main
```

```
s1 = Happy
s2 = New Year

After strcat(s1, s2):
s1 = Happy New Year
s2 = New Year

After strncat(s3, s1, 6):
s1 = Happy New Year
s3 = Happy

After strcat(s3, s1):
s1 = Happy New Year
s3 = Happy Happy New Year
```

شکل ۳۲-۸ | توابع strcat و strncat.

www.yourshop.4kia.ir

***مقایسه رشته‌ها با* strcmp *و* strncmp**

در برنامه شکل ۳۳-۸ سه رشته با استفاده از توابع **strcmp** (در خطوط 21، 22 و 23) و **strncmp** (در خطوط 26، 27 و 28) مقایسه شده‌اند. تابع **strcmp** مبادرت به مقایسه کاراکتر به کاراکتر اولین آرگومان رشته‌ای با دومین آرگومان رشته‌ای خود می‌کند. در صورتیکه رشته‌ها باهم برابر باشند تابع مقدار صفر، و در صورتیکه رشته اول کوچکتر از رشته دوم باشد مقدار منفی و اگر رشته اول بزرگتر از رشته دوم باشد مقدار مثبت برگشت می‌دهد. عملکرد تابع **strncmp** همانند **strcmp** است بجز اینکه **strncmp** به تعداد مشخص از کاراکترها شروع به مقایسه می‌کند. تابع **strncmp** در صورتیکه به کاراکتر null در یکی از آرگومان‌های رشته‌ای خود برسد، عملیات مقایسه را متوقف خواهد کرد. برنامه در هر بار فراخوانی تابع یک مقدار صحیح برگشت می‌دهد.

```
1  // Fig. 8.33: fig08_33.cpp
2  // Using strcmp and strncmp.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <iomanip>
8  using std::setw;
9
10 #include <cstring> // prototypes for strcmp and strncmp
11 using std::strcmp;
12 using std::strncmp;
13
14 int main()
15 {
16    char *s1 = "Happy New Year";
17    char *s2 = "Happy New Year";
18    char *s3 = "Happy Holidays";
19
20    cout << "s1 = " << s1 << "\ns2 = " << s2 << "\ns3 = " << s3
21       << "\n\nstrcmp(s1, s2) = " << setw( 2 ) << strcmp( s1, s2 )
22       << "\nstrcmp(s1, s3) = " << setw( 2 ) << strcmp( s1, s3 )
23       << "\nstrcmp(s3, s1) = " << setw( 2 ) << strcmp( s3, s1 );
24
25    cout << "\n\nstrncmp(s1, s3, 6) = " << setw( 2 )
26       << strncmp( s1, s3, 6 ) << "\nstrncmp(s1, s3, 7) = " << setw( 2 )
27       << strncmp( s1, s3, 7 ) << "\nstrncmp(s3, s1, 7) = " << setw( 2 )
28       << strncmp( s3, s1, 7 ) << endl;
29    return 0; // indicates successful termination
30 } // end main
```

```
s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Holidays

strcmp(s1,s2) =  0
strcmp(s1,s3) =  1
strcmp(s3,s1) = -1

strcmd(s1, s3, 6) =  0
strcmd(s1, s3, 7) =  1
strcmd(s3, s1, 7) = -1
```

شکل ۳۳-۸ | توابع strcmp و strncmp.

www.yourshop.4kia.ir

برای اینکه بخوبی با مفهوم جمله یک رشته «بزرگتر» یا «کوچکتر» از رشته دیگر است آشنا شوید، به فرآیند به ترتیب الفباء نوشتن نام‌های خانوادگی دقت کنید. نبایستی در قرار دادن"Jones" قبل از "Smith" شک کنید چرا که حرف اول"Jones" قبل از حرف اول "Smith" در الفباء آمده است. اما الفباء چیزی بیش از لیست 26 حرفی است و یک لیست مرتب شده از کاراکترها می‌باشد. هر حرف در مکان خاصی از لیست جای دارد.

چگونه کامپیوتر از قرار گرفتن کاراکتری قبل از کاراکتر دیگری مطلع است؟ تمام کاراکترهای عرضه شده در کامپیوتر دارای کدهای عددی هستند و زمانیکه کامپیوتر مبادرت به مقایسه دو رشته می‌کند، در واقع به مقایسه کدهای عددی کاراکترهای موجود در رشته می‌پردازد.

برای استانداردسازی عرضه کاراکترها، اکثر سازندگان کامپیوتر، براساس یکی از دو طرح کدگذاری، *ASCII* یا *EBCODIC* کامپیوترهای خود را می‌سازند. بخاطر دارید که ASCII سرنام کلمات " *American Standard Code for Information Interchang*" و EBCDC سرنام کلمات " Extended *Binary Coded Decimal Interchange Code*" است. البته طرح‌های کدگذاری دیگری هم وجود دارند اما این دو طرح از محبوبیت بیشتری برخوردار هستند. به ASCII و EBCDIC *کدهای کاراکتر* یا *مجموعه کاراکتری* هم گفته می‌شود.

نشانه‌گذاری رشته با strtok

تابع **strtok** مبادرت به تقسیم یک رشته به دنباله‌ای از *نشانه‌ها* یا *توکن‌ها* می‌کند. یک توکن توالی از کاراکترهای مجزا شده توسط کاراکترهای حائل است (معمولاً فاصله یا نماد نقطه‌گذاری). برای مثال، در یک خط از متن هر کلمه می‌تواند بعنوان یک توکن در نظر گرفته شود و فاصله بین کلمات می‌توانند بعنوان حائل مورد توجه واقع شوند.

برای تقسیم یک رشته‌ها به توکن‌ها، نیاز به فراخوانی‌های مضاعف **strtok** است(با فرض اینکه رشته حاوی بیش از یک توکن است). اولین فراخوانی **strtok** حاوی دو آرگومان، یکی رشته برای توکن شدن و دیگری رشته‌ای حاوی حائل است. در خط 19 برنامه شکل ۳۴-۸ مبادرت به اشاره دادن **tokenPtr** به اولین توکن در **sentence** شده است. آرگومان دوم،" "، نشان می‌دهد که توکن‌ها در **sentence** توسط فاصله از هم متمایز خواهند شد. تابع **strtok** جستجویی برای یافتن اولین کاراکتر در **sentence** می‌کند که یک کاراکتر حائل نیست (فاصله). این شروع اولین توکن است. سپس تابع، کاراکتر حائل بعدی در رشته را پیدا کرده و آنرا با یک کاراکتر null جایگزین می‌سازد. ('**\n**'). در اینحالت توکن جاری خاتمه می‌پذیرد. تابع **strtok** اشاره‌گر را به کاراکتری بعدی پس از توکن در **sentence** انتقال می‌دهد و یک اشاره‌گر به توکن جاری برگشت داده می‌شود.

www.yourshop.4kia.ir

```
1  // Fig. 8.34: fig08_34.cpp
2  // Using strtok.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <cstring> // prototype for strtok
8  using std::strtok;
9
10 int main()
11 {
12    char sentence[] = "This is a sentence with 7 tokens";
13    char *tokenPtr;
14
15    cout << "The string to be tokenized is:\n" << sentence
16       << "\n\nThe tokens are:\n\n";
17
18    // begin tokenization of sentence
19    tokenPtr = strtok( sentence, " " );
20
21    // continue tokenizing sentence until tokenPtr becomes NULL
22    while ( tokenPtr != NULL )
23    {
24       cout << tokenPtr << '\n';
25       tokenPtr = strtok( NULL, " " ); // get next token
26    } // end while
27
28    cout << "\nAfter strtok, sentence = " << sentence << endl;
29    return 0; // indicates successful termination
30 } // end main
```

```
The string to be tokenized is :
This is a sentence with 7 tokens

The tokens are:

This
is
a
sentence
with
7
tokens

After strtok, sentence = This
```

شکل ۳٤-۸ | تابع strtok.

با فراخوانی‌های پشت سرهم **strtok**، رشته sentence توکن می‌شود که **NULL** بعنوان آرگومان اول ارسال می‌گردد. آرگومان NULL بر این نکته تاکید دارد که فراخوانی **strtok** بایستی عملیات توکن کردن را از مکانی در sentence انجام دهد که در آخرین فراخوانی **strtok** باقی مانده است. توجه کنید که **strtok** این اطلاعات را به روشی ذخیره می‌سازد که در دید برنامه‌نویس قرار ندارند. اگر هیچ توکنی به هنگام فراخوانی **strtok** باقی نمانده باشد، تابع مقدار **NULL** برگشت می‌دهد. برنامه شکل ۳۴-۸ با استفاده از تابع **strtok** مبادرت به توکن کردن رشته "This is a sentence with 7 tokens" کرده است. برنامه هر توکن را در یک خط متمایز چاپ می‌کند. خط 28 پس از توکن‌سازی جمله مبادرت به چاپ sentence می‌کند. توجه کنید که تابع **strtok** در رشته ورودی تغییر بوجود می‌آورد، از اینرو، یکی کپی از رشته در صورتیکه برنامه نیاز به رشته اصلی پس از فراخوانی **strtok** داشته باشد، تهیه کنید. مشاهده

www.yourshop.4kia.ir

می‌کنید که پس از توکن کردن رشته، زمانیکه sentence چاپ می‌شود فقط حاوی کلمه "This" است، چرا که **strtok** هر فاصله در sentence را با یک کاراکتر null در زمان توکن کردن جایگزین ساخته است.

تعیین طول رشته

تابع **strlen** یک رشته بعنوان یک آرگومان دریافت و تعداد کاراکترهای موجود در رشته را برگشت می‌دهد (کاراکتر null در طول رشته محاسبه نمی‌شود). برنامه شکل ۳۵-۸ به بررسی عملکرد تابع **strlen** پرداخته است.

```
1  // Fig. 8.35: fig08_35.cpp
2  // Using strlen.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <cstring> // prototype for strlen
8  using std::strlen;
9
10 int main()
11 {
12    char *string1 = "abcdefghijklmnopqrstuvwxyz";
13    char *string2 = "four";
14    char *string3 = "Boston";
15
16    cout << "The length of \"" << string1<< "\" is "<< strlen( string1 )
17       << "\nThe length of \"" << string2<< "\" is "<< strlen( string2 )
18       << "\nThe length of \"" << string3<< "\" is "<< strlen( string3 )
19       << endl;
20    return 0; // indicates successful termination
21 } // end main
```

```
The length of "abcdefghijklmnopqrstuvwxyz" is 26
The length of "four" is 4
The length of "Boston" is 6
```

شکل ۳۵-۸ | تابع strlen طول یک رشته *char را برگشت می‌دهد.

خودآزمایی

۱-۸ به موارد زیر پاسخ دهید:

a) اشاره‌گر متغیری است که مقدار آن ..... یک متغیر دیگر است.

b) سه مقداری که می‌توان در مقداردهی اولیه یک اشاره‌گر بکار گرفت عبارتند از .....، ..... و .....

c) تنها مقدار صحیحی که می‌توان مستقیماً به یک اشاره‌گر تخصیص داد، مقدار..... است.

۲-۸ کدامیک از موارد زیر صحیح و کدامیک اشتباه است.

a) عملگر آدرس & می‌تواند فقط بر روی ثابت‌ها و عبارات بکار گرفته شود.

b) اشاره‌گری که از نوع void * اعلان شده است می‌تواند ردگیری شود.

c) اشاره‌گرها از انواع مختلف هرگز نمی‌توانند بدون عملیات تبدیل (cast) به نوع دیگری تخصیص داده شوند.

www.yourshop.4kia.ir

۸-۳ برای هر یک از موارد زیر عبارتی در C++ بنویسید که وظیفه درخواستی را به انجام برساند. فرض کنید که اعداد با دقت مضاعف، اعشاری در هشت بایت ذخیره می‌شوند و آدرس شروع آرایه در مکان 1002500 قرار دارد. هر بخش از تمرین باید از نتایج بخش‌های قبلی استفاده کند.

a) آرایه‌ای از نوع double بنام numbers با 10 عنصر اعلان کرده و عناصر آرایه را با مقادیر 0.0,1.1,2.2,...,9.9 مقداردهی کنید. فرض کنید که ثابت نمادین SIZE با مقدار 10 تعریف شده باشد.

b) اشاره‌گر nPtr را اعلان کنید که به متغیری از نوع double اشاره کند.

c) از یک عبارت for برای چاپ عناصر آرایه numbers با استفاده از شاخص آرایه استفاده کنید. هر عدد را با دقت یک رقم در سمت راست نقطه دیسمال چاپ کنید.

d) دو عبارت مجزا بنویسید که به هر یک آدرس شروع آرایه numbers را به متغیر اشاره‌گر nPtr تخصیص دهد.

e) با استفاده از عبارت for عناصر آرایه numbers را با استفاده از نماد اشاره‌گر/افست با اشاره‌گر nPtr چاپ کنید.

f) با استفاده از عبارت for عناصر آرایه numbers را با استفاده از نماد اشاره‌گر/افست و توسط نام آرایه بعنوان اشاره‌گر چاپ کنید.

g) با استفاده از عبارت for عناصر آرایه numbers را با استفاده از نماد اشاره‌گر/افست با اشاره‌گر nPtr چاپ کنید.

h) به پنجمین عنصر آرایه numbers با استفاده از شاخص اشاره‌گر/افست با نام آرایه بعنوان اشاره‌گر، نماد شاخص اشاره‌گر با nPtr و نماد اشاره‌گر/افست با nPtr مراجعه کنید.

i) فرض کنید که nPtr به ابتدای آرایه numbers اشاره می‌کند، آدرس مورد مراجعه، کدام آدرس توسط nPrt + 8 مورد مراجعه قرار می‌گیرد؟ چه مقداری در آن مکان ذخیره شده است؟

j) فرض کنید که nPtr به numbers[5] اشاره دارد، کدام آدرس توسط nPtr پس از nPtr -= 4 بکار گرفته خواهد شد؟ چه مقداری در آن مکان ذخیره شده است؟

۸-۴ برای هر یک از موارد زیر، یک عبارت بنویسید که کار خواسته شده را انجام دهد. فرض کنید که متغیرهای اعشاری number1 و number2 اعلان شده‌اند و number1 با 7.3 مقداردهی اولیه شده است. فرض کنید که متغیر ptr از نوع char * است. همچنین فرض نمائید که آرایه‌های s1 , s2 هر یک آرایه‌های 100 عنصری از نوع char هستند که با رشته لیترال مقداردهی اولیه شده‌اند.

a) اعلان متغیر fPtr برای اشاره به یک شی از نوع double.

b) تخصیص آدرس متغیر number1 برای اشاره به متغیر fPtr.

c) چاپ مقدار شی مورد اشاره توسط fPtr.

d) تخصیص مقدار شی مورد اشاره توسط fPtr به متغیر number2.

e) چاپ مقدار number2.

f) چاپ آدرس number1.

g) چاپ آدرس ذخیره شده در fPtr. آیا مقدار چاپ شده همان آدرس number1 است؟

www.yourshop.4kia.ir

h) کپی رشته ذخیره شده در آرایه s2 به آرایه s1.

i) مقایسه رشته موجود در s1 با رشته s2 و چاپ نتیجه.

j) اضافه کردن 10 کاراکتر اول از رشته s2 به رشته s1.

k) تعیین طول رشته موجود در s1 و چاپ نتیجه.

L) تخصیص مکان اولین توکن در s2 به ptr.توکن‌ها توسط کاما (،) از هم مجزا می‌شوند.

۸–۵ برای برآورده کردن هر یک از موارد زیر، عبارتی بنویسید:

a) سرآیندی برای تابعی بنام exchange بنویسید که دو اشاره‌گر به مقدار اعشاری با دقت مضاعف x , y دریافت کرده و مقداری برگشت نمی‌دهد.

b) نمونه اولیه تابع برای تابع معرفی شده در بخش (a) بنویسید.

c) سرآیندی برای تابع evaluate بنویسید که یک مقدار صحیح برگشت داده و پارامتری x از نوع صحیح و اشاره‌گری به تابع poly را دریافت نماید. تابع poly یک پارامتر صحیح دریافت کرده و مقدار صحیحی را برگشت می‌دهد.

d) نمونه اولیه تابع برای تابع معرفی شده در بخش (c) بنویسید.

e) دو عبارت بنویسید که هر یک آرایه کاراکتری vowel را با رشته "AEIOU" مقداردهی اولیه کنند.

۸–۶ در هر یک از بخش‌های زیر خطای رخ داده را پیدا کنید. فرض کنید که اعلان‌ها و عبارات زیر صورت گرفته‌اند:

```
int *zPtr ; //zPtr will refernce array z
int *aPtr = 0;
voild *sPtr = 0;
int number;
int z[5] = {1,2,3,4,5};

b)
//use pointer to get first value of array
Number=zPtr;
c)
//assign array element 2 (the value 3) to number
number=*zPtr[2];
d)
//print entire array z
for(int i=0; i<=5; i++)
 cout<<zPtr[i]<<endl;
e)
//assign the value pointed to by sPtr to number
number=*sPtr;
f)
++z;
g)
```

www.yourshop.4kia.ir

```
char s[10];
cout<<strncpy(s,"hello",5)<<endl;
h)
char s[12];
strcpy(s, "Welcome home");
i)
if(strcmp(string1,string2))
cout<<"The strings are equal"<<endl;
```

۸–۷ به هنگام اجرای هر یک از عبارات زیر چه جمله‌ای چاپ خواهد شد؟ اگر عبارتی حاوی خطا باشد، خطا را توضیح داده و نحوه اصلاح آنرا بیان کنید. فرض کنید متغیرهای زیر اعلان شده‌اند:

```
Char s1[50]= "Jack";
Char s2[60]="jill";
Char s3[50];

a) cout<<strcpy(s3,s2)<<endl;
b) cout<<strcat(strcat(strcpy(s3,s1), "and"),s2)<<endl;
c) cout<<strlen(s1) + strlen(s2)<<endl;
d) cout<<strlen(s3)<<endl;
```

پاسخ خودآزمایی

۸–۱

a) آدرس. b) صفر، Null، آدرس. c) صفر.

۸–۲

a) اشتباه. عملوند، عملگر آدرس بایستی یک lvalue باشد. عملگر آدرس نمی‌تواند با ثابت‌ها یا عباراتی بکار رود که نتیجه‌ای از مراجعه‌ها ندارند.

b) اشتباه. اشاره‌گر به void نمی‌تواند ردگیری شود. چنین اشاره‌گری دارای نوع نیست که کامپایلر بتواند تعداد بایت‌های حافظه را برای ردگیری و نوع داده که اشاره‌گر به آن اشاره می‌کند را تعیین کند.

c) اشتباه. اشاره‌گرها از هر نوع می‌توانند به اشاره‌گرهای void تخصیص یابند. اشاره‌گرها از نوع void می‌توانند فقط با تبدیل صریح نوع به اشاره‌گرهایی از نوع دیگر تخصیص داده شوند.

۸–۳

```
a)
double numbers[SIZE]={0.0,1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8,9.9};
b)
double *nPtr;
c)
cout<<fixed<<showpoint<<setprecision(1);
 for(int i=0; i<SIZE;i++)
   cout<<numbers[i]<<' ';
d)
nPtr=numbers;
nPtr=&numbers[0];
```

www.yourshop.4kia.ir

```
e)
cout<<fixed<<showpoint<<setprecision(1);
 for(int j=0; j<SIZE;j++)
  cout<<*(nPtr + j)<<' ';
f)
cout<<fixed<<showpoint<<setprecision(1);
 for(int k=0; k<SIZE;K++)
  cout<< *(numbers +k) <<' ';
g)
cout<<fixed<<showpoint<<setprecision(1);
 for(int m=0; m<SIZE; m++)
  cout<<nPtr[m]<<' ';
h)
numbers[3]
*(numbers+3)
nPtr[3]
*(nPtr + 3)
```

i) آدرس برابر است با 1002500+8*8=1002564، مقدار 8.8 است.

j) آدرس numbers[5] برابر است با 1002500+5*8=1002540

آدرس nPtr -=4 برابر است با 1002540-4*8=1002508

مقدار موجود در آن مکان برابر 1.1 است.

۸-۴

```
a) double *fPtr;
b) fPtr=&number1;
c) cout<<"The value of *fPtr is"<<*fPtr<<endl;
d) number2=*fPtr;
e) cout<<"The value of number2 is"<<number2<<endl;
f) cout<<"The address of number1 is"<<&number1<<endl;
g) cout<<"The address stored in fPtr is"<<fPtr is"<<fPtr<<endl;
h) strcpy(s1,s2);
i) cout<<"strcmp(s1,s2)="<<strcmp(s1,s2)<<endl;
j) strncat(s1,s2,10);
k) cout<<"strlen(s1)="<<strlen(s1)<<endl;
l) ptr= strtok(s2,",");
```

۸-۵

```
a) void exchange (double **,double *y)
b) void exchange(double*,double*);
c) int evaluate (int x,int(*poly)(int))
d) int evaluate(int,int(*)(int));
e) char vawe[] = "AEIOU";
   char vawe[] = {'A','E','I','O','U','10'};
```

۸-۶

a) خطا: ptr= مقدار دهی نشده است.

www.yourshop.4kia.ir

اصلاح: مقداردهی zftr با zptr=z;

b) خطا: اشاره‌گر ردگیری نشده است.

اصلاح: تغییر عبارت به number= *zPtr;

c) خطا: zPtr[2] اشاره‌گر نبوده و نمی‌توان ردگیری شود.

اصلاح: تغییر *zPtr[2] به zPtr[2]

e) خطا: مراجعه به عنصر آرایه خارج از مرزهای آرایه با شاخص اشاره‌گر.

اصلاح: برای اجتناب از اینکار, عملکرد رابطه در عبارت for را به > تغییر دهید.

۸-۷

a) jill
b) jack and jill
c) 8
d) 13

۸-۸ مشخص کنید که آیا عبارات زیر صحیح هستند یا اشتباه. اگر اشتباه هستند، علت را توضیح دهید.

a) دو اشاره‌گر که به آرایه‌های متفاوت اشاره می‌کنند، نمی‌توانند بطور موثر با هم مقایسه شوند.

b) چون نام آرایه یک اشاره‌گر به اولین عنصر آرایه است، اسامی آرایه‌ها می‌توانند به همان روش اشاره‌گرها، دچار تغییر شوند.

۸-۹ برای هر یک از موارد زیر، عباراتی به C++ بنویسید که کار درخواستی را انجام دهند. فرض کنید که مقادیر صحیح بدون علامت در دو بایت ذخیره شده و آدرس شروع آرایه در مکان 1002500 قرار دارد.

a) آرایه از نوع unsigned int بنام values با پنج عنصر اعلان کنید. عناصر آرایه را با مقادیر زوج 2 تا 10 مقداردهی اولیه نمائید. فرض کنید نماد ثابت SIZE با 5 تعریف شده است.

b) اعلان اشاره‌گر vPtr که به شی از نوع unsigned int اشاره می‌کند.

c) استفاده از عبارت for برای چاپ عناصر آرایه values با استفاده از شاخص.

d) دو عبارت مجزا بنویسید که آدرس شروع آرایه values را به متغیر اشاره‌گر vPtr تخصیص دهد.

e) استفاده از عبارت for برای چاپ عناصر آرایه values با استفاده از نماد اشاره‌گر/افست.

f) استفاده از عبارت for برای چاپ عناصر آرایه values با استفاده از نماد اشاره‌گر/افست به همراه نام آرایه بعنوان اشاره‌گر.

g) استفاده از عبارت for برای چاپ عناصر آرایه values توسط شاخص.

h) مراجعه به عنصر پنجم values با استفاده از شاخص آرایه، اشاره‌گر/افست توسط نام آرایه بعنوان اشاره‌گر، شاخص اشاره‌گر آرایه و نماد اشاره‌گر/افست.

i) آدرس مورد مراجعه vPtr + 3 کجاست؟ مقدار ذخیره شده در آن مکان؟

j) فرض کنید که vPtr به values[4] اشاره دارد، آدرس مورد مراجعه توسط vPtr -=4 کجاست؟ مقدار ذخیره شده در آن مکان؟

www.yourshop.4kia.ir

۸–۱۰ برای هر یک از موارد زیر یک عبارت C++ بنویسید. فرض کنید که متغیرهای صحیح long بنام‌های value1 و value2 قبلا اعلان شده‌اند و value1 با 200000 مقداردهی اولیه شده است.

a) اعلان متغیر longPtr برای اشاره به یک شی از نوع long.

b) تخصیص آدرس متغیر value1 به متغیر اشاره‌گر longPtr.

c) چاپ مقدار شی اشاره شده توسط longPtr.

d) تخصیص مقدار شی مورد اشاره توسط longPtr به متغیر value2.

e) چاپ مقدار value2.

f) چاپ آدرس value1.

g) چاپ آدرس ذخیره شده در longPtr. آیا مقدار چاپ شده همان آدرس value1 است؟

۸–۱۱ برای انجام موارد زیر عباراتی در C++ بنویسید:

a) سرآیند تابع zero را بنویسید که یک مقدار صحیح long از پارامتر آرایه bigIntegers دریافت کرده و مقداری برگشت نمی‌دهد.

b) نمونه اولیه تابع را برای تابع معرفی شده در بخش (a) بنویسید.

c) سرآیند تابع add1AndSum را بنویسید که پارامتر آرایه‌ای صحیح oneTooSmall را دریافت و مقدار صحیح برگشت دهد.

d) نمونه اولیه تابع را برای تابع معرفی شده در بخش (c) بنویسید.

www.yourshop.4kia.ir