فصل هیجدهم

کلاس string و پردازش رشته

اهداف

- استفاده از کلاس string موجود در کتابخانه استاندارد C++.
- تخصیص دادن، به هم پیوستن، مقایسه، جستجو و عوض کردن رشته‌ها.
- تعیین خصوصیات string.
- یافتن، جایگزین کردن و وارد ساختن کاراکترها در یک رشته.
- تبدیل رشته‌ها به رشته‌های سبک C و برعکس.
- استفاده از تکرار شونده‌های string.
- انجام عملیات ورودی و خروجی رشته‌ها در حافظه.

www.yourshop.4kia.ir

رئوس مطالب

۱۸-۱ مقدمه
۱۸-۲ تخصیص و به هم پیوستن رشته‌ها
۱۸-۳ مقایسه رشته‌ها
۱۸-۴ زیر رشته‌ها
۱۸-۵ عوض کردن رشته‌ها
۱۸-۶ خصوصیات string
۱۸-۷ یافتن رشته‌ها و کاراکترها در یک رشته
۱۸-۸ جایگزین ساختن کاراکترها در یک رشته
۱۸-۹ درج کاراکترها در یک رشته
۱۸-۱۰ تبدیل به سبک رشته‌های C
۱۸-۱۱ تکرار شونده‌ها
۱۸-۱۲ پردازش استریم رشته

۱۸-۱ مقدمه

الگوی کلاس **basic_string** در C++ سرمشق انواع عملیات بکار رفته بر روی رشته‌ها همانند کپی، جستجو و موارد دیگر است. تعریف الگو و تمام امکانات پشتیبانی شده در فضای‌نامی **std** تعریف شده‌اند، که با عبارت **typedef** به حساب می‌آیند

```
typedef basic_string< char > string;
```

که نام جانشین **string** را برای **basic_string<char>** ایجاد می‌کند. همچنین یک **typedef** برای نوع **wchar_t** وجود دارد. نوع **wchar_t** اقدام به ذخیره کاراکترها می‌کند (مثلاً کاراکترهای دو بایتی، چهار بایتی و غیره) تا از دیگر مجموعه‌های کاراکتری پشتیبانی شود. در این فصل انحصاراً از **string** استفاده کرده‌ایم. برای استفاده از رشته‌ها، فایل سرآیند **<string>** بکار گرفته می‌شود.

یک شی **string** می‌تواند توسط آرگومان یک سازنده همانند،

```
string text( "Hello" ); //creates string form const char *
```

مقداردهی اولیه شود، که یک رشته حاوی کاراکترهای موجود در "Hello" ایجاد می‌کند، یا با دو آرگومان سازنده همانند

```
string name( 8,'x' ); //string of 8 'x' charcters
```

که رشته‌ای حاوی هشت کاراکتر 'x' تولید می‌کند. همچنین کلاس **string** دارای سازنده پیش‌فرض (که یک رشته تهی ایجاد می‌کند) و سازنده کپی کننده است. یک *رشته تهی* (*empty string*) رشته‌ای است که حاوی هیچ کاراکتری نمی‌باشد.

www.yourshop.4kia.ir

همچنین می‌توان رشته‌ها را از طریق گرامر ساخت متناوب یا جایگزین در تعریف یک رشته مقداردهی اولیه کرد همانند

```
string month = "March"; //same as: string month( "March" );
```

بخاطر داشته باشید که عملگر = در عبارت قبلی یک عملگر تخصیص نمی‌باشد، بلکه فراخوانی ضمنی سازنده کلاس **string** است که عمل تبدیل را انجام می‌دهد. دقت نماید که کلاس **string** تبدیلی از **int** یا **char** به رشته در تعریف یک رشته انجام نمی‌دهد. برای مثال نتیجه تعاریف زیر

```
sring error1 = 'c';
string error2 = ( 'u' );
string error3 = 22;
string error4( 8 );
```

خطاهای نحوی است. توجه کنید که تخصیص یک کاراکتر به یک شی **string** در یک عبارت تخصیص همانند زیر مجاز است

```
string1 = 'n';
```

برخلاف رشته‌های * **char** در سبک C، رشته‌ها ضرورتاً با **null** خاتمه پیدا نمی‌کنند. طول یک رشته را می‌توان با توابع عضو **length** و **size** بازیابی کرد. عملگر شاخص، **[]**، می‌تواند به همراه رشته‌ها به منظور دسترسی و اصلاح کاراکترهای مجزا بکار گرفته شود. همانند رشته‌های سبک C، رشته‌ها در اولین شاخص مقدار صفر و آخرین شاخص مقدار **length () - 1** دارند.

اکثر توابع عضو **string** آرگومان‌های بعنوان شاخص موقعیت شروع و تعداد کاراکترهای که بر روی آنها کار خواهد شد دریافت می‌کنند.

عملگر << برای پشتیبانی از رشته‌ها سرسربارگذاری شده است. عبارت

```
string stringObject;
cin >> stringObject;
```

رشته‌ای از یک دستگاه ورودی می‌خواند. ورودی توسط کاراکترهای *white-space* تعیین حدود می‌شود. زمانیکه با چنین کاراکتری مواجه شود، عملیات ورودی خاتمه می‌یابد. همچنین تابع **getline** برای رشته سرسربارگذاری شده است. عبارت

```
string string1;
getline( cin, string1);
```

رشته را از صفحه‌کلید بدرون **string1** می‌خواند. حدود ورودی توسط خط جدید ('**\n**') مشخص می‌شود، از اینرو **getLine** می‌تواند یک خط از متن را بدرون یک شی **string** بخواند.

۱۸-۲ تخصیص و به هم پیوستن رشته‌ها

برنامه شکل ۱-۱۸ به بررسی تخصیص و به هم پیوستن رشته‌ها پرداخته است. خط 7 سرآیند <**string**> را برای کلاس رشته بکار برده است. رشته‌های **string1**، **string2** و **string3** در خطوط 14-12 ایجاد می‌شوند. خط 16 مقدار **string1** را به **string2** تخصیص می‌دهد. پس از انجام تخصیص **string2** کپی از

www.yourshop.4kia.ir

string1 خواهد بود. خط 17 از تابع عضو **assign** برای کپی کردن **string1** به **string3** استفاده کرده است. یکی کپی مجزا ایجاد می‌گردد (یعنی **string1** و **string3** شی‌های مستقلی هستند). همچنین کلاس رشته دارای نسخه سرسربارگذاری شده‌ای از تابع عضو **assign** است که به تعداد مشخص شده از کاراکترها را کپی می‌کند، بصورت زیر

```
targetString.assign( sourceString, start, numberOfCharacters );
```

در عبارت فوق، **sourceString** رشته‌ای است که کپی خواهد شد، **start** شاخص شروع بوده و **numberOfCharacters** تعداد کاراکترهای است که کپی می‌شود.

خط **22** از عملگر شاخص برای تخصیص **'r'** به **string3[2]** و تخصیص **'r'** به **string2[0]** استفاده کرده است. سپس رشته‌ها چاپ شده‌اند.

```
1  // Fig. 18.1: Fig18_01.cpp
2  // Demonstrating string assignment and concatenation.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <string>
8  using std::string;
9
10 int main()
11 {
12    string string1( "cat" );
13    string string2;
14    string string3;
15
16    string2 = string1; // assign string1 to string2
17    string3.assign( string1 ); // assign string1 to string3
18    cout << "string1: " << string1 << "\nstring2: " << string2
19       << "\nstring3: " << string3 << "\n\n";
20
21    // modify string2 and string3
22    string2[ 0 ] = string3[ 2 ] = 'r';
23
24    cout << "After modification of string2 and string3:\n"<< "string1:"
25       << string1 << "\nstring2: " << string2 << "\nstring3: ";
26
27    // demonstrating member function at
28    for ( int i = 0; i < string3.length(); i++ )
29       cout << string3.at( i );
30
31    // declare string4 and string5
32    string string4( string1 + "apult" ); // concatenation
33    string string5;
34
35    // overloaded +=
36    string3 += "pet"; // create "carpet"
37    string1.append( "acomb" ); // create "catacomb"
38
39    // append subscript locations 4 through end of string1 to
40    // create string "comb" (string5 was initially empty)
41    string5.append( string1, 4, string1.length() - 4 );
42
43    cout << "\n\nAfter concatenation:\nstring1: " << string1
44       << "\nstring2: " << string2 << "\nstring3: " << string3
45       << "\nstring4: " << string4 << "\nstring5: " << string5 << endl;
46    return 0;
47 }  // end main
```

```
string1: cat
string2: cat
string3: cat

After modification of string2 and string3:
```

www.yourshop.4kia.ir

```
string1: cat
string2: rat
string3: car

After concatenation:
string1: catacomb
string2: rat
string3: carpet
string4: catapult
string5: comb
```

شکل ۱-۱۸ | تخصیص و هم پیوستن رشته‌ها.

خطوط 29-28 محتویات **string3** را یک به یک توسط تابع عضو **at** چاپ می‌کنند. تابع عضو **at** دارای قابلیت دسترسی بررسی شده است (یا محدودهْ بررسی شده است). به این معنی که اگر از انتهای رشته عبور شود، استثنا **out_of_range** به راه می‌افتد. توجه کنید که عملگر شاخص **[]** دسترسی بررسی شده را انجام نمی‌دهد. اینکار بر روی آرایه‌ها انجام می‌شود.

رشته **string4** در خط 32 اعلان شده و با نتیجه به هم پیوستن **string1** و "apult" با استفاده از عملگر جمع سرسربارگذاری شده، +، که بر روی کلاس رشته نقش پیوند زننده را بازی می‌کند، مقداردهی اولیه شده است. در خط 36 از عملگر += برای پیوند زدن **string3** و "pet" استفاده شده است. خط 37 از تابع عضو **append** برای پیوند زدن **string1** و "acomb" استفاده کرده است.

خط 41 رشته "comb" را به رشته تهی **string5** متصل کرده است. این تابع عضو مبادرت به ارسال رشته **string1** به منظور بازیابی کاراکترها از، شاخص شروع از **(4)string** و تعداد کاراکترها برای متصل شدن کرده است (مقدار برگشتی توسط **(string1.Length() - 4**)

۳-۱۸ مقایسه رشته‌ها

کلاس رشته دارای چندین تابع عضو برای مقایسه کردن رشته‌ها است. برنامه شکل ۲-۱۸ به بررسی قابلیت‌های مقایسه‌ای در کلاس رشته پرداخته است. برنامه در خطوط 15-12 مبادرت به اعلان چهار رشته کرده است و هر رشته را چاپ می‌کند (خطوط 18-17). شرط موجود در خط 21 به مقایسه رشته **string1** با **string4** به لحاظ تساوی می‌پردازد و اینکار را توسط عملگر سرسربارگذاری شده تساوی یا برابری انجام می‌دهد. اگر شرط برقرار باشد، "string1== string4" چاپ می‌شود. اگر شرط برقرار نباشد، شرط موجود در خط 25 تست می‌شود. تمام عملگرهای سرسربارگذاری شده در کلاس **string** در این برنامه بکار گرفته شده‌اند (**!=**, >، =< و =>) اما به توضیح خط به خط آنها نپرداخته‌ایم.

```
1  // Fig. 18.2: Fig18_02.cpp
2  // Demonstrating string comparison capabilities.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <string>
8  using std::string;
9
10 int main()
11 {
```

www.yourshop.4kia.ir

```
   string string1( "Testing the comparison functions." );
   string string2( "Hello" );
   string string3( "stinger" );
   string string4( string2 );

   cout << "string1: " << string1 << "\nstring2: " << string2
      << "\nstring3: " << string3 << "\nstring4:" << string4 << "\n\n";

   // comparing string1 and string4
   if ( string1 == string4 )
      cout << "string1 == string4\n";
   else // string1 != string4
   {
      if ( string1 > string4 )
         cout << "string1 > string4\n";
      else // string1 < string4
         cout << "string1 < string4\n";
   } // end else

   // comparing string1 and string2
   int result = string1.compare( string2 );

   if ( result == 0 )
      cout << "string1.compare( string2 ) == 0\n";
   else // result != 0
   {
      if ( result > 0 )
         cout << "string1.compare( string2 ) > 0\n";
      else // result < 0
         cout << "string1.compare( string2 ) < 0\n";
   } // end else

   // comparing string1 (elements 2-5) and string3 (elements 0-5)
   result = string1.compare( 2, 5, string3, 0, 5 );

   if ( result == 0 )
      cout << "string1.compare( 2, 5, string3, 0, 5 ) == 0\n";
   else // result != 0
   {
      if ( result > 0 )
         cout << "string1.compare( 2, 5, string3, 0, 5 ) > 0\n";
      else // result < 0
         cout << "string1.compare( 2, 5, string3, 0, 5 ) < 0\n";
   } // end else

   // comparing string2 and string4
   result = string4.compare( 0, string2.length(), string2 );

   if ( result == 0 )
      cout << "string4.compare( 0, string2.length(), "
         << "string2 ) == 0" << endl;
   else // result != 0
   {
      if ( result > 0 )
         cout << "string4.compare( 0, string2.length(), "
            << "string2 ) > 0" << endl;
      else // result < 0
         cout << "string4.compare( 0, string2.length(), "
            << "string2 ) < 0" << endl;
   } // end else

   // comparing string2 and string4
   result = string2.compare( 0, 3, string4 );

   if ( result == 0 )
      cout << "string2.compare( 0, 3, string4 ) == 0" << endl;
   else // result != 0
   {
      if ( result > 0 )
         cout << "string2.compare( 0, 3, string4 ) > 0" << endl;
```

www.yourshop.4kia.ir

```
82        else // result < 0
83           cout << "string2.compare( 0, 3, string4 ) < 0" << endl;
84     } // end else
85
86     return 0;
87 } // end main
```

```
string1: Testing the comparison functions.
string2: Hello
string3: stinger
string4: Hello

string1> string4
string1.compare( string2 ) > 0
string1.compare( 2, 5, string3, 0, 5 ) == 0
string4.compare( 0, string2.length(), string2 ) == 0
string2.compare( 0, 3, string4 ) < 0
string1.compare( string2 ) >0
```

شکل ۲-۱۸ | مقایسه رشته‌ها.

خط 32 از تابع عضو **compare** برای مقایسه **string1** با **string2** استفاده کرده است. اگر رشته‌ها با هم برابر باشند، متغیر **result** با صفر مقداردهی می‌شود. اگر **string1** بزرگتر از **string2** باشد با یک عدد مثبت، یا اگر **string1** کوچکتر از **string2** باشد با یک عدد منفی مقداردهی می‌شود. بدلیل اینکه رشته‌ای که با حرف **'T'** شروع می‌شود به لحاظ لغت‌نویسی بزرگتر از رشته‌ای است که با **'H'** شروع می‌شود، پس مقداری بزرگتر از صفر به **result** تخصیص می‌یابد (خروجی هم بر این نکته تاکید دارد).

خط 45 از نسخه سربارگذاری شده تابع عضو **compare** برای مقایسه بخش‌های از **string1** و **string3** استفاده کرده است. دو آرگومان اول (2 **و** 5) مشخص کننده شاخص شروع و طول بخشی از **string1** یعنی "string" برای مقایسه با **string3** هستند. آرگومان سوم، رشته مقایسه شونده است. دو آرگومان آخر (0 و 5) شاخص شروع و طول بخشی از رشته هستند که مورد مقایسه قرار خواهند گرفت.

در صورت برابری، مقدار صفر به **result**، در صورت بزرگتر بودن **string1** از **string3** مقدار مثبت و در صورت کوچکتر بودن **string1** از **string3** مقدار منفی به **result** تخصیص می‌یابد. بدلیل اینکه در این برنامه دو رشته مقایسه شده با هم یکسان هستند، **result** با صفر مقداردهی شده است.

خط 58 از نسخه سربارگذاری شده دیگری از تابع **compare** برای مقایسه **string4** و **string2** استفاده کرده است. دو آرگومان اول همان هستند، شاخص شروع و طول. آرگومان آخر، رشته مقایسه شونده است. مقدار برگشتی هم همان است، صفر برای تساوی، مقدار مثبت اگر **string4** بزرگتر از **string2** باشد یا مقدار منفی اگر **string4** کوچکتر از **string2** باشد. چون دو بخش مقایسه شده در این برنامه با هم یکسان هستند، **result** با صفر مقداردهی شده است.

خط 74 با فراخوانی تابع عضو **compare** مبادرت به مقایسه سه کاراکتر اول در رشته **string2** با **string4** کرده است. چون "Hel" کوچکتر از "Hello" است، مقداری کوچکتر از صفر برگشت داده شده است.

www.yourshop.4kia.ir

۱۸-۴ زیر رشته‌ها

کلاس رشته دارای تابع عضو **substr** برای بازیابی یک زیررشته از یک رشته است. نتیجه یک رشته جدید است که از رشته منبع کپی شده است. برنامه شکل ۳-۱۸ به بررسی **substr** پرداخته است.

```
1  // Fig. 18.3: Fig18_03.cpp
2  // Demonstrating string member function substr.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <string>
8  using std::string;
9
10 int main()
11 {
12    string string1( "The airplane landed on time." );
13
14    // retrieve substring "plane" which
15    // begins at subscript 7 and consists of 5 elements
16    cout << string1.substr( 7, 5 ) << endl;
17    return 0;
18 } // end main
plane
```

شکل ۳-۱۸ | بررسی تابع عضو substr.

برنامه در خط 12 یک رشته اعلان و مقداردهی اولیه کرده است. در خط 16 از تابع عضو **substr** برای بازیابی یک زیر رشته از **string1** استفاده شده است. آرگومان اول مشخص کننده شاخص آغاز از زیررشته مورد نظر، آرگومان دوم مشخص کننده طول زیررشته است.

۱۸-۵ عوض کردن رشته‌ها

کلاس رشته دارای تابع عضو **swap** برای عوض کردن رشته‌ها است. در برنامه شکل ۴-۱۸ دو رشته عوض شده‌اند. خطوط 13-12 مبادرت به اعلان و مقداردهی رشته‌های **first** و **second** کرده‌اند. سپس هر رشته چاپ شده است. خط 18 از تابع عضو **swap** برای عوض کردن مقادیر **first** و **second** استفاده کرده است. مجدداً دو رشته چاپ شده‌اند تا تاییدی برای عوض شدن جای رشته‌ها باشد. این تابع مناسب برنامه‌های است که مرتب‌سازی رشته‌ها را انجام می‌دهند.

```
1  // Fig. 18.4: Fig18_04.cpp
2  // Using the swap function to swap two strings.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <string>
8  using std::string;
9
10 int main()
11 {
12    string first( "one" );
13    string second( "two" );
14
15    // output strings
16    cout << "Before swap:\n first: " << first << "\nsecond: " << second;
17
18    first.swap( second ); // swap strings
```

www.yourshop.4kia.ir

```
19
20    cout << "\n\nAfter swap:\n first: " << first
21       << "\nsecond: " << second << endl;
22    return 0;
23 } // end main
```

```
Before swap:
 first: one
second: two

After swap:
 first: two
second: one
```

شکل ٤-١٨ | استفاده از تابع swap برای عوض کردن رشته‌ها.

٦-١٨ خصوصیات string

کلاس **string** دارای توابع عضوی برای جمع‌آوری اطلاعاتی در ارتباط با سایز، طول، ظرفیت، حداکثر طول و خصوصیات دیگر رشته است. سایز یا طول یک رشته تعداد کاراکترهای جاری ذخیره شده در رشته است. ظرفیت یا گنجایش یک رشته تعداد کاراکترهای است که می‌تواند در یک رشته بدون اخذ حافظه بیشتر ذخیره گردد. ظرفیت یک رشته بایستی حداقل برابر سایز جاری رشته باشد، اگرچه می‌تواند بزرگتر هم باشد.

ظرفیت دقیق یک رشته وابسته به پیاده‌سازی آن است. بزرگترین سایز، سایز ممکنه‌ای است که یک رشته می‌تواند داشته باشد. اگر از این مقدار تجاوز شود، استثنا **length_error** رخ می‌دهد. برنامه شکل ۵-۱۸ به بررسی توابع عضو کلاس رشته پرداخته و از این خصوصیات در آن استفاده شده است.

```
1  // Fig. 18.5: Fig18_05.cpp
2  // Demonstrating member functions related to size and capacity.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::cin;
7  using std::boolalpha;
8
9  #include <string>
10 using std::string;
11
12 void printStatistics( const string & );
13
14 int main()
15 {
16    string string1;
17
18    cout << "Statistics before input:\n" << boolalpha;
19    printStatistics( string1 );
20
21    // read in only "tomato" from "tomato soup"
22    cout << "\n\nEnter a string: ";
23    cin >> string1; // delimited by whitespace
24    cout << "The string entered was: " << string1;
25
26    cout << "\nStatistics after input:\n";
27    printStatistics( string1 );
28
29    // read in "soup"
30    cin >> string1; // delimited by whitespace
31    cout << "\n\nThe remaining string is: " << string1 << endl;
32    printStatistics( string1 );
```

www.yourshop.4kia.ir

```
33
34    // append 46 characters to string1
35    string1 += "1234567890abcdefghijklmnopqrstuvwxyz1234567890";
36    cout << "\n\nstring1 is now: " << string1 << endl;
37    printStatistics( string1 );
38
39    // add 10 elements to string1
40    string1.resize( string1.length() + 10 );
41    cout << "\n\nStats after resizing by (length + 10):\n";
42    printStatistics( string1 );
43
44    cout << endl;
45    return 0;
46 } // end main
47
48 // display string statistics
49 void printStatistics( const string &stringRef )
50 {
51    cout << "capacity: " << stringRef.capacity() << "\nmax size: "
52       << stringRef.max_size() << "\nsize: " << stringRef.size()
53       << "\nlength: " << stringRef.length()
54       << "\nempty: " << stringRef.empty();
55 } // end printStatistics
```

```
Statistics befor input:
capacity: 0
max size: 4294967293
size: 0
length: 0
empty: true

Enter a string: tomato soup
The string entered was: tomato
Statistics after input:
capacity: 15
max size: 4294967293
size: 6
length: 6
empty: false

The remaing string is: soup
capacity: 15
max size: 4294967293
size: 4
length: 4
empty: false

string1 is now: soup1234567890abcdefghijklmnopqrstuvwxyz1234567890
capacity: 63
max size: 4294967293
size: 50
length: 50
empty: false

Stats after resizing by (length + 10):
capacity: 63
max size: 4294967293
size: 60
length: 60
empty: false
```

شکل ۵-۱۸ | چاپ خصوصیات رشته.

برنامه رشته تهی **string1** را اعلان (خط 16) و آنرا به تابع **printStatistics** ارسال می‌کند (خط 19). این تابع در خطوط 55-49 یک مراجعه به یک رشته ثابت بعنوان آرگومان دریافت و ظرفیت (با استفاده از تابع عضو **capacity**)، حداکثر سایز (با استفاده از تابع عضو **max_size**)، سایز (با استفاده از تابع عضو **size**)،

www.yourshop.4kia.ir

طول (با استفاده از تابع عضو **length**) و اینکه آیا رشته تهی است یا خیر (با استفاده از تابع عضو **empty**) را چاپ می‌کند. در ابتدای فراخوانی **printStatistics** دیده می‌شود که مقادیر اولیه برای ظرفیت، سایز و طول رشته **string1** همگی صفر است.

طول و سایز صفر بر این نکته دلالت دارند که هیچ کاراکتری در رشته ذخیره نشده است. چون ظرفیت اولیه صفر است، زمانیکه کاراکترها در **string1** جای داده می‌شوند، حافظه برای تطبیق شدن با کاراکترهای جدید، اخذ می‌شود. بخاطر داشته باشد که سایز و طول همیشه با هم یکسان هستند. در این پیاده‌سازی حداکثر سایز 4294967293 است. شی **string1** یک رشته تهی است، از اینرو تابع **empty**، مقدار **true** برگشت می‌دهد.

خط 23 از طریق خط فرمان رشته‌ای را می‌خواند. در این مثال، رشته ورودی "tomato soup" است. بدلیل اینکه کاراکتر فاصله بعنوان حد اعمال می‌شود، فقط "tomato" در **string1** ذخیره می‌گردد، با این وجود، "soup" در بافر ورودی باقی می‌ماند. خط 27 تابع **printStatistics** را برای چاپ آمار متعلق به **string1** فراخوانی می‌کند. توجه کنید که در خروجی، طول برابر 6 و ظرفیت برابر 15 است.

خط 30، رشته "soup" را از بافر ورودی خوانده و آنرا در **string1** ذخیره می‌کند، بنابر این جایگزین "tomato" می‌شود. خط 32 رشته **string1** را به تابع **printStatistics** ارسال می‌کند.

خط 32 از عملگر سربارگذاری شده += برای پیوند دادن یک رشته 46 کاراکتری به **string1** استفاده کرده است. خط 32 رشته **string1** را به **printStatistics** ارسال می‌کند. دقت کنید که ظرفیت به 63 عنصر و طول به 50 افزایش یافته است.

خط 40 از تابع عضو **resize** برای افزایش طول **string1** به میزان 10 کاراکتر استفاده کرده است. عناصر اضافی با کاراکترهای **null** تنظیم می‌شود. دقت کنید که در خروجی ظرفیت تغییری نکرده و طول به 60 رسیده است.

۱۸-۷ یافتن رشته‌ها و کاراکترها در یک رشته

کلاس رشته دارای توابع عضو ثابت (***const***) برای یافتن زیررشته‌ها و کاراکترها در یک رشته است. برنامه شکل ۶-۱۸ به بررسی توابع با قابلیت یافتن رشته پرداخته است.

```
1  // Fig. 18.6: Fig18_06.cpp
2  // Demonstrating the string find member functions.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <string>
8  using std::string;
9
10 int main()
11 {
12    string string1( "noon is 12 pm; midnight is not." );
13    int location;
```

www.yourshop.4kia.ir

```
14
15    // find "is" at location 5 and 25
16    cout << "Original string:\n" << string1
17       << "\n\n(find) \"is\" was found at: " << string1.find( "is" )
18       << "\n(rfind) \"is\" was found at: " << string1.rfind( "is" );
19
20    // find 'o' at location 1
21    location = string1.find_first_of( "misop" );
22    cout << "\n\n(find_first_of) found '" << string1[ location ]
23       << "' from the group \"misop\" at: " << location;
24
25    // find 'o' at location 29
26    location = string1.find_last_of( "misop" );
27    cout << "\n\n(find_last_of) found '" << string1[ location ]
28         << "' from the group \"misop\" at: " << location;
29
30    // find '1' at location 8
31    location = string1.find_first_not_of( "noi spm" );
32    cout << "\n\n(find_first_not_of) '" << string1[ location ]
33       << "' is not contained in \"noi spm\" and was found at:"
34       << location;
35
36    // find '.' at location 12
37    location = string1.find_first_not_of( "12noi spm" );
38    cout << "\n\n(find_first_not_of) '" << string1[ location ]
39       << "' is not contained in \"12noi spm\" and was "
40       << "found at:" << location << endl;
41
42    // search for characters not in string1
43    location = string1.find_first_not_of(
44       "noon is 12 pm; midnight is not." );
45    cout << "\nfind_first_not_of(\"noon is 12 pm; midnight is not.\")"
46       << " returned: " << location << endl;
47    return 0;
48 } // end main
```

```
Original string:
noon is 12 pm; midnight is not.

(find) "is"was found at: 5
(rfind) "is"was found at: 25

(find_first_of) found 'O' from the group "misop" at: 1

(find_first_not_of) '1' is not contained in "noi spm" and was found at: 8

(find_first_not_of) '.' is not contained in "12noi spm" and was found at:
12

find_first_not_of("noon is 12 pm; midnight is not.") returned: -1
```

شکل ٦-١٨ | توابع یافتن رشته.

در خط 12 رشته **string1** اعلان و مقداردهی اولیه شده است. خط 17 مبادرت به یافتن "is" در رشته **string1** با استفاده از تابع **find** کرده است. اگر "is" پیدا شود، شاخص موقعیت شروع از آن رشته برگشت داده می‌شود. اگر رشته پیدا نشود، مقدار **string::npos** (یک ثابت استاتیک عمومی تعریف شده در کلاس **string**) برگشت داده خواهد شد. این مقدار توسط توابع مرتبط با یافتن رشته برگشت داده می‌شود تا نشان داده شود که زیررشته یا کاراکتر در رشته پیدا نشده است.

www.yourshop.4kia.ir

خط 18 از تابع عضو **find**، برای جستجوی پس‌گشت (یعنی از راست به چپ) رشته **string1** استفاده کرده است. اگر "is" پیدا شود، شاخص موقعیت آن برگشت داده می‌شود. اگر رشته پیدا نشود، **string::npos** برگشت داده خواهد شد.

خط 21 از تابع عضو **find_first_of** برای یافتن اولین پیشامد در رشته **string1** از هر کاراکتری در "misop" استفاده کرده است. جستجو از ابتدای **string1** شروع می‌شود. کاراکتر "o" در عنصر 1 پیدا شده است.

خط 26 از تابع عضو **find_last_of** برای یافتن آخرین پیشامد در رشته **string1** از هر کاراکتری در "misop" استفاده کرده است. جستجو از انتهای **string1** شروع می‌شود. کاراکتر 'o' در عنصر 29 پیدا شده است.

خط 31 از تابع عضو **find_first_not_of** برای یافتن اولین کاراکتر در **string1** که حاوی "noi spm" نیست، استفاده کرده است. کاراکتر '1' در عنصر 8 پیدا شده است. جستجو از ابتدا **string1** شروع می‌شود.

خط 37 از تابع **find_first_not_of** برای یافتن اولین کاراکتر که در "12noi spm" وجود ندارد، استفاده کرده است. کاراکتر '.' در عنصر 12 پیدا شده است. جستجو از انتهای رشته صورت گرفته است.

خطوط 33-43 از تابع عضو **find_first_not_of** برای یافتن اولین کاراکتری که در "noon is 12 pm; midnight is not". وجود ندارد استفاده کرده‌اند. در این مورد، رشته جستجو شده حاوی هر کاراکتر مشخص شده در آرگومان رشته است. بدلیل اینکه کاراکتری پیدا نشده است، **string::npos** برگشت داده شده است (که در این مورد، مقدار 1- دارد).

۸-۱۸ جایگزین ساختن کاراکترها در یک رشته

برنامه شکل ۷-۱۸ به بررسی توابع عضو رشته که در ارتباط با جایگزین‌سازی و پاک کردن کاراکترها هستند، پرداخته است. خطوط 17-13 رشته **string1** را اعلان و مقداردهی اولیه کرده‌اند. خط 23 از تابع عضو **erase** برای پاک کردن هر چیزی از موقعیت 62 تا انتهای رشته **string1** استفاده کرده است.

خطوط 36-29 از **find** برای یافتن هر پیشامدی از کاراکتر فاصله استفاده کرده‌اند. سپس هر فاصله با یک نقطه توسط تابع **replace** جایگزین شده است. تابع **replace** سه آرگومان دریافت می‌کند: شاخص کاراکتر در رشته که بایستی جایگزینی از آنجا آغاز شود، تعداد کاراکترها برای جایگزینی و رشته جایگزین. تابع عضو **find** در زمانیکه کاراکتر مورد جستجو پیدا نشود، **string::npos** برگشت می‌دهد. در خط 35، عدد 1 به **position** افزوده شده تا جستجو از موقعیت کاراکتر بعدی ادامه یابد.

www.yourshop.4kia.ir

در خطوط 48-40 از تابع **find** برای یافتن هر نقطه و تابع سربارگذاری شده **replace** برای جایگزین ساختن هر نقطه و کاراکتر بعدی آن با دو سیمکولن استفاده شده است. آرگومان‌های ارسالی به این نسخه از تابع **replace** عبارتند از شاخص عنصر در محلی که عملیات جایگزینی آغاز می‌شود، تعداد کاراکترهای جایگزین شونده، کاراکتر جایگزین شونده، عنصر در رشته کاراکتری در محلی که زیررشته جایگزین شروع شده و تعداد کاراکترها در رشته کاراکتری جایگزین شونده، برای استفاده.

```
1  // Fig. 18.7: Fig18_07.cpp
2  // Demonstrating string member functions erase and replace.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <string>
8  using std::string;
9
10 int main()
11 {
12    // compiler concatenates all parts into one string
13    string string1( "The values in any left subtree"
14       "\nare less than the value in the"
15       "\nparent node and the values in"
16       "\nany right subtree are greater"
17       "\nthan the value in the parent node" );
18
19    cout << "Original string:\n" << string1 << endl << endl;
20
21    // remove all characters from (and including) location 62
22    // through the end of string1
23    string1.erase( 62 );
24
25    // output new string
26    cout << "Original string after erase:\n" << string1
27       << "\n\nAfter first replacement:\n";
28
29    int position = string1.find( " " ); // find first space
30
31    // replace all spaces with period
32    while ( position != string::npos )
33    {
34       string1.replace( position, 1, "." );
35       position = string1.find( " ", position + 1 );
36    } // end while
37
38    cout << string1 << "\n\nAfter second replacement:\n";
39
40    position = string1.find( "." ); // find first period
41
42    // replace all periods with two semicolons
43    // NOTE: this will overwrite characters
44    while ( position != string::npos )
45    {
46       string1.replace( position, 2, "xxxxx;;yyy", 5, 2 );
47       position = string1.find( ".", position + 1 );
48    } // end while
49
50    cout << string1 << endl;
51    return 0;
52 } // end main
```

```
Original string:
The values in any left subtree
are less than the value in the
parent node and the values in
any right subtree are greater
than the value in the parent node
```

www.yourshop.4kia.ir

```
Original string after erase:
The values in any left subtree
are less than the value in the

After first replacement:
The.values.in.any.left.subtree
are.less.than.the.value.in.the

After second replacement:
The;;alues;;n;;ny;;eft;;ubtree
are;;ess;;han;;he;;alue;;n;;he
```

شکل ۷-۱۸ | بررسی توابع erase و replace.

۹-۱۸ درج کاراکترها در یک رشته

کلاس رشته دارای توابع عضوی برای درج کاراکتر در یک رشته است. برنامه شکل ۸-۱۸ به بررسی قابلیت‌های درج کلاس رشته پرداخته است.

برنامه مبادرت به اعلان، مقداردهی اولیه سپس چاپ رشته‌های **string1**، **string2**، **string3** و **string4** کرده است. خط 22 از تابع عضو **insert** برای درج محتویات **string2** قبل از عنصر 10 در رشته **string1** استفاده کرده است.

خط 25 با استفاده از تابع **insert**، مبادرت به درج **string4** قبل از عنصر 3 رشته **string3** کرده است. دو آرگومان آخر مشخص کننده عنصر شروع و پایان **string4** هستند که بایستی درج شوند. استفاده از **string::npos** سبب می‌شود تا کل رشته درج شود.

```
1  // Fig. 18.8: Fig18_08.cpp
2  // Demonstrating class string insert member functions.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <string>
8  using std::string;
9
10 int main()
11 {
12    string string1( "beginning end" );
13    string string2( "middle " );
14    string string3( "12345678" );
15    string string4( "xx" );
16
17    cout << "Initial strings:\nstring1: " << string1
18       << "\nstring2: " << string2 << "\nstring3: " << string3
19       << "\nstring4: " << string4 << "\n\n";
20
21    // insert "middle" at location 10 in string1
22    string1.insert( 10, string2 );
23
24    // insert "xx" at location 3 in string3
25    string3.insert( 3, string4, 0, string::npos );
26
27    cout << "Strings after insert:\nstring1: " << string1
28       << "\nstring2: " << string2 << "\nstring3: " << string3
29       << "\nstring4: " << string4 << endl;
30    return 0;
31 } // end main
```

```
Initial strings:
```

www.yourshop.4kia.ir

```
string1: beginning end
string2: middle
string3: 12345678
string4: xx

Strings after insert:
string1: beginning middle end
string2: middle
string3: 123xx45678
string4: xx
```

شکل ۸-۱۸ | بررسی تابع عضو insert.

۱۰-۱۸ تبدیل به سبک رشته‌های C

کلاس رشته دارای توابع عضوی است که می‌توانند شی‌های کلاس رشته را تبدیل به رشته‌های مبتنی بر اشاره‌گر سبک C (یعنی C-style) کنند. همانطوری که قبلاً هم گفته شدهٔ، برخلاف رشته‌های مبتنی بر اشاره‌گر، رشته‌ها ضرورتاً با **null** خاتمه پیدا نمی‌کنند. این توابع تبدیل کننده زمانی سودمند هستند که تابعی یک رشته مبتنی بر اشاره‌گر بعنوان آرگومان دریافت کند. برنامه شکل ۹-۱۸ به بررسی تبدیل رشته‌ها به رشته‌های مبتنی بر اشاره‌گر پرداخته است.

برنامه مبادرت به اعلان یک رشته، یک **int** و دو اشاره‌گر **char** کرده است (خطوط 15-12). رشته **string1** با "STRINGS"، **ptr1** با 0 و **length** با طول **string1** مقداردهی اولیه شده است. حافظه به میزان کافی برای نگهداری یک رشته مبتنی بر اشاره‌گر معادل رشته **string1** بصورت دینامیکی اخذ شده و به اشاره‌گر **ptr2** از نوع **char** الصاق می‌شود.

خط 18 از تابع عضو **copy** برای کپی شی **string1** به آرایه **char** مورد اشاره **ptr2** استفاده کرده است.

خط 19 بصورت غیراتوماتیک یک کاراکتر خاتمه دهنده **null** در آرایه مورد اشاره **ptr2** قرار می‌دهد.

خط 23 از تابع **c_str** برای کپی شی **string1** و افزودن اتوماتیک کاراکتر خاتمه دهنده **null** استفاده کرده است. این تابع یک **const char *** برگشت می‌دهد که توسط عملگر درج استریم چاپ می‌شود.

خط 29 مبادرت به تخصیص **const char * ptr1** به اشاره‌گر برگشتی توسط تابع عضو **data** می‌کند. این تابع عضو یک کاراکتر غیر **null** خاتمه دهنده آرایه کاراکتری سبک C برگشت می‌دهد. دقت کنید که در این مثال تغییری در رشته **string1** بوجود نیاورده‌ایم.

اگر رشته **string1** دچار تغییر شود (برای مثال حافظه دینامیکی رشته آدرس خود را به علت فراخوانی یک تابع عضو همانند **string1.insert(0, "abcd");** تغییر دهد)، اشاره‌گر **ptr1** نامعتبر خواهد شد، که می‌تواند نتایج غیرقابل پیش‌بینی بوجود آورد.

خطوط 33-32 از محاسبه اشاره‌گر برای چاپ آرایه کاراکتری مورد اشاره توسط **ptr1** استفاده کرده‌اند.

در خطوط 36-35 رشته سبک C مورد اشاره توسط **ptr2** چاپ شده و حافظه اخذ شده حذف می‌شود تا از فقدان حافظه جلوگیری گردد.

```
1  // Fig. 18.9: Fig18_09.cpp
```

www.yourshop.4kia.ir

```
// Converting to C-style strings.
#include <iostream>
using std::cout;
using std::endl;

#include <string>
using std::string;

int main()
{
   string string1( "STRINGS" ); // string constructor with char* arg
   const char *ptr1 = 0; // initialize *ptr1
   int length = string1.length();
   char *ptr2 = new char[ length + 1 ]; // including null

   // copy characters from string1 into allocated memory
   string1.copy( ptr2, length, 0 ); // copy string1 to ptr2 char*
   ptr2[ length ] = '\0'; // add null terminator

   cout << "string string1 is " << string1
      << "\nstring1 converted to a C-Style string is "
      << string1.c_str() << "\nptr1 is ";

   // Assign to pointer ptr1 the const char * returned by
   // function data(). NOTE: this is a potentially dangerous
   // assignment. If string1 is modified, pointer ptr1 can
   // become invalid.
   ptr1 = string1.data();

   // output each character using pointer
   for ( int i = 0; i < length; i++ )
      cout << *( ptr1 + i ); // use pointer arithmetic

   cout << "\nptr2 is " << ptr2 << endl;
   delete [] ptr2; // reclaim dynamically allocated memory
   return 0;
} // end main
```

```
string string1 is STRINGS
string1 converted to a C-Style string is STRINGS
ptr1 is STRINGS
ptr2 is STRINGS
```

شکل ۹-۱۸ | تبدیل رشته به رشته‌ها و آرایه‌های کاراکتری سبک C.

۱۱-۱۸ تکرار شونده‌ها

کلاس رشته دارای *تکرار شونده‌های* (*iterators*) برای پیمایش به سمت جلو و عقب در میان رشته‌ها است. تکرار شونده‌ها دسترسی به کاراکترهای مجزا را با گرامری که شبیه عملیات اشاره‌گر است، فراهم می‌آورند. تکرار شونده‌ها بررسی محدودهٔ را انجام نمی‌دهند. توجه کنید که در این بخش مثال‌های مطرح شده فقط جنبه آشنا کردن شما با تکرار شونده‌ها را دارند. برنامه شکل ۱۰-۱۸ به بررسی تکرار شونده‌ها پرداخته است.

```
// Fig. 18.10: Fig18_10.cpp
// Using an iterator to output a string.
#include <iostream>
using std::cout;
using std::endl;

#include <string>
using std::string;

int main()
{
```

www.yourshop.4kia.ir

```
12    string string1( "Testing iterators" );
13    string::const_iterator iterator1 = string1.begin();
14
15    cout << "string1 = " << string1
16       << "\n(Using iterator iterator1) string1 is: ";
17
18    // iterate through string
19    while ( iterator1 != string1.end() )
20    {
21       cout << *iterator1; // dereference iterator to get char
22       iterator1++; // advance iterator to next char
23    } // end while
24
25    cout << endl;
26    return 0;
27 } // end main
```

```
string1 = Testing iterators
(Using iterator iterator1) string1 is: Testing iterators
```

شکل ۱۰-۱۸ | استفاده از تکرار شونده‌ها در چاپ رشته.

خطوط 13-12 رشته **string1** و تکرار شونده **string::const_iterator iterator1** را اعلان کرده‌اند. **const_iterator** تکرار شونده‌ای است که نمی‌تواند رشته را تغییر دهد. تکرار شونده **iterator1** با ابتدای **string1** توسط تابع عضو **begin** مقداردهی اولیه شده است.

دو نسخه از تابع **begin** وجود دارد، یکی که یک تکرار شونده برای حرکت در میان یک رشته غیرثابت برگشت می‌دهد و یک نسخه ثابت که یک **const_iterator** برای حرکت در میان یک رشته ثابت برگشت می‌دهد. خط 15 رشته **string1** را چاپ می‌کند.

خطوط 23-19 با استفاده از **iterator1** در میان رشته **string1** حرکت می‌کنند. تابع عضو **end** یک تکرار شونده (یا یک **const_iterator**) برای موقعیت قبل از آخرین عنصر رشته **string1** برگشت می‌دهد.

کلاس **string** دارای توابع عضو **read** و **rbegin** برای دستیابی به کاراکترهای مجزای رشته بصورت معکوس از انتها به سمت ابتدای رشته است. توابع عضو **rend** و **rbegin** می‌توانند **reverse_iterators** و **const_reverse_iterators** برگشت دهند.

۱۲-۱۸ پردازش استریم رشته

علاوه بر استریم استاندارد I/O و استریم فایل I/O، استریم I/O در C++ حاوی قابلیت‌های برای ورودی از و خروجی به رشته‌های موجود در حافظه است. غالباً از این قابلیت‌ها بعنوان I/O حافظه یا پردازش استریم رشته یاد می‌شود.

ورودی از یک رشته توسط کلاس **istringstream** و خروجی به رشته توسط کلاس **ostringstream** پشتیبانی می‌شود. اسامی کلاس **istringstream** و **ostringstream** در واقع اسامی دیگری هستند که توسط *typedef*های زیر تعریف شده‌اند

```
typedef basic_istringstream< char > istringstream;
typedef basic_ostringstream< char > ostringstream;
```

www.yourshop.4kia.ir

الگوهای کلاس **basic_istringstream** و **basic_ostringstream** همان قابلیت‌های کلاس **istream** و **ostream** را به همراه تعدادی توابع عضو خاص کار با حافظه هستند. برنامه‌های که از قالب‌بندی حافظه استفاده می‌کنند بایستی حاوی فایل سرآیند <**ssream**> و <**iostream**> باشند.

یکی از کاربردهای این تکنیک‌ها، اعتبارسنجی داده‌ها می‌باشد. برنامه می‌تواند کل یک خط را در یک زمان از استریم ورودی بدرون یک رشته بخواند، سپس یک روتین اعتبارسنجی می‌تواند بدقت محتویات رشته را بررسی کرده و در صورت نیاز، داده را اصلاح (یا تعمیر) نماید. سپس برنامه می‌تواند به دریافت رشته ادامه دهد با اطمینان از اینکه داده ورودی در قالب صحیح وارد می‌شود.

خارج کردن (چاپ) یک رشته روش مناسبی برای بهره‌مند شدن از مزایای خروجی قالب‌بندی شده استریم‌ها در C++ است. داده موجود در یک رشته می‌تواند مهیای ویرایش گردد.

برنامه شکل ۱۱-۱۸ به بررسی یک شی **ostringstream** پرداخته است. برنامه شی **outputString** را ایجاد کرده (خط 15) و از عملگر درج استریم برای چاپ دنباله‌ای از رشته‌ها و مقادیر عددی استفاده می‌کند.

خطوط 28-27 مبادرت به چاپ رشته **string1**، رشته **string2**، رشته **string3**، **double1**، رشته **string4**، **integer**، رشته **string5** و آدرس **int** چاپ می‌کنند، که همگی **outputString** در حافظه هستند. خط 31 از عملگر درج استریم و فراخوانی **outputString.str()** برای نمایش یک کپی از رشته ایجاد شده در خطوط 28-27 کرده است. خط 34 به بررسی این مطلب پرداخته است که داده می‌تواند به رشته‌ای در حافظه الصاق شود که اینکار به آسانی و با صدور یک عملیات درج استریم دیگر به **outputString** صورت می‌گیرد. خطوط 36-35 رشته **outputString** را پس از الصاق کاراکترهای اضافی، چاپ می‌کنند.

```
1  // Fig. 18.11: Fig18_11.cpp
2  // Using a dynamically allocated ostringstream object.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <string>
8  using std::string;
9
10 #include <sstream> // header file for string stream processing
11 using std::ostringstream; // stream insertion operators
12
13 int main()
14 {
15    ostringstream outputString; // create ostringstream instance
16
17    string string1( "Output of several data types " );
18    string string2( "to an ostringstream object:" );
19    string string3( "\n        double: " );
20    string string4( "\n           int: " );
21    string string5( "\naddress of int: " );
22
23    double double1 = 123.4567;
```

www.yourshop.4kia.ir

```
24    int integer = 22;
25
26    // output strings, double and int to ostringstream outputString
27    outputString << string1 << string2 << string3 << double1
28       << string4 << integer << string5 << &integer;
29
30    // call str to obtain string contents of the ostringstream
31    cout << "outputString contains:\n" << outputString.str();
32
33    // add additional characters and call str to output string
34    outputString << "\nmore characters added";
35    cout << "\n\nafter additional stream insertions,\n"
36       << "outputString contains:\n" << outputString.str() << endl;
37    return 0;
38 } // end main
```

```
outputString contains:
Output of several data types to an ostringstream object:
        double: 123.457
           int: 22
address of int: 0012F540

after additional stream insertions,
outputString contains:
Output of several data types to an ostringstream object:
        double: 123.457
           int: 22
address of int: 0012F540
more characters added
```

شکل ۱۱-۱۸ | استفاده از شی ostringstream اخذ شده به روش دینامیکی.

یک شی **istringstream** داده را از یک رشته موجود وارد متغیرهای برنامه می‌کند. داده‌های ذخیره شده در یک شی **istringstream** نقش کاراکتر دارند. ورودی از شی **istringstream** همانند ورودی از هر فایلی عمل می‌کند. انتهای رشته توسط شی **istringstream** همانند انتهای فایل تفسیر می‌شود.

برنامه شکل ۱۲-۱۸ به بررسی ورودی از یک شی **istringstream** پرداخته است. خطوط 16-15 یک رشته **input** حاوی داده و یک شی **inpuString** ایجاد شده با داده موجود در رشته **input** ایجاد می‌کنند. رشته **input** حاوی داده زیر است.

```
Input test 123 4.7 A
```

که به هنگام خوانده شدن بعنوان ورودی به برنامه، متشکل از دو رشته ("test" و "Input")، یک (123) **int**، یک **double**(4.7) و یک **char** ('A') است. این کاراکترها به متغیرهای **string1**، **string2**، **integer**، **double1** و **character** در خط 23 انتقال داده می‌شوند.

```
1  // Fig. 18.12: Fig18_12.cpp
2  // Demonstrating input from an istringstream object.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <string>
8  using std::string;
9
10 #include <sstream>
11 using std::istringstream;
12
13 int main()
14 {
15    string input( "Input test 123 4.7 A" );
16    istringstream inputString( input );
17    string string1;
```

www.yourshop.4kia.ir

```
18    string string2;
19    int integer;
20    double double1;
21    char character;
22
23    inputString >> string1 >> string2 >> integer >> double1 >> character;
24
25    cout << "The following items were extracted\n"
26       << "from the istringstream object:" << "\nstring: " << string1
27       << "\nstring: " << string2 << "\n   int: " << integer
28       << "\ndouble: " << double1 << "\n  char: " << character;
29
30    // attempt to read from empty stream
31    long value;
32    inputString >> value;
33
34    // test stream results
35    if ( inputString.good() )
36       cout << "\n\nlong value is: " << value << endl;
37    else
38       cout << "\n\ninputString is empty" << endl;
39
40    return 0;
41 } // end main
```

```
The follwing items were extracted
from the istringstream object:
string: Input
string: test
   int: 123
double: 4.7
  char: A

inputString is empty
```

شکل ۱۲-۱۸ | توصیف عملکرد ورودی از طریق شی istringstream.

سپس داده‌ها توسط خطوط 28-25 خارج می‌شوند. برنامه مبادرت به خواندن مجدد **inputString** در خط 32 می‌کند. شرط **if** در خط 35 از تابع **good** برای تست اینکه آیا داده‌ای باقی مانده است، استفاده می‌کند. چون هیچ داده‌ای باقی نمانده است، تابع مقدار **false** برگشت داده و بخش **else** از عبارت **if...else** اجرا می‌شود.

www.yourshop.4kia.ir