Character sequences

The `string` class has been briefly introduced in an earlier chapter. It is a very powerful class to handle and manipulate strings of characters. However, because strings are, in fact, sequences of characters, we can represent them also as plain arrays of elements of a character type.

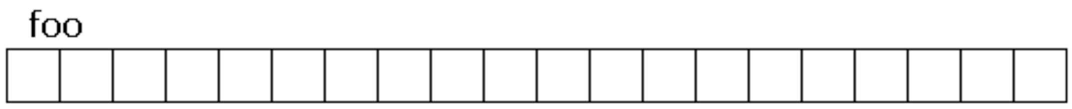
在前面的章节中已经简要介绍了' string '类。它是一个非常强大的类，用于处理和操作字符串。但是，因为字符串实际上是字符序列，所以我们也可以将它们表示为字符类型元素的普通数组

For example, the following array:

```
char foo [20];
```

is an array that can store up to 20 elements of type `char`. It can be represented as:

是一个最多可存储20个char类型元素的数组。可以表示为:



Therefore, this array has a capacity to store sequences of up to 20 characters. But this capacity does not need to be fully exhausted: the array can also accommodate shorter sequences. For example, at some point in a program, either the sequence `"Hello"` or the sequence `"Merry Christmas"` can be stored in `foo`, since both would fit in a sequence with a capacity for 20 characters.

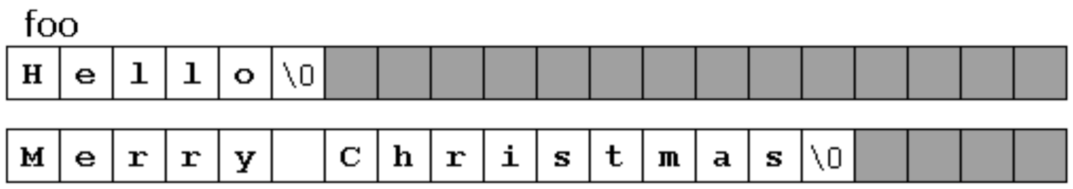
因此，该数组具有最多存储20个字符的序列的容量。但是这个容量不需要完全耗尽:数组也可以容纳更短的序列。例如，在程序的某个点上，序列“Hello”或序列“Merry Christmas”可以存储在foo中，因为它们都适合容纳20个字符的序列。

By convention, the end of strings represented in character sequences is signaled by a special character: the *null character*, whose literal value can be written as `'\0'` (backslash, zero).

按照惯例，用字符序列表示的字符串的结尾由一个特殊字符表示: 空字符，其字面量可以写成'\0'(反斜杠，0)。

In this case, the array of 20 elements of type `char` called `foo` can be represented storing the character sequences `"Hello"` and `"Merry Christmas"` as:

在这种情况下，由20个char类型元素组成的foo数组可以表示为将字符序列“Hello”和“Merry Christmas”存储为:



Notice how after the content of the string itself, a null character (`'\0'`) has been added in order to indicate the end of the sequence. The panels in gray color represent `char` elements with undetermined values.

请注意，在字符串本身的内容之后，如何添加一个空字符('\0')，以指示序列的结束。灰色的面板表示值未确定的char元素。

Initialization of null-terminated character sequences

Because arrays of characters are ordinary arrays, they follow the same rules as these. For example, to initialize an array of characters with some predetermined sequence of characters, we can do it just like any other array:

因为字符数组是普通数组，所以它们遵循与这些数组相同的规则。例如，要用预定的字符序列初始化一个字符数组，可以像其他数组一样:

```
char myword[] = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

The above declares an array of 6 elements of type `char` initialized with the characters that form the word `"Hello"` plus a *null character* `'\0'` at the end.

上面声明了一个由char类型的6个元素组成的数组，初始化的字符是构成单词“Hello”的字符加上末尾的空字符“\0”。

But arrays of character elements have another way to be initialized: using *string literals* directly.

但是字符元素数组有另一种初始化方式:直接使用字符串字面值。

In the expressions used in some examples in previous chapters, string literals have already shown up several times. These are specified by enclosing the text between double quotes (`"`). For example:

在前几章的一些示例中使用的表达式中，字符串字面量已经出现过好几次了。它们是通过将文本括在双引号(")之间来指定的。例如:

```
"the result is: "
```

This is a *string literal*, probably used in some earlier example.

这是一个字符串字面量，可能在之前的例子中使用过。

Sequences of characters enclosed in double-quotes (`"`) are *literal constants*. And their type is, in fact, a null-terminated array of characters. This means that string literals always have a null character (`'\0'`) automatically appended at the end.

用双引号(")括起来的字符序列是文字常量。实际上，它们的类型是一个以null结尾的字符数组。这意味着字符串文本总是在末尾自动附加一个空字符('\0')。

Therefore, the array of char elements called `myword` can be initialized with a null-terminated sequence of characters by either one of these two statements:

因此，名为myword的char元素数组可以通过以下两条语句中的任意一条来初始化以null结尾的字符序列:

```
char myword[] = { 'H', 'e', 'l', 'l', 'o', '\0' };
char myword[] = "Hello";
```

In both cases, the array of characters `myword` is declared with a size of 6 elements of type `char`: the 5 characters that compose the word `"Hello"`, plus a final null character (`'\0'`), which specifies the end of the sequence and that, in the second case, when using double quotes (`"`) it is appended automatically.

在这两种情况下,字符数组myword声明一个char类型的6个元素的大小:5字符组成“你好”这个词,加上最后一个null字符(\ 0),它指定序列的末尾,在第二种情况下,当使用双引号(")是自动添加的。

Please notice that here we are talking about initializing an array of characters at the moment it is being declared, and not about assigning values to them later (once they have already been declared). In fact, because string literals are regular arrays, they have the same restrictions as these, and cannot be assigned values.

请注意，这里我们讨论的是在声明字符数组时对其进行初始化，而不是在以后(一旦它们已经声明)给它们赋值。事实上，因为字符串字面值是常规数组，所以它们具有与这些相同的限制，并且不能被赋值。

Expressions (once myword has already been declared as above), such as:

表达式(一旦myword已经如上声明)，例如:

```
myword = "Bye";
myword[] = "Bye";
```

would **not** be valid, like neither would be:

都是无效的，就像两者都不是一样:

```
myword = { 'B', 'y', 'e', '\0' };
```

This is because arrays cannot be assigned values. Note, though, that each of its elements can be assigned a value individually. For example, this would be correct:

这是因为数组不能被赋值。但是请注意，它的每个元素都可以被单独赋值。例如，这是正确的:

```
myword[0] = 'B';
myword[1] = 'y';
myword[2] = 'e';
myword[3] = '\0';
```

Strings and null-terminated character sequences

Plain arrays with null-terminated sequences of characters are the typical types used in the C language to represent strings (that is why they are also known as *C-strings*). In C++, even though the standard library defines a specific type for strings (class `string`), still, plain arrays with null-terminated sequences of characters (C-strings) are a natural way of representing strings in the language; in fact, string literals still always produce null-terminated character sequences, and not `string` objects.

带有以null结尾的字符序列的普通数组是C语言中用来表示字符串的典型类型(这就是为什么它们也被称为C字符串)。在c++中，尽管标准库为字符串定义了一种特定类型(类string)，但带有以空结尾的字符序列的普通数组(C-string)仍然是语言中表示字符串的一种自然方式;事实上，字符串字面值仍然总是产生以null结尾的字符序列，而不是字符串对象。

In the standard library, both representations for strings (C-strings and library strings) coexist, and most functions requiring strings are overloaded to support both.

在标准库中，字符串的两种表示形式(c字符串和标准库字符串)同时存在，大多数需要字符串的函数都被重载以支持这两种形式。

For example, `cin` and `cout` support null-terminated sequences directly, allowing them to be directly extracted from `cin` or inserted into `cout`, just like strings. For example:

例如，cin和cout直接支持以null结尾的序列，允许它们直接从cin中提取或插入cout，就像字符串一样。例如:

```
// strings and NTCS:
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  char question1[] = "What is your name? ";
  string question2 = "Where do you live? ";
  char answer1 [80];
  string answer2;
  cout << question1;
  cin >> answer1;
  cout << question2;
  cin >> answer2;
  cout << "Hello, " << answer1;
  cout << " from " << answer2 << "!\n";
  return 0;
}
```

In this example, both arrays of characters using null-terminated sequences and strings are used. They are quite interchangeable in their use together with `cin` and `cout`, but there is a notable difference in their declarations: arrays have a fixed size that needs to be specified either implicit or explicitly when declared; `question1` has a size of exactly 20 characters (including the terminating null-characters) and `answer1` has a size of 80 characters; while strings are simply strings, no size is specified. This is due to the fact that strings have a dynamic size determined during runtime, while the size of arrays is determined on compilation, before the program runs.

在本例中，使用了以null结尾的序列和字符串的字符数组。在与cin和cout一起使用时，它们是可以互换的，但它们的声明有一个显著的区别:数组的大小是固定的，需要在声明时隐式或显式指定;问题1的长度正好是20个字符(包括结尾的空字符)，而answer1的长度是80个字符;虽然字符串只是字符串，但没有指定大小。这是因为字符串的大小是在运行时确定的，而数组的大小是在程序运行之前编译时确定的。

In any case, null-terminated character sequences and strings are easily transformed from one another:

在任何情况下，以空结尾的字符序列和字符串很容易相互转换:

Null-terminated character sequences can be transformed into strings implicitly, and strings can be transformed into null-terminated character sequences by using either of `string`'s member functions `c_str` or `data`:

以空结尾的字符序列可以隐式转换为字符串，而字符串可以通过使用string的成员函数c_str或data转换为以空结尾的字符序列:

```
char myntcs[] = "some text";
string mystring = myntcs;  // convert c-string to string
cout << mystring;          // printed as a library string
cout << mystring.c_str();  // printed as a c-string
```

(note: both `c_str` and `data` members of `string` are equivalent)

(注意:c_str和string的data成员是等价的)