

As you may already know, the C++ Standard Library implements a powerful `string` class, which is very useful 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 `char` elements.

For example, the following array:

```
char jenny [20];
```

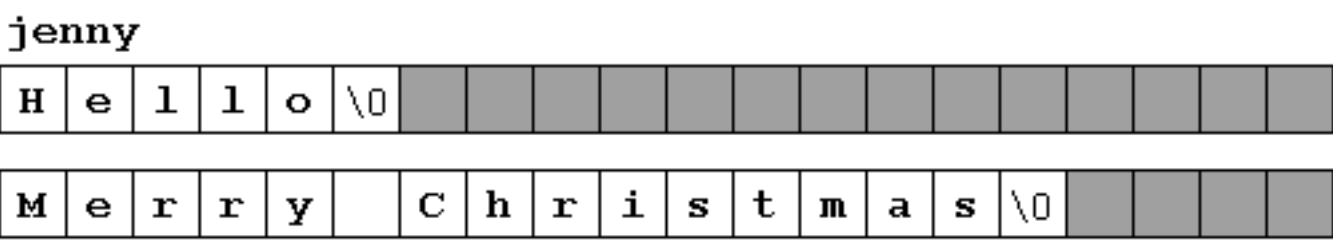
is an array that can store up to 20 elements of type `char`. It can be represented as:



Therefore, in this array, in theory, we can store sequences of characters up to 20 characters long. But we can also store shorter sequences. For example, `jenny` could store at some point in a program either the sequence `"Hello"` or the sequence `"Merry Christmas"`, since both are shorter than 20 characters.

Therefore, since the array of characters can store shorter sequences than its total length, a special character is used to signal the end of the valid sequence: the *null character*, whose literal constant can be written as `'\0'` (backslash, zero).

Our array of 20 elements of type `char`, called `jenny`, can be represented storing the characters sequences `"Hello"` and `"Merry Christmas"` as:



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

Initialization of null-terminated character sequences

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

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

In this case we would have declared 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. But arrays of `char` elements have an additional method to initialize their values: using string literals.

In the expressions we have used in some examples in previous chapters, constants that represent entire strings of characters have already showed up several times. These are specified enclosing the text to become a string literal between double quotes (`"`). For example:

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

is a constant string literal that we have probably used already.

Double quoted strings (`"`) are literal constants whose type is in fact a null-terminated array of characters. So string literals enclosed between double quotes always have a null character (`'\0'`) automatically appended at the end.

Therefore we can initialize the array of `char` elements called `myword` with a null-terminated sequence of characters by either one of these two methods:

```
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.

Please notice that we are talking about initializing an array of characters in the moment it is being declared, and not about assigning values to them once they have already been declared. In fact because this type of null-terminated arrays of characters are regular arrays we have the same restrictions that we have with any other array, so we are not able to copy blocks of data with an assignment operation.

Assuming `mystext` is a `char[]` variable, expressions within a source code like:

```
mystext = "Hello";
mystext[] = "Hello";
```

would not be valid, like neither would be:

```
mystext = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

The reason for this may become more comprehensible once you know a bit more about pointers, since then it will be clarified that an array is in fact a constant pointer pointing to a block of memory.

Using null-terminated sequences of characters

Null-terminated sequences of characters are the natural way of treating strings in C++, so they can be used as such in many procedures. In fact, regular string literals have this type (`char[]`) and can also be used in most cases.

For example, `cin` and `cout` support null-terminated sequences as valid containers for sequences of characters, so they can be used directly to extract strings of characters from `cin` or to insert them into `cout`. For example:

```
// null-terminated sequences of characters
#include <iostream>
using namespace std;

int main ()
{
    char question[] = "Please, enter your first name: ";
    char greeting[] = "Hello, ";
    char yourname [80];
    cout << question;
    cin >> yourname;
    cout << greeting << yourname << "!";
    return 0;
}
```

Please, enter your first name: John
Hello, John!

As you can see, we have declared three arrays of `char` elements. The first two were initialized with string literal constants, while the third one was left uninitialized. In any case, we have to specify the size of the array: in the first two (`question` and `greeting`) the size was implicitly defined by the length of the literal constant they were initialized to. While for `yourname` we have explicitly specified that it has a size of 80 chars.

Finally, sequences of characters stored in `char` arrays can easily be converted into `string` objects just by using the assignment operator:

```
string mystring;
char myntcs[]="some text";
mystring = myntcs;
```