**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:

|  |  |  |
| --- | --- | --- |
|  | 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.

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

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:

|  |  |  |
| --- | --- | --- |
|  | "the result is: " |  |

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

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.

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:

|  |  |  |
| --- | --- | --- |
| 1 2 | 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 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.

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

|  |  |  |
| --- | --- | --- |
| 1 2 | myword = "Bye";  myword[] = "Bye"; |  |

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

|  |  |  |
| --- | --- | --- |
|  | 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:

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 | myword[0] = 'B';  myword[1] = 'y';  myword[2] = 'e';  myword[3] = '\0'; |  |

Notice how you can print out two aforementioned *myword* statements:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3  4  5  6  7 | #include <iostream>  using namespace std;  int main()  {  char myword[] = {'H', 'e', 'l', 'l', 'o', '\0'};  cout << myword << endl;  } | #include <iostream>  using namespace std;  int main()  {  char myword[] = "Hello";  cout << myword << endl;  } |  |
|  | Output:  Hello | Output:  Hello |  |

In below code which is on the left depicts the case where if you do not include the null-terminated character:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3  4  5  6  7 | #include <iostream>  using namespace std;  int main()  {  char myword[] = {'H', 'e', 'l', 'l', 'o'};  cout << myword << endl;  } | #include <iostream>  using namespace std;  int main()  {  char myword[] = "\0\0\0\0\0\0\0\0";  cout << myword << endl;  } |  |
|  | Output:  Hello◄ё | Output: |  |

Above on the right code it is shown whether what happens if you print null character(s).

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3  4  5  6  7  8 | #include <iostream>  using namespace std;  int main()  {  char myword[] = {'H', 'e', 'l', 'l', 'o'};  for (int i = 0; i < 5; i++)  cout << myword[i];  } | #include "iostream"  using namespace std;  int main()  {  char myword[] = "Hello";  for (int i = 0; i < 5; i++)  cout << myword[i];  } |  |
|  | Output:  Hello | Output:  Hello |  |
| 1  2  3  4  5  6  7  8 | #include <iostream>  using namespace std;  int main()  {  char myword[] = {'H', 'e', 'l', 'l', 'o'};  for (int i = 0; i < 6; i++)  cout << myword[i];  } | #include "iostream"  using namespace std;  int main()  {  char myword[] = "Hello";  for (int i = 0; i < 6; i++)  cout << myword[i];  } |  |
|  | Output:  Hello◄ | Output:  Hello |  |
| 1  2  3  4  5  6  7  8 | #include <iostream>  using namespace std;  int main()  {  char myword[] = {'H', 'e', 'l', 'l', 'o', '\0'};  for (int i = 0; i < 6; i++)  cout << myword[i];  } | #include "iostream"  using namespace std;  int main()  {  char myword[] = "Hello";  for (int i = 0; i < 8; i++)  cout << myword[i];  } |  |
|  | Output:  Hello | Output:  Hello├ |  |

Print out using for loops. But notice how the counter values for two programs which are above is 5, and for the two programs below it is 6 affect them:

Again, it can be explained with the fact that if we declare and initialize at the same time as string literal the null character ('\0') is appended automatically to prevent printing of the undetermined values.