edX (https://www.edx.org) Sequential access | 9.4 Strings | UT.6.01x Co... https://courses.edx.org/courses/UTAustinX/UT...

UTAustinX: UT.6.01x Embedded Systems - Shape the World KarenWest (/dashboard) ▼

Courseware (/courses/UTAustinX/UT.6.01x/1T2014/courseware) Course Info (/courses/UTAustinX/UT.6.01x/1T2014/info)

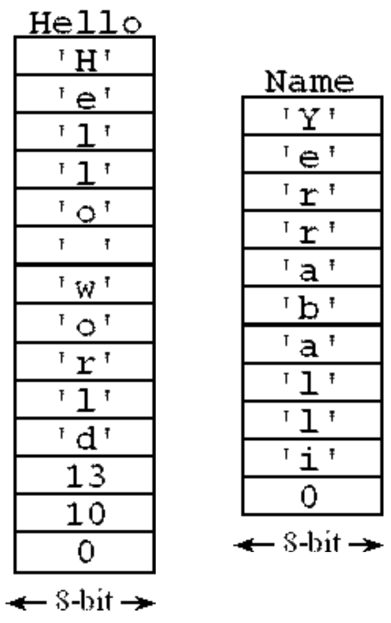
Discussion (/courses/UTAustinX/UT.6.01x/1T2014/discussion/forum) Progress (/courses/UTAustinX/UT.6.01x/1T2014/progress)

Questions (/courses/UTAustinX/UT.6.01x/1T2014/a3da417940af4ec49a9c02b3eae3460b/)

Syllabus (/courses/UTAustinX/UT.6.01x/1T2014/a827a8b3cc204927b6efaa49580170d1/)

A **string** is a data structure with equal size elements that only allows sequential access. The bytes of the string are always read in order from the first to the last. In contrast, an **array** allows random access to any element in any order. The same mechanisms introduced for variable length arrays will also apply to strings. If the string is variable length then we need a way to determine its length. We could place the length in the first position or we could place a termination code (**sentinel**) at the end. In general, we store the length of the string in the first position, when the data can take on any value, negating the possibility of using a termination code. From a programming perspective we access strings in the same manner as arrays. In C, a string of ASCII characters ends with a null character (0). This null character is not considered a character in the string, but a place holder to mark the end of the string. In C, we can define a constant string of ASCII characters with null termination:

```
const char Hello[] = "Hello world\n\r";
const char Name[] = "Yerraballi";
```



When defining constant strings or arrays we must specify their values because the data will be loaded into ROM and cannot be changed at run time. In the previous constant arrays we specified the size; however for constant arrays the size can be left off and the compiler will calculate the size automatically. Notice also the string can contain special characters, some of which are listed in Table 9.1. The `Hello World` string has 13 characters followed by a null (0) termination.

STRINGS

7:58 / 7:58 1.0x

Help

RAMESH YERRABALLI: As a second example of array manipulation,

we will look at an example where we will work with an array of characters.

An array of characters is called a string.

So a string, then, is defined as an array of characters that is null

terminated.

So I can define an array-- here is an example

of an array which describes a name.

So let's say a name.

Now, if I want to leave the size out, I can leave it out

by not putting it in the square brackets.

So I declare an array.

And I can specify-- like, for example, this is John Smith.

So an array declared this way will result in allocation in memory

where we will see a name.

And the location 0 will have J in it.

Notice that the quotes don't go inside.

The location one of the name will have an O and H and N

and space, S, M I, T, H.

And the last location, any time you declare an array, will have a 0 in it.

This is what null termination means.

So null terminated means that the last element will have a 0.

It's automatically inserted by the compiler

when it produces the code as a result of this statement.

So let's look at an example problem where we can use arrays of characters or strings.

So the problem I'm going to look at-- the problem

is count the number of occurrences of the letter 'j' in a given string.

So again, in order to demonstrate this, we will write a subroutine.

I'm going to call this subroutine jcount.

And jcount, obviously, is going to return a number.

I'm going to make it return unsigned long.

Now, I don't have to say unsigned long, but I'm just trying to give you

an example, because the number cannot be a negative number.

So I'm saying it returns an unsigned long.

And it takes as input a string of some arbitrary length--

so I don't specify what the length is-- that

Help

Character	*Escape Sequence*
alert (beep)	**\a**
backslash	\\
backspace	**\b**
carriage return	**\r**
double quote	**\"**
form feed	**\f**
horizontal tab	**\t**
newline	**\n**
null character	**\0**
single quote	**\'**
vertical tab	**\v**
question mark	**\?**

Table 9.1. Escape sequences.

In C, ASCII strings are stored with null-termination. In C, the compiler automatically adds the zero at the end, but in assembly, the zero must be explicitly defined.

Help