# A String ADT

This is a **pair** exercise and must be competed in your **tutorial** or **lab** with your partner.

This week, we're taking some of the new data types that we've seen over the last few weeks, and asking, "can we make that an ADT?"

Strings, for example. We spent a long time looking at strings, and we know that a string is just a piece of data with a well-defined type that has well-defined characteristics (it's a `char *`, with a `'\0'` at the end) and that there are some fairly common operations on Strings.

So, what might a String ADT look like?

Download `String.h`, or copy it into your current directory on a CSE system by running

```
$ cp /web/cs1511/17s2/week09/files/String.h .
```

**Don't change *String.h*!** If you do, all your time-pieces will start ticking backwards.

With the String ADT, you get to decide the implementation. We haven't provided a stub `.c` file; you have to build it yourself. The functions you have to implement are listed below, and in `String.h`.

- `String newString (char *str);`
  Create a new `String`, from a C-style null-terminated array of characters.
- `void destroyString (String s);`
  Release all resources associated with a `String`.
- `int stringLength (String s);`
  Get the length of a `String`.
- `String stringClone (String s);`
  Make a (mutable) copy of a `String`. You should ensure this is a *deep copy* that allows you to dispose of the string you started with.
- `char stringAt (String s, int index);`
  Given a `String`, and an index in the `String`, return the character at that index.

- `int stringFindChar (String s, char find);`
  Given a `String`, and a character to look for in the `String`, return either the index of the character, or `-1` if the character could not be found in the `String`.

- `int stringsEqual (String s1, String s2);`
  Are two `String`s equal?

- `String stringConcat (String s1, String s2);`
  Given two `String`s, concatenates them into a new `String`.

A note on the terminology around ADTs – there are some new words to be familiar with: *constructor*, *destructor*, *setter*, and *getter* all describe the different varieties of functions common in an ADT's interface. `newString` is a constructor; `destroyString` is a destructor. `stringAt` and `stringFindChar` are getters. This ADT's interface doesn't define any setter functions; what setters might you add?

You will want to make two allocations to store your string: one for the array of characters that makes up the string, and one for just the structure, which holds a reference to that array of characters. This gives you an assurance that the lifetime of the character array will be long enough. You could choose not to null-terminate this character array, but you must be extremely careful if you do.

Once you've written your ADT, you should write some white-box tests, and black-box tests.

> If you're confused about where to begin, try the warm-up exercises *A Complex ADT* and *Testing a Complex ADT*.

To run some simple automated tests:

```
$ 1511 autotest stringADT
```

To run Styl-o-matic:

```
$ 1511 stylomatic String.c
Looks good!
```

You'll get advice if you need to make changes to your code.

Submit your work with the *give* command, like so:

```
$ give cs1511 wk09_stringADT
```

Or, if you are working from home, upload the relevant file(s) to the `wk09_stringADT` activity on Give Online.