Learning Objectives: String Basics

- **Identify the three properties of strings**
- **Understand the meaning of mutability**
- **Determine if a string is present in another string and at what index**
- **Print a string from the start index to end index**
- **Utilize escape characters to add special characters to a string**

String Properties

String Length

We have already seen strings in the “Fundamentals”section. We are going to dig a little deeper with this data type. All strings have the following characteristics:

1. **Characters** - Strings are made up of characters between quotation marks (previously covered in the “Fundamentals” section).
2. **Length** - Each string has a length (total number of characters).
3. **Index** - Each character in a string has a position, called an index.

To calculate the length of a string, use the `length()` function. This function will return an integer that is the sum of all of the characters between the quotation marks.

```
string my_string = "Hello";
int len_string = my_string.length();

cout << len_string << endl;
```

challenge

What happens if you:

- Change `my_string` to `"Hello world!"`? $\longrightarrow$ 12
- Change `my_string` to `""`? $\longrightarrow$ 0
- Change `my_string` to `"-1"`? $\longrightarrow$ 2
- Change `my_string` to `"Привет"`

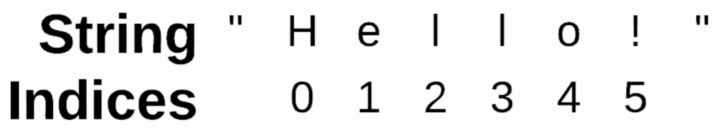
important

IMPORTANT

Although each character that you see on a typical keyword usually has a length of `1`, there are other foreign characters that do not follow this convention. For example, `Привет`, which stands for `Hello` in Russian, actually has a length of `12` instead of `6`.

String Index

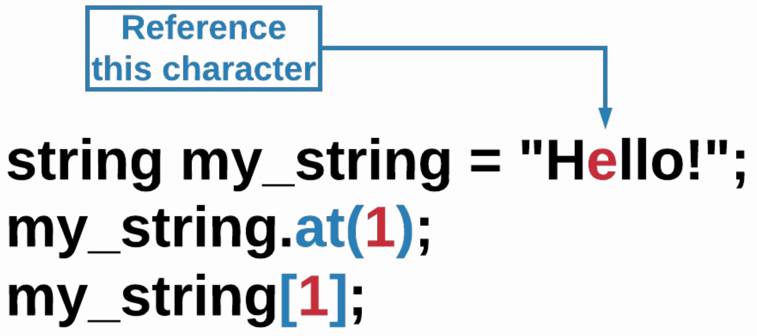
Previously in the **vectors** module, we learned that vectors and arrays have elements that reside in certain positions or **indices**. A string too has indices that correspond to the position where each of its character resides. Like vector and array indices, string indices also start at 0.



.guides/img/StringIndex

▼ **Strings & Quotation Marks**
Quotation marks are required to declare the value of a string. However, quotation marks are not a part of the string itself. That is why quotation marks are not counted with the `length()` function and why they do not have an index.

To reference a character, use the string name followed by the `at()` function. Within parentheses `()`, provide the index number of the character you want the system to return. Alternatively, you can also use brackets `[]` to enclose the index number.



```
string my_string = "Hello!";
my_string.at(1);
my_string[1];
```

.guides/img/StringAtIndex

```
string my_string = "Hello!";
char character = my_string.at(1);

cout << character << endl; → e
```

challenge

What happens if you:

- Change `char character` to `my_string.at(my_string.length())`? → error
- Change `char character` to `my_string.at(my_string.length()-1)`? → o
- Change `char character` to `my_string.at(5)`? → error
- Change `char character` to `my_string.at(-1)`? → error
- Change `char character` to `my_string[5]`? → nothing
- Change `char character` to `my_string[my_string.length()]`? → nothing

Note that you cannot use `my_string[my_string.length()]`. An index number is needed.

my_string[my_string.length()-1] works

Mutability

Mutability

You now know how to reference each character of a string. What do you think the code below will do?

```
string my_string = "House";
my_string.at(0) = "M";

cout << my_string << endl;
```

If you thought the code above would print `Mouse`, that would be a logical guess. However, you see an error. Unlike vectors and arrays where the characters can be manipulated, string literals are immutable. That means you cannot change the string literal itself. You can, however, manipulate a **particular** character within the string.

```
string my_string = "House";
my_string.at(0) = 'M';

cout << my_string << endl;
```

Can you spot the difference between the two code snippets mentioned above? The difference lies within the double quotes `""` and single quotes `''`. You cannot change the string literal, but you can change a character at a particular index. Thus, `my_string.at(0) = 'M'` changes the string to `Mouse` but `my_string.at(0) = "M"` does not.

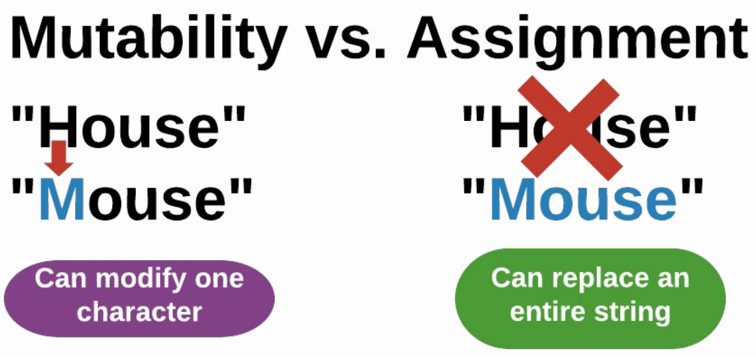
String Re-Assignment

In addition to character manipulation, you can also change the **entire** string itself by overwriting it with a new value(s).

```
string my_string = "House";
my_string = "Mouse";

cout << my_string << endl;
```

Strings in C++, like other data types, can be re-assigned. The previous examples on this page are about mutability. That is, changing just a part of a whole. The current example is about the assignment operator. Re-assignment replaces the entire value of a variable with a new value. In conclusion, you can either change one character at a time within a string, or change the entire string literal by reassigning it.



.guides/img/MutabilityAssignment

Find

The find() Function

The `find()` function tells you if a character or a string is present in another string, and if so, at what index it is at. `find()` returns an integer index if the character or string is present and `18446744073709551615` if not.

▼ What does `18446744073709551615` mean?

`18446744073709551615` is the largest integer value possible in C++. When `find()` is called and `18446744073709551615` is returned, it means the system has searched through all values and cannot locate the specified value. `18446744073709551615` is an unsigned value, but it is equivalent to `-1` as a signed value.

```
string my_string = "The brown dog jumps over the lazy fox.";

cout << my_string.find("dog") << endl;
```

→ 10

challenge

What happens if you:

- Change the `cout` statement to be `cout << my_string.find("cat");`? → 184...
- Change the `cout` statement to be `cout << my_string.find("Dog");`? → 184...
- Change the `cout` statement to be `cout << my_string.find(" ");`? → 3
- Change the `cout` statement to be `cout << my_string.find(my_string);`? → 0

If you want to start searching for a string or character starting at a particular index, you can specify the start index number after the specified string.

```
string my_string = "The brown dog jumps over the lazy fox.";

cout << my_string.find("he", 4) << endl; //start at index 4
```

→ 26

challenge

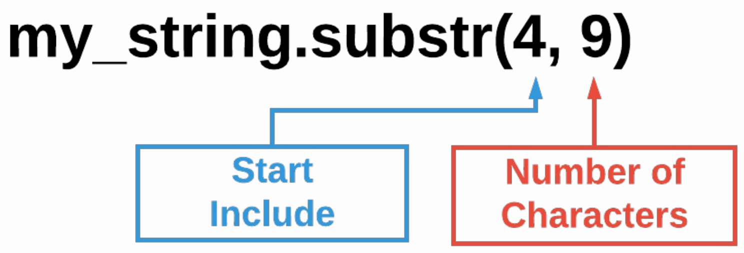
What happens if you:

- Change `my_string.find("he", 4)` to `my_string.find("he", 27)`? $\longrightarrow$ 184...
- Change `my_string.find("he", 4)` from the original code to `my_string.find("he")`? $\longrightarrow$ 1

Substr

The substr() Function

The `subst()` function returns a portion of the string. Within parentheses `()`, provide the index at which you want the string to start followed by a comma followed by the number of characters you want the string to include. **Note** that if you don't specify the *number of characters*, the system will start copying from the start index through to the end of the string. If you don't specify the *start index*, then the system will copy the entire string. The `subst()` function does not modify the original string. Instead, it returns a partial or entire copy of the original string.



.guides/img/StringSubstr

```
string my_string = "The brown dog jumps over the lazy fox.";
string my_slice = my_string.substr(4, 9);

cout << my_slice << endl;
```

challenge

What happens if you:

- Change `my_slice` to be `my_string.substr(1, 2)`? → he
- Change `my_slice` to be `my_string.substr()`? → entire string
- Change `my_slice` to be `my_string.substr(1, 1)`? → h
- Change `my_slice` to be `my_string.substr(2)`? → 2nd index til the end

Escape Characters

Escape Characters

An escape character is a character that has a different interpretation than what you see in a string. Escape characters always start with a backslash (`\`). The most common escape character is the newline character (`\n`) which causes C++ to print on the next line.

```
string my_string = "Hello\nworld";
cout << my_string << endl;
```

Escape Character	Description	Example
\\	Prints a backslash	`cout << "\\" << endl;`
\'	Prints a single quote	`cout << "\'" << endl;`
\"	Prints a double quote	`cout << "\"" << endl;`
\t	Prints a tab (spacing)	`cout << "Hello\tworld" << endl;`

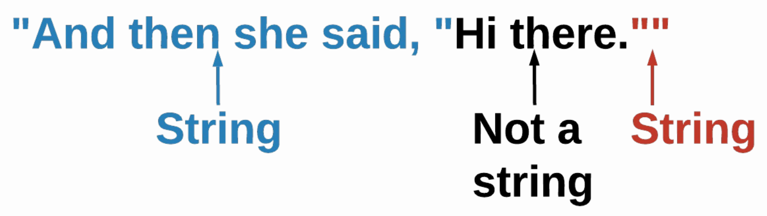
challenge

What happens if you:

- Use `\n\n` instead of `\n` in the original code?
- Replace `\n\n` in your current code with `\t`?
- Replace `\t` in your current code with `\"`?

Quotes Inside Quotes

Imagine that you have this small bit of dialog, `And then she said, "Hi there."` and want to store it as a string. Typing `"And then she said, "Hi there.""` would cause an error.



.guides/img/StringQuotes

When you use a `"` to start a string, C++ looks for the next `"` to end it. To avoid syntax errors, you can use a double quote to start your string, single quotes for the inner quote, and end the string with a double quote.

```
string my_string = "And then she said, 'Hi there.'";
cout << my_string << endl;
```

challenge

What happens if you:

- Replace single quotes (`'`) with double quotes (`"`) and double quotes(`"`) with single quotes (`'`) in the original code? → error
- Change all quotes to double quotes (`"`)? → error
- Use the escape character `\"` for the inner quotation marks that surround `Hi there.` (e.g. `"And then she said, \"Hi there.\""`)? → work as well