Programming Fundamentals

Muhammad Ateeq

[Updated: 08-04-2023]

Strings

We have just finished discussing character arrays. What are strings now?
In C++, a string is a sequence of characters that represents text. It is a data type that allows you to store and manipulate text data, such as names, addresses, and messages. You can declare a string variable by using the string class, which is part of the C++ standard library (what is a class? It is beyond the scope of this course and will be covered in object oriented programming that you study in next semester). Here is an example of how to declare and initialize a string variable:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
  string message = "Hello, world!";
  cout << message << endl;
  return 0;
}
```

In this example, the string class is used to declare a variable called message, which is initialized with the string value "Hello, world!", and is printed on the console.
You can also perform various operations on strings, such as concatenation, comparison, and substring extraction. The string class provides a number of member functions to perform these operations.

Isn't string the same as character array? If yes, why do we need strings? If not, what is the difference?
In C++, a string and a character array are similar in the sense that they both can represent a sequence of characters. However, there are some important differences between the two.

- A character array is a **fixed-size** array of characters that is allocated in memory. It is represented as a pointer to its first element, and you can access its individual elements using array notation or pointer arithmetic (as you saw in previous lesson). A character array has a fixed size, which means that you must specify its size when you declare it, and you cannot change its size later. Here's an example of how to declare and initialize a character array:

  ```
  char message[13] = "Hello, world!";
  ```

- A string, on the other hand, is an object that represents a **dynamic** sequence of characters. It is implemented as a class that provides a set of member functions for manipulating strings. A string can **grow or shrink dynamically** as you add or remove characters from it, and you do not need to specify its size when you declare it. Here's an example of how to declare and initialize a string:

  ```
  #include <string>
  string message = "Hello, world!";
  ```

One of the main advantages of using strings over character arrays is that strings are more flexible and easier to work with. For example, you can easily concatenate two strings using the + operator, or compare two strings using the == operator. With character arrays, you would need to use library functions like strcpy or strcmp to perform these operations. Additionally, string objects handle memory allocation and deallocation automatically, which makes them less prone to memory management errors.

In summary, while both strings and character arrays can represent sequences of characters, strings are more flexible and easier to work with than character arrays in many cases.

So, do strings use character arrays at backend and make a programmer's life easier at front?
Internally, a string in C++ may use a character array as a storage mechanism, but the implementation details are abstracted away from the programmer.

The string class in C++ is a wrapper around a dynamic array of characters that can grow or shrink in size as needed. The implementation of this dynamic array is managed by the string class, which handles memory allocation and deallocation, as well as other operations like concatenation, comparison, and substring extraction.

Ok, so strings facilitate programming! Let's begin then.
There can be a numbers of ways to begin learning about strings. It looks fair to start by saying that a string has:

- Characters: alphabets, numeric, special characters, spaces, etc.

  ```
  string welcome = “Hello”;
  ```

  The double quotes “” are not part of the string and are required to declare the strings.

- Length: total number of characters in a string. In the case of character arrays, the length was pre-specified, whereas strings are dynamic and size can change during execution.

  ```
  int len_welcome = welcome.length()
  cout << len_welcome; // or cout << welcome.length; // prints 5
  ```

- Index: each character has a position indicated by index starting at 0.

  ```
    0   1   2   3   4
  +---+---+---+---+---+
  | H | e | l | l | o |
  +---+---+---+---+---+
  ```

Notice, the null character ‘\0’ is not visible here. Actually, it is present but unlike character arrays it is handled automatically by strings and programmer is not required to take care of it.

Characters in an array are accessed using subscript. Is there any better way using strings?
In C++, you can access the individual characters of a string using two different methods:

- Subscript ([]) operator: This method allows you to access the characters of a string using zero-based indexing, just like an array. For example, to access the first character of a string my_string, you can use my_string[0].

- at() method: This method allows you to access the characters of a string using an index, just like the subscript operator. However, unlike the subscript operator, the at() method performs bounds checking to ensure that the index is valid. If the index is out of range, the at() method throws an std::out_of_range exception.

Here's an example that illustrates the difference between the two methods:

```
#include <iostream>
#include <string>
```

```
using namespace std;

int main() {
  string my_string = "Hello, world!";

  // Using the subscript operator
  cout << "Using subscript operator:" << endl;
  for (int i = 0; i < my_string.length(); i++) {
    cout << my_string[i];
  }
  cout << endl;

  // Using the at() method
  cout << "Using at() method:" << endl;
  try {
    for (int i = 0; i < my_string.length(); i++) {
      cout << my_string.at(i);
    }
  } catch (const out_of_range& e) {
    cout << "Error: " << e.what() << endl;
  }

  return 0;
}
```

In this example, the my_string variable is a string that contains the value "Hello, world!". The program uses both the subscript operator and the at() method to access the characters of the string and print them to the console.

When using the subscript operator, the program iterates over the string using a loop and accesses each character using the subscript operator (my_string[i]). When using the at() method, the program does the same thing, but it wraps the loop in a try-catch block to handle any out_of_range exceptions that may be thrown if the index is out of range. You do not need to worry about the use of try-catch block here. It is called exception handling and you will get to know more about it later.

In summary, the difference between the at() method and the subscript operator is that the at() method performs bounds checking to ensure that the index is valid, while the subscript operator does not. If you are certain that the index is within the bounds of the string, you can use the subscript operator for faster access to individual characters. However, if you are not sure about the index, or if you want to handle out-of-bounds errors gracefully, you should use the at() method instead.

Let's focus on changing the values in a string. This is called mutability.
Consider the following declaration:

```
string test_str = “Boat”;
```

Now let's compare the following two statements:

```
test_str.at(0) = “C”; // not allowed
test_str.at(0) = ‘C’; // allowed
```

The first statement, test_str.at(0) = "C";, is not allowed because "C" is a string literal and test_str.at(0) expects a single character. The at() function returns a reference to the character at the specified index, so it can be used to modify the contents of the string.

The second statement, test_str.at(0) = 'C';, is allowed because 'C' is a single character and matches the expected input for at(). This statement would replace the first character of the string "Boat" with the character 'C', resulting in the string "Coat".

This process of changing a single character is a string is called **mutability** and strings are mutable in this regard. Another possibility is to change the entire string and reassign it:

```
string test_str = “Boat”;
test_str = “Coat”;
```

In this case the entire string is replaced with new content as with other types of variables.

Is there a way to find or locate or search something within a string?
The find() method in C++ strings is used to search for a particular substring within a string. It returns the index (location) of the first occurrence of the substring within the string, or string::npos (a static constant defined in the string class) if the substring is not found. Don’t worry about the syntax of string::npos for now, you can just interpret it as “not found”.

This can be useful for tasks such as parsing data or checking for the presence of a certain character sequence in a string. The find() method can take up to three inputs, also called arguments:

- The substring to search for
- The starting index for the search (optional)
- The number of characters to search (optional)

Here's an example of using the find() method:

```
#include <iostream>
#include <string>

using namespace std;

int main() {
    string my_string = "hello world";

    // Find the index of the substring "world"
    size_t index = my_string.find("world");

    if (index != string::npos) {
        cout << "Substring found at index " << index << endl;
    } else {
        cout << "Substring not found" << endl;
    }

    return 0;
}
```

In this example, find() is called on the my_string object to search for the substring "world". The resulting index (if found) is stored in the index variable, and then checked for the value string::npos to determine if the substring was found.

Note that the find() method is case-sensitive by default, meaning that it will only find substrings that match the case of the search string.

Ok, so using find() method we can search for the presence of something within a string. What if we want to actually get that part of string and use it rather than just knowing the location/index?
The substr() method in C++ strings is used to extract a substring from a larger string. It takes two arguments:

- the starting index of the substring, and
- the length of the substring to extract.

Here's the syntax for using substr():

```
#include <iostream>
#include <string>

using namespace std;

int main() {
    string my_string = "Hello, World!";

    // Extract the substring "World"
    string substring = my_string.substr(7, 5);

    cout << "Substring: " << substring << endl;

    return 0;
}
```

In this example, substr() is called on the my_string object to extract the substring "World", which starts at index 7 (the 'W' in "World") and has a length of 5 characters. The resulting substring is stored in the substring variable and printed to the console.

The substr() method has many applications, such as:

- Extracting parts of a larger string for use in data processing or manipulation.
- Searching for specific patterns within a string and extracting them for further analysis.
- Manipulating and reformatting string data, such as changing the case of certain characters or replacing certain characters with others.
- Parsing data in specific formats, such as dates or times, and extracting the relevant components for further processing or display.