Strings in C and C++

CHAPTER 12

Strings in C are different from Strings in C++

There is NOT a string type in C

- We use **arrays of characters as strings in C**.
  - This has the same management problems as other arrays, fixed length, walking on memory, etc.
- In addition, **strings MUST end with a null terminator (\0)**, this creates more management issues.
  - You need an extra space in the array for the null terminator '\0'.
  - You must remember to put the null terminator at the end of the string in certain cases.
  - The C language tries to help you with special features for strings, which can be confusing.
- The string.h library has functions to help you work with character array style strings that are very useful such as concatenating strings, comparing strings, copying strings and finding their length.

There is a string type in C++, but it is a class

- C++ string is an object with data and functionality built into it.
- The string type in C++ works in a very intuitive manner, it is in the iostream library, so you do not have to include it.
- It automatically grows to fit the text.
- It manages it own memory
- It has functionality to concatenate, copy, search, and compare strings.

C++ is a Superset of C and this causes trouble

C++ is a superset of C, which means all the **C syntax works in C++** including C style strings which are character arrays.

The **C libraries are available in C++ for compatibility with a prefix of C**.

- C has a string library (string.h and it is available in C++ as <cstring>) that has functions that concatenate, compare and manipulate C style strings (null terminated arrays of characters). For strings**, these are not the C++ style libraries you should use**, but a copy of the C libraries for use in C++ for compatibility reasons. The C++ library <string> has functions to convert strings to other types which are very useful for data validation and a string iterator.
- The C math library (math.h) is available in C++ as <cmath>. The math.h library is implemented in C++ as the <cmath> library. Note: the C macros are implemented as functions in C++.
- The C time library (time.h) has the basic time functions and is available in C++ as <ctime>, C++ has <chrono> https://en.cppreference.com/w/cpp/chrono which adds a lot of functionality.
- See this for an overview of the C++ Libraries: http://www.cplusplus.com/reference/

Strings in C

What is a String Literal? (review)

Pictures of Strings

Examples

In C, **a string is an array of characters** with a **null terminator**

- All strings are **null terminated, which takes one character-sized space.**
- A string can be **declared as an array of characters**, which has type `char str1[] = "Happy"; //this is mutable.`
- A **string pointer**, declared as `char*`, can point at an array of characters (a string) or a string literal.
- The C compiler **treats the character array and string pointer at a string the same way** in most situations. This is like numeric arrays and pointers to numeric arrays.
- http://www.cplusplus.com/doc/tutorial/ntcs/

String Literals are immutable

- A **string literal** is zero or more characters enclosed in quotes.
- **String literals are not variables** and cannot be modified, they are immutable.
- The **compiler assigns read-only storage for string literals** and returns their memory address.
- The string literal has **type `const char*` and it is immutable.**
- You can **create a string pointer** and point it at a string literal by declaring it like this: `char* pStr = "Hello";`
- The pointer is **storing the address of the string literal.**
- You can output string literals directly.

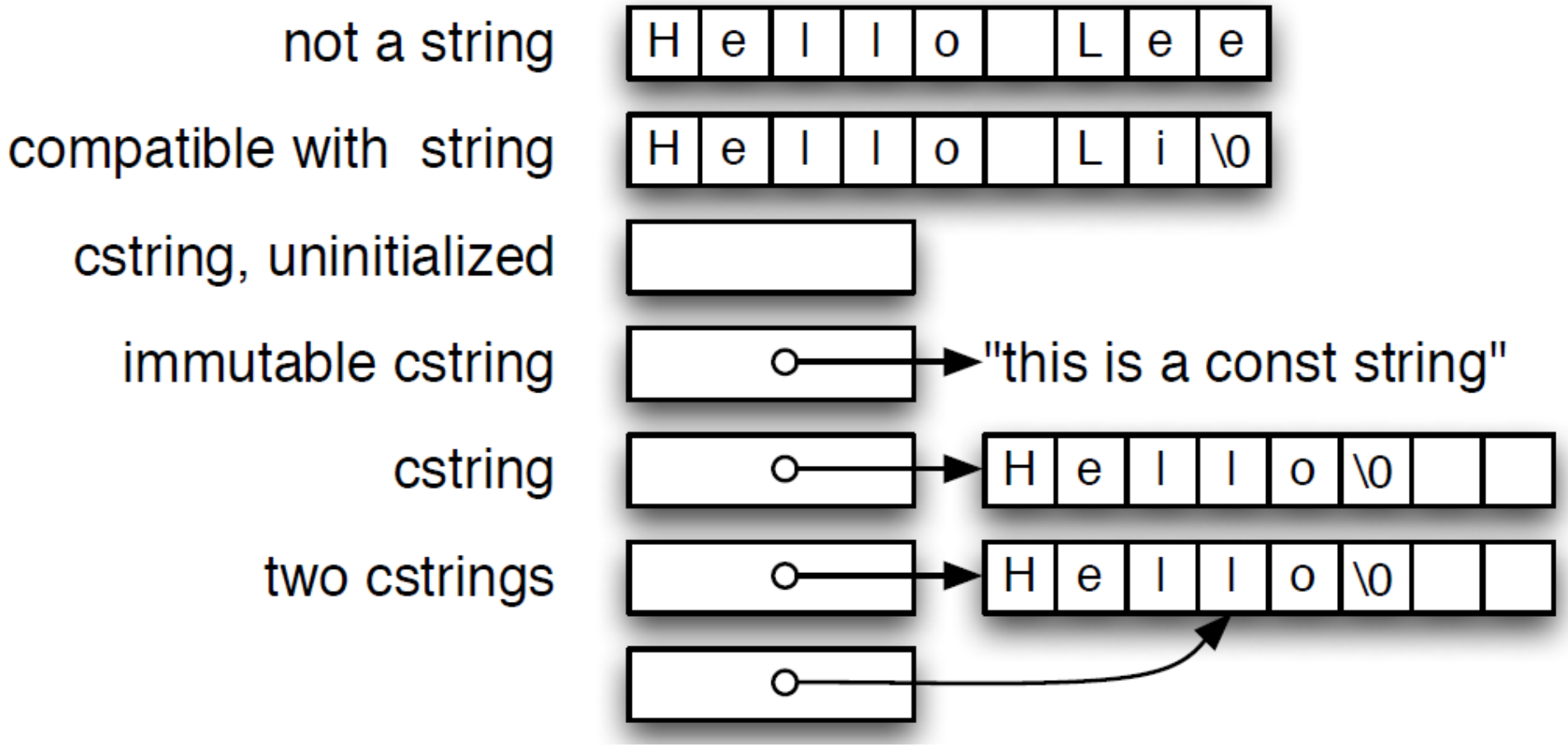
Pointer

String Literal

About String Variables

- If a string is stored in an array, it might or might not fill up the array.
- The **\0 character** marks the **end of the string**. After that, the contents of the array are unused.
- You can **modify the contents of the arra**y by using assignment and/or the string functions.
- When scanf() inputs a string, it will **ALWAYS end in \0** .
- However, if the string is longer than the array that stores it, it will overwrite the value of some other variable. This is called **walking on memory**.

Pictures of Strings



How do you store a string in C?

- A C string is an array of `char`, so it has two parts**: a pointer** and **an array of characters**.
- To **declare the array for a single string**, create an array of chars that is long enough. This array will hold up to 9 chars and a null terminator:

```
char ary[10];
```

- We declare **an array to be 1 longer than the longest possible contents** because there must be space at the end of the data for a **null terminator character, `'\0'`.**
- You can also create a pointer to point at your string **using the type `char*`**.
- Remember: a string can occupy many bytes of storage, and **a pointer is 8 bytes (64-bit machines)**. So, a **char* does not store a string**. It is the **pointer to the string and contains its address!**

How do you **declare** a **mutable string** in C?

- You can **declare an array to store a string WITHOUT initialization**, like ary above. For this kind of declaration, you must give the **maximum string length +1 in the square brackets**.

```
char str1[16]; //can hold up to 15 char + '\0'
```

  You can initialize a string in the declaration

```
char greeting[] = "Hello"; //will have 6 elements
```

- You can also give BOTH the array length AND an initializer

```
char name[16] = "M. Jones";
                //9 char including '\0' are used
```

- When you choose the array length, consider the longest thing you intend to store in it. That is often longer than the initial value for the string.

How do you **use** a string in C?

- A C **string pointer variable (`char*`)** can point at one literal string or an array of characters, then later, be changed to point at another.
- A pointer **can point at the any character** in the char array or literal string.
- To print a string, use `printf( "%s", myStringName );`
- To accept input of a string that **does not have internal spaces**, use

  `scanf( "%ns", myArrayName );`

  - **The number n is the length-1** of the array you will be reading the string into.

Input that **Includes Whitespace** in C

- Reading a string with embedded whitespace is more complex than reading one word in both C and C++.
- Suppose `name` is an array of 10 chars; to read everything up to the next newline:

```
scanf( "%9[^\n]", name ); //[^\n] for a newline delimiter
```

  - To read everything up to the next delimiter, which is a comma here:

```
scanf( "%9[^,]", name );//[^,] for a comma delimiter
```

- **The delimiter**, a newline or something else, **is left in the input stream and will be read in by the next call on scanf()**. To remove it from the stream and discard it on this read, add it to the format like this:

```
scanf( "%9[^,],", name );
//automatically discard the delimiter
```

C Examples

```
char* fname = "Beatrice"; // Point at string literal.
char letters[20] = "Franklin";
char* lname = letters; // Point at letters[0].
char* fun = &fname[5]; // Point at fname[5].
printf("My name was %s %c. %s\n", fname, 'K', lname);
lname = “Jones";
printf("Now my name is %s %c. %s ", fname, 'K',lname);
printf("\nI like to skate on %s.\n", fun);
```

Output:

My name was Beatrice K. Franklin

Now my name is Beatrice K. Jones

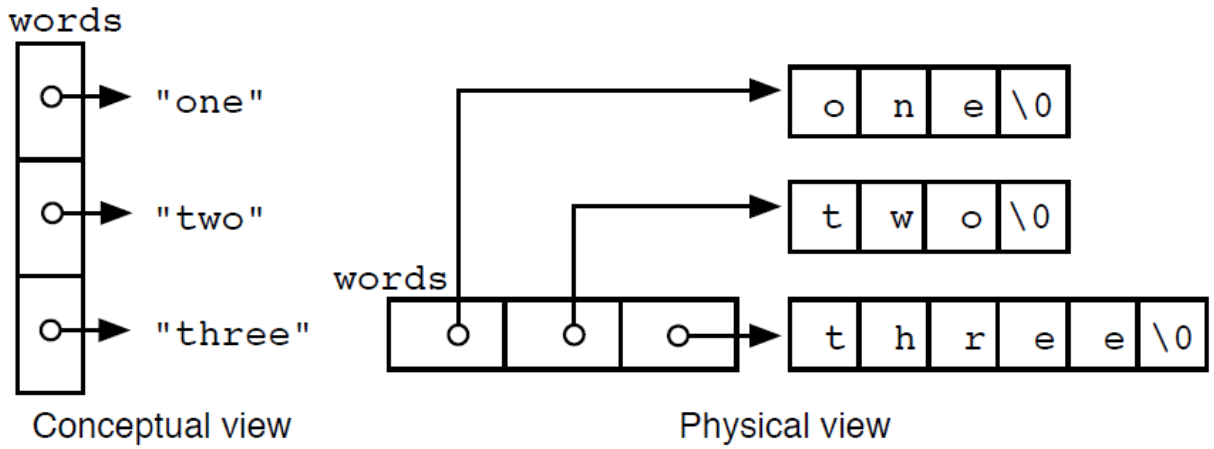
I like to skate on ice.

An Array of Strings

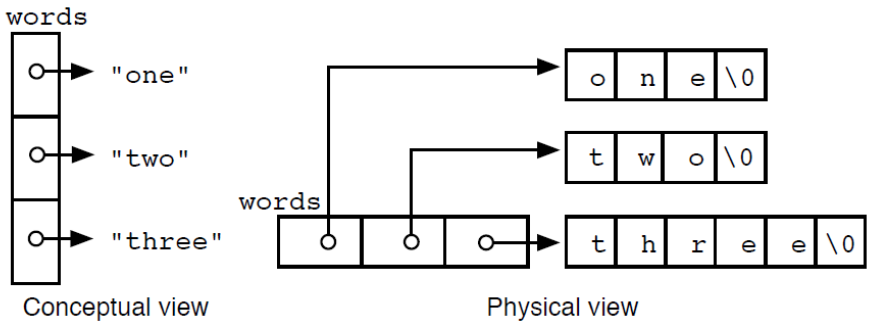
Arrays of strings are useful in many application. This data structure (often called a ragged array) has two parts:

- A backbone: an array of pointers.
- Several attachments of type char*, possibly const char*.

This is a 2-dimensional data structure: you can subscript the backbone, then you can subscript the strings attached to it.



Example



To declare the array of strings:

```
char* words[3] = {"one", "two", "three"};
```

- Each element/slot in the **array is a pointer that stores address** of a string (these can be string literals or character arrays)
- The array of characters that makes up the string is at the address the slot of the array points to/contains.
  - If it is created as a separate array, it is mutable.
  - If it is a string literal, it is immutable.

Basic String Operations
Copying a String
String Comparison
Searching a String
An Array of Strings
A 2-D array of chars

THE C STRING LIBRARY

Programming Language Standard

- **All modern languages must** implement the functionality of the string library in C.
- These functions do not have to have the same name or look the same, but **the functional ability must exist**.
- **By knowing the C String Library functionality**, you have **insight into other languages,** because you know what features to expect.

C String Operations to **Change** a String

```
char word[10] = "Hi"; // character array
char* str = word; //a pointer
```

There are two ways to change a string:

- Assignment so the pointer **points at a different string (address).**

```
str = "Harmony"; //Make str point at a different word
```

- **change one or more of the chars** in the string.

```
word[1] = 'a'; //Change “Hi” to “Ha”
```

C String **Pitfall of String Comparison**

```
char word[10] = "Hi";
char* str = word;
if (str == "Hi") {}
```

- This is an **incorrect** comparison, it **compares the memory addresses**
- It will only resolve as false in C because **the string literal "Hi" is at a different memory location** than word[]
- This is **checking if `str` points at the string literal's address**. It **does not compare the contents** of the two strings.

C String Basic Functions

- There are many functions in the C string library. These are a few that are very useful, you should look at all of them.
- These functions operate on the series of characters:

```
strcmp(word, "Joy"); // Compares the letters in the
array word to the string literal “Joy”

strlen(word); // Returns the # of letters before the \0.

strcpy(word, "Joy"); // Copies the letters J-o-y into
the array called word.
```

Destination, copy to here

Source, copy from here

Working with C Strings

```
size_t strlen( const char* s ); //prototype
```

- **Returns the number of characters in the string s, excluding the null character on the end.**
- `size_t is a common return type, it is an` **`unsigned short int`**
- `sizeof() // returns the number of bytes in the address/array`

General notes about strings:

- Use subscript with = to modify individual chars of a string (stored in an array of char).
- Subscript can be used for an array of chars, and for a pointer to an array of chars.
- Subscripts with a pointer: the subscripts are relative to the slot the pointer addresses.
- If two or more string pointers point into the same array of chars, they can both/all be used to modify the string stored there.

More Useful C String Functions

- `char* strcpy( char* dest, const char* src );`
  - Copies the string src into the array dest. We assume that dest has space for the string.
- `char* strncpy(char* to,const char* src, size_t n)`
  - Copies exactly n characters from src into to. If fewer than n characters are in src, null characters are appended until exactly n have been written.
- `char* strcat( char* dest, const char* src );`
  - Appends the string src to the end of the string dest, overwriting its null terminator. It is a serious error is dest does not have space for the combined string.
- `char* strncat(char* to, const char* src, size_t n)`
  - Same as strcat() except that it stops after copying n characters, then writes a null terminator.

C String Comparisons.

```
char* s1; char* s2; //creates two pointers
```

- `s1 == s2` asks if the pointers **store the same memory address**.
- To compare the chars in the strings that s1 and s2 point at, use strcmp().

```
int strcmp( const char* p, const char* q );
```

- Compares string p to string q and returns a **negative** value if p is **lexicographically** less than q, **0 if they are equal**, or a **positive** value if p is greater than q.

```
int strncmp(const char* p,const char* q,size_t n)
```

- Same as strcmp() but returns after **comparing at most n characters. It will return sooner if a null character happens sooner.**

Searching a String in C

```
char* strchr( const char* s, int ch );
```

- Searches the string s for the **first (leftmost) occurrence of the character ch.** Returns a pointer to that occurrence if it exists; otherwise returns NULL.

```
char* strrchr( const char* s, int ch );
```

- Searches the string s for the **last (rightmost) occurrence of the character ch**. Returns a pointer to that occurrence if it exists; otherwise returns NULL.

```
char* strstr( const char* s, const char* sub );
```

- Searches the string s for the **first (leftmost) occurrence** of the substring sub. Returns a pointer to the first character of that occurrence if it exists; otherwise returns NULL.

Packaging an Array Data Structure
Why Package?
The DataPack Structure

VECTOR CLASS

Make your Life Easier

A basic organizational rule that applies throughout life: If two things are always used together, keep them together in one place.

- In C, an array has only one part (a series of variables of the same base type).
- But when you use a C array you also need two numbers:
  - the number of slots that are in the array (allocation length)
  - the number of slots that have data in them (number filled)
- It makes sense to bundle the array and the two numbers needed to manage it into a single structure that can be passed with one parameter to a function.

In modern OO languages, a bundle of parts that belong together is called a class. A class includes a set of related data parts and the functions that operate on them.

The vector template class.

vector is the name of an STL template class in C++.

- It is an array-like class that can "grow" to hold as much data as needed.
- To implement this capability, a vector needs at least these three data members:
  - **A pointer** to a **dynamically-allocated array** of objects of the base type (BT).
  - **max**, the number of array slots that are **currently allocated**.
  - **size**, the number of BT values **currently stored** in the array.
- When n equals max, the array is full.
- The next insertion will cause the array to be reallocated at double the length.
- The data will be copied into the new array and the old one will be freed.

Vector functions

vector provides these functions and many others:

- **size()**: the number of data values currently stored in the vector.
- subscript: **Use []** as you would with an ordinary array
- **push_back()**: Store a new value at the end of the vector. This is the normal way to store data in a vector.
- **data()**: a pointer to the first slot of the data array that is inside the vector.
- **begin() and end():** These return iterators (like pointers) to the first data item in the vector and the address just past the end of the vector. Use them with sort.
- **sort(iterator start, iterator end):** Sorts the data in the vector between start and end.

C++ template classes.

A template in C++ is a way to define a **generic container class** that can store **any type of data**.

- To use a template class, write its name followed by angle brackets.
- The type of data stored is given in the angle brackets.
- The type name in the angle brackets can be any primitive type or previously-defined class type.
- Sample declarations: vector<int> or vector<BT>
- Writing this declaration instantiates the template class and produces a normal class.
- The name of the resulting class is vector<BT>

Using a vector in C++

- Assume the istream named fin is properly open.
- Read numbers from a file into the vector named vec, starting with the first unfilled slot.

```
typedef double BT;
vector<BT> vec;
for(;;) {
 fin >> data;
 vec.push_back( data ); //C++ 11 required
 if( fin.eof()) break;
}
```

- The vector class keeps track of the number of items that have been pushed into it.
- At the end of the operation, the value of n in the vector will correspond to the number of items in the array.

Strings in C++
Syntax for Strings in C++
Examples

Why C++ is the better choice for working with strings!

What is a string in C++?

A C++ string is more complex than a C string and easier to use.

- It is **an object with multiple parts**, including an **array for the string data**, a **counter for the number of data items in the array,** and the **current allocated capacity of the array**. Get them with size() for the number of data items and capacity() for the number of allocated spaces.
- A C++ string can **hold as much data as you put into it**.
- When it gets full, **it grows automatically by reallocating the data space**.
- **No memory management is necessary** because the string **manages its own memory.**
- The **type string is defined by STL** (the Standard Template Library) and is **derived** from the STL **template class vector**.

How do you declare a string in C++?

Your options:

```
string st;   //declares an empty string
string st1( "Mary" );   //declares and initializes a string
string st2( st1 );    //Makes a copy of st1
string st3 = "Jake";  // Copy the literal into st3
```

. . . and there are other ways to construct strings that are less often needed.

A Few C++ String Functions

String processing in C++ ***is less complex*** than it was in C.

Suppose we have two **C++** strings names str1 and str2:

- Use =, the assignment operator, to **copy the contents** of one string into another.

```
string ss = st1;
```

- Use [ ] to access one letter in the string

```
char ch = name[0];
```

See the documentation for a complete list of functions and how they work:
https://cplusplus.com/reference/string/string/

More C++ String Functions

Suppose we have two **C++** strings named str1 and str2:

```
string st1 = "Happy";
string st2 = "Day";
```

- Find the **length** of a string:

```
int len = str1.size();
```

- Get the **first char in a string:**

```
char c2 = str1.front();
```

- Set the last char in a string:

```
str1.back() = 'y';
```

More C++ String Functions

- **Concatenate** two strings:

```
cout << str1 +" "+ str2;
```

- **Extract the C-string** from the C++ string (for future reference):

```
char* cstr = str2.c_str();
```

- Use ==, the comparison operator, to **compare two strings** for equality.

```
if (str1 == str2)
    cout << "This is not bad." << endl;
```

- The comparison operator also works to **compare a string to a string literal**.

```
if (str2 == "Ann")
    cout << "Also doable.";
```

More C++ String Functions

- **Search a string for first occurrence** of a character:

```
int idx = str1.find_first_of('x');
```

- The **search can start at any position** in the string:

```
//This searches for ALL occurrences of the character 'x'.
    for( idx = 0; idx <str1.size(); ++idx){
                                        //↑ starts at next char
            idx = str1.find_first_of('x',idx)
            cout<< idx;
    }
```

- **Search a string for last occurrence** of a character:

```
int idx = str1.find_last_of('x');
```

How do you use a string in C++?

- You can input into a string variable?:

```
cin >> st;   // This will read one word (not a char, not a line) into the string
```

- To input a string that **has internal spaces**, use getline().

```
getline( instr, mystring ); // Read to \n
getline( instr, mystring, 'X' ); // Read to 1st X.
```

Delimiter

- To print any string, use

```
cout << myString ;
```

- A parameter (in a function declaration) can be type string.
  - In that case, the argument (function call) should also be a string, and the argument value is **copied** into the parameter variable.

A Few C++ String Functions

Suppose we have two C++ strings names str1 and str2:

- Find the length of a string:

```
int len = str1.size();
```

- Search a string for a character:

```
int idx = str1.find_first_of("x");
```

C++ Examples

```
string name ("Beatrice");
const char* lname = "Franklin"; //
string fun = name.substr(5, 3); // Get a substring.
cout <<"My name was " <<name <<" W. " <<lname  <<endl;
lname = "Jones";
cout << "Now my name is " <<name <<" W. " << lname << endl;
cout <<"I like to skate on " <<fun << endl;
```

Output:

My name was Beatrice W. Franklin

Now my name is Beatrice W. Jones

I like to skate on ice

Replit.com Practice Assignment in Pairs

- Choose a partner and sit together. One person must create the team by clicking on the Strings Practice assignment. The other person must ***wait for their partner to start the team and then join that team***.
- Each team will **work with C++ strings** to do a task using their assigned **method from the string class**.
- Write a short program together using your assigned string function to demonstrate what it does.
- You can both write and edit the code in the assignment repl.
- You have 15 minutes and then we will share the results today or at the beginning of the next class.
- Each team will present their repl to the class.

Advanced Strings: Extracting the array of char from a string in C++

A C++ string object contains an array of char among other things.

If you need just the array of char as an argument to a C function call, you can extract it.

- If you have a C++ string, **you can extract the C string from it for use with C functions** by calling c_str() with the string object with **const char*** as its type:

```
char cstr[str2.size()+1]; //create c style var
strcpy(cstr,str2.c_str()); //copy data to it
```

That’s it for now.