```
//===========================
// Section 10 Characters and strings
//===========================

// Character functions
#include <cctype>
Functions for testing characters
Functions for converting character case

functionName(char); // all the functions expect a single character

Function            Properties
isalpha(c);         True if c is a letter
isalnum(c);         True if c is a letter or a digit
isdigit(c);         True if c is a digit
islower(c);         True if c is a lowercase letter
isprint(c);         True if c is a printable character
ispunct(c);         True if c is a punctuation character
isupper(c);         True if c is an uppercase letter
isspace(c);         True if c is a whitespace

// Conversion methods
tolower(c)
toupper(c)
- If they can't convert they will just spit back out what was passed in

// C style strings
Sequence of characters
- Stored contiguously in memory
- implemented as an array of characters
- terminated by a null character that is the equivelent of 0
- Referred to as zero or null terminated strings

String literal
- Sequence of characters in double quotes - like "Josie"
- constant
- terminated with a null character

// Example of how it looks in memory
c++ is fun\0
------------

// Declaring c-style string variables
char MyName[]{"Josie"}; // Compiler will allocate 6 characters - one for the null
- BE CAREFUL - if you reassign thei variable to a larger string you will go out of bounds

// Declaring without initializing
char myName[8];

We can't use assignment with these
myName = "Josie"; // This is invalid and will not work

-Instead, use strcopy();
strcpy(myName, "Josie"); // This will add null characters

// Other functions for cstyle strings
#include <cstring>
Copying, Concatentation, Comparison, Searching, etc.

strcpy(); // Copy
strcat(); // Concatenate
strcmp(); // Compare strings
```

```
// General purpose to convert C-style strings to other types
#include <cstdlib>

Can convert to integer, float, long, etc

strlen() actually returns a type size_t
This is an unsigned type that is dynamically configured when we use it
It can be different on different systems but it's always unsigned

// getline() function
cin.getline(fullName, 50); // Get the line and put into the variable, stopping at the numebr of characters

// C-Style string examples

#include <iostream>
#include <vector>
#include <string>

int main(){
    // Always initialize variables
    char firstName[20]{};
    char lastName[20]{};
    char fullName[50]{};
    char temp[50]{};

    // These uninitialized will display garbage
    //char firstName[20];
    //char lastName[20];
    //char fullName[50];
    //char temp[50];

    std::cout << firstName; // Will display garbage

    std::cout << "Enter first name: ";
    std::cin >> firstName;

    std::cout << "Enter last name:";
    std::cin >> lastName;
    std::cout << "--------------------" << std::endl;

    std::cout << "Hello, " << firstName << ". Your first name has: " << strlen(firstName) << " characters" << std::endl;
    std::cout << "Your last name is: " << firstName << ". Your last name has: " << strlen(lastName) << " characters" << std::endl;

    // Copies from the second argument to the first one
    // Note that the compiler on my system threw errors and made me use strcat_s
    strcpy_s(fullName, firstName);  // Copy first name to full name
    strcat_s(fullName, " ");            // Copy a space to full name
    strcat_s(fullName, lastName);       // Copy last name to full name
    std::cout << "Your full name is: " << fullName << std::endl;

    // use getline() to get a full line of input
    std::cout << "Enter your full name: ";
    std::cin.getline(fullName, 50);
    std::cout << "Your full name from getline is: " << fullName << std::endl;

    std::cout << "---------------------" << std::endl;
    strcpy_s(temp, fullName);
    // If we get back a 0 that means they are the same
    // If the string lexically comes before it will return -1
    // If the string lexically comes after it will return 1
    if (strcmp(temp, fullName) == 0) {
        std::cout << temp << " and " << fullName << " are the same " << std::endl;
    } else {
        std::cout << temp << " and " << fullName << " are different " << std::endl;
    }
    std::cout << "----------------" << std::endl;

    // Change the name to all uppercase letters
    for (size_t i{ 0 }; i < strlen(fullName); i++) {
        if (isalpha(fullName[i])) {
            fullName[i] = toupper(fullName[i]);
        }
    }

    // Check it again
    if (strcmp(temp, fullName) == 0) {
        std::cout << temp << " and " << fullName << " are the same " << std::endl;
    } else {
        std::cout << temp << " and " << fullName << " are different " << std::endl;
    }
    std::cout << "-----------------" << std::endl;

    return 0;
}

// C++ style strings
std::string is a class in the Standard Template Library
#include <string>
std namespace
contiguous in memory
dynamically sized
Works with input and output streams
lots of useful member functions
can use the familiar operators
can be converted to Cstyle strings if needed
safer - does bounds checking and etc

// Initializing strings
#include <string>

std::string s1;                 //Empty
std::string s2{"Josie"};        // Josie
std::string s3{s2};             // Josie
// Number of characters to use
std::string s4{"Josie", 3};     // Jos
// Index start and how many to copy
std::string s5{s3, 0, 2};       // Jo
// Constructor
std::string s6(5, "J");         // JJJJJJ

// Assignment
std::string1;
s1 = "C++ is amazing";
std::string s2;
s2 = s1;

// Concatenation
std::string part1{"C++"};
std::string part2{"Is a powerful"};

std::string sentence;
sentence = part1 + " " + part2  + " language";
// This will print C++ is a powerful language

sentence = "C++" + " is powerful"; // illegal - cannot put to C style string literals
together

// Accessing characters with [] and .at()
sentence[0];
sentence.at(0);




```

```
// Can also iterate over
for(char c: sentence){
    std::cout << c << std::endl;
}

// If you use an integer type it will convert the chars to ints in the ascii table
for (int c: sentence){
    std::cout << c << std::endl;
}

// Comparing strings
==, !=, >, >=, <, <=

The objects are compared character by character lexically

Can compare:
two std::string objects
std::string object and C-style string literal
std::String object and C-Style string variable

CANNOT be used on two C-style literals

// Comparing examples
std:string s1{"Apple"}
std:string s2{"Banana"}
std:string s3{"Kiwi"}
std:string s4{"apple"}
std:string s5{s1}

s1 == s5        // True
s1 == s2        // False
s1 != s2        // True
s1 < s2         // True
s2 > s1         // True
s4 < s5         // False
s1 == "Apple"   // True

// Extracting substrings
object.substr(start_index, length);

std::string s1{"This is a string"};

std::cout << sq.substring(0,4);     //This
std::cout << sq.substring(5,2);     //is
std::cout << sq.substring(10,5);    //String

// Searching strings
object.find(searchString);
std::string s1{"This is a string"};
std::cout << s1.find("This");           // Finds it at 0
std::cout << s1.find("is", 4);          // Starts looking at position 4
std::cout << s1.find("XX");             // string::npos - no position, not found

object.rfind(); // Same as above but in reverse

// Removing characters - erase() and clear()
Removes a substring of characters from a std::string

object.erase(start_index, length);

string s1{"This is a test"};

std::cout << s1.erase(0,5);     // Deletes up to the is
std::cout << s1.clear();        // Empty string

// count the string length
.length();
```

```
// Compound concatenation operator
overloaded +=

s1 += " of C++"; // will return "This is a test of C++"

// Input and getline();
When you read input with cin, it only takes up to the first whitespace
std::string s1;
std::cin >> s1; // Hello There
                // Only accepts to the first space
std::cout << s1 << std::endl; // Will only print Hello


// getline(inputstream, variableToStoreIn);

getline(cin, s1);   // Will read the entire line until \n
std::cout << s1 << std::endl;   // Will dislay hello there

// getline(inputStream, variableToStoreIn, delimiter);
getline(cin, s1, 'x');  will stop taking input at the delimiter

// C++ string examples
#include <iostream>
#include <vector>
#include <string>
#include <iomanip>

int main(){

    std::string s0; // Will still be automatically initialized
    std::string s1{ "Apple" };
    std::string s2{ "Banana" };
    std::string s3{ "Kiwi" };
    std::string s4{ "apple" };
    std::string s5{ s1 };
    std::string s6{ s1, 0, 3 }; // first three characters of Apple
    // Constructor style initialization
    std::string s7( 10, 'X' ); // 10 X characters

    //std::cout << s0 << std::endl;
    //std::cout << s0.length() << std:: endl;

    // Initilialization
    std::cout << "\nInitialization" << "\n---------------------------" << std::endl;
    std::cout << "s0 is initialized to: " << s0 << std::endl;
    std::cout << "s1 is initialized to: " << s1 << std::endl;
    std::cout << "s2 is initialized to: " << s2 << std::endl;
    std::cout << "s3 is initialized to: " << s3 << std::endl;
    std::cout << "s4 is initialized to: " << s4 << std::endl;
    std::cout << "s5 is initialized to: " << s5 << std::endl;
    std::cout << "s6 is initialized to: " << s6 << std::endl;
    std::cout << "s7 is initialized to: " << s7 << std::endl;

    // Comparison
    std::cout << "\nComparison" << "\n---------------------------" << std::endl;
    std::cout << std::boolalpha;
    std::cout << s1 << " == " << s5 << ": " << (s1 == s5) << std::endl;
    std::cout << s1 << " == " << s2 << ": " << (s1 == s2) << std::endl;
    std::cout << s1 << " != " << s2 << ": " << (s1 != s2) << std::endl;
    std::cout << s1 << " < " << s2 << ": " << (s1 < s2) << std::endl;
    std::cout << s2 << " > " << s1 << ": " << (s2 > s1) << std::endl;
    // Uppercase characters come before the lowercase ones in ascii table
    std::cout << s4 << " < " << s5 << ": " << (s4 < s5) << std::endl;
    std::cout << s1 << " == " << "Apple" << ": " << (s1 == "Apple") << std::endl;

    // Assignment
    std::cout << "\nAssignment" << "\n---------------------------" << std::endl;
    s1 = "Watermelon";
    std::cout << "s1 is now: " << s1 << std::endl;
```

```cpp
    s2 = s1;
    std::cout << "s2 is now: " << s2 << std::endl;

    s3 = "Yaya";
    std::cout << "s3 is now: " << s3 << std::endl;

    s3[0] = 'W';
    std::cout << "s3 is now: " << s3 << std::endl; // Changes to Wawa
    s3.at(0) = 'Y';
    std::cout << "s3 is now: " << s3 << std::endl; // Back to Yaya

    // Concatenation
    std::cout << "\nConcatenation" << "\n----------------------------" << std::endl;
    s3 = "Watermelon";
    s3 = s5 + " and " + s2 + " juice"; // Apple and banana juice
    std::cout << "s3 is now: " << s3 << std::endl;
    //s3 = "nice " + " cold" + s5 + "juice"; // compiler error

    // For loop
    std::cout << "\nLooping" << "\n----------------------------" << std::endl;
    s1 = "Apple";
    for (size_t i{ 0 }; i < s1.length(); i++) {
        std::cout << s1.at(i) << std::endl; // or index style s1[i]
    }

    // Range-based for loop
    for (auto letter : s1) {
        std::cout << letter << std::endl;
    }

    // Substring
    std::cout << "\nSubstring" << "\n----------------------------" << std::endl;
    s1 = "This is a test";
    std::cout << s1.substr(0, 4) << std::endl;  // This
    std::cout << s1.substr(5, 2) << std::endl;  // is
    std::cout << s1.substr(10, 4) << std::endl; // test

    // Erase
    std::cout << "\nErase" << "\n----------------------------" << std::endl;
    s1 = "This is a test";
    s1.erase(0, 5); // Erase the first five characters
    std::cout << "s1 is now: " << s1 << std::endl;
    s1.erase();
    std::cout << "s1 is now: " << s1 << std::endl;

    // getline()
    std::cout << "\ngetline()" << "\n----------------------------" << std::endl;
    std::string fullName{};
    std::cout << "Enter your full name: ";
    getline(std::cin, fullName);
    std::cout << "Your full name is: " << fullName << std::endl;

    // find()
    std::cout << "\nfind()" << "\n----------------------------" << std::endl;
    s1 = "The secret word is yaya";
    std::string word{};
    std::cout << "Enter the word to find: ";
    std::cin >> word;
    size_t position = s1.find(word);
    if (position != std::string::npos) {
        std::cout << "Found " << word << " at position: " << position << std::endl;
    } else {
        std::cout << "Sorry, that word was not found." << std::endl;
    }
    return 0;
}
```

```cpp
//===========================
// Section 10 Characters and strings Challenge
//===========================
#include <iostream>
#include <vector>
#include <string>

int main(){
    // Parse each letter, find the position in the main ciper

    // Substitute the corresponding letter in the cipher

    std::string mainAlphabet{"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"};
    std::string cipherLetters{"zyxwvutsrqponmlkjihgfedcbaZYXWVUTSRQPONMLKJIHGFEDCBA"};
    std::string phraseFromUser{"This is my test phrase"};
    std::string swapString{};
    //std::cout << "Please enter a phrase for the substitution:";
    //getline(std::cin, phraseFromUser);

    // Note when using size_t or unsigned int this threw an error.
    // i >= 0 is always true for an unsigned value.
    // Also got a narrowing conversion error with initialization syntax. Static cast it
    since I knew the values
    //for (signed int i{ static_cast<signed int>(mainAlphabet.length() - 1) }; i >= 0;
    --i) {
    //  std::cout << mainAlphabet.at(i) << " at " << i << std::endl;
    //}

    for (size_t i{ 0 }; i < phraseFromUser.length(); i++) {
        // Get the character we need to substitute
        char letterToSubstitute{ phraseFromUser[i] };
        // Find what position that character is in the real alphabet
        int position = mainAlphabet.find(letterToSubstitute);
        std::cout << position << std::endl;
        // Check that it exists in the real alphabet
        if (position == std::string::npos) {
            // and go back to the start of the loop if it does not
            continue;
        } else {
            phraseFromUser.at(i) = cipherLetters.at(position);
        }
    }

    std::cout << phraseFromUser;

    for (size_t i{ 0 }; i < phraseFromUser.length(); i++) {
        // Get the character we need to substitute
        char letterToSubstitute{ phraseFromUser[i] };
        // Find what position that character is in the real alphabet
        int position = cipherLetters.find(letterToSubstitute);
        std::cout << position << std::endl;
        // Check that it exists in the real alphabet
        if (position == std::string::npos) {
            // and go back to the start of the loop if it does not
            continue;
        } else {
            phraseFromUser.at(i) = mainAlphabet.at(position);
        }
    }
    std::cout << phraseFromUser;
    return 0;
}
```

```cpp
//===========================
// Section 10 Characters and strings Franks solution
//===========================

#include <iostream>
#include <vector>
#include <string>

int main(){

    std::string mainAlphabet{ "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz" };
    std::string cipherLetters{ "zyxwvutsrqponmlkjihgfedcbaZYXWVUTSRQPONMLKJIHGFEDCBA" };
    std::string secretMessage{};
    std::cout << "Enter the message: ";
    getline(std::cin, secretMessage);
    std::string encryptedMessage{};
    std::cout << "\nEncrypting Message..." << std::endl;

    for (char c : secretMessage) {
        size_t position = mainAlphabet.find(c);
        if (position != std::string::npos) {
            char newCharacter{ cipherLetters.at(position) };
            encryptedMessage += newCharacter;
        } else {
            encryptedMessage += c;
        }
    }

    std::cout << encryptedMessage << std::endl;

    std::string decryptedMessage{};
    for (char c : encryptedMessage) {
        size_t position = cipherLetters.find(c);
        if (position != std::string::npos) {
            char newCharacter{ mainAlphabet.at(position) };
            decryptedMessage += newCharacter;
        } else {
            decryptedMessage += c;
        }
    }
    std::cout << decryptedMessage << std::endl;

    return 0;
}
```