

```

1 //=====
2 // Section 10 Characters and strings
3 //=====
4
5 // Character functions
6 #include <cctype>
7 Functions for testing characters
8 Functions for converting character case
9
10 functionName(char); // all the functions expect a single character
11
12 Function          Properties
13 isalpha(c);       True if c is a letter
14 isalnum(c);       True if c is a letter or a digit
15 isdigit(c);       True if c is a digit
16 islower(c);       True if c is a lowercase letter
17 isprint(c);       True if c is a printable character
18 ispunct(c);       True if c is a punctuation character
19 isupper(c);       True if c is an uppercase letter
20 isspace(c);       True if c is a whitespace
21
22 // Conversion methods
23 tolower(c)
24 toupper(c)
25 - If they can't convert they will just spit back out what was passed in
26
27 // C style strings
28 Sequence of characters
29 - Stored contiguously in memory
30 - implemented as an array of characters
31 - terminated by a null character that is the equivalent of 0
32 - Referred to as zero or null terminated strings
33
34 String literal
35 - Sequence of characters in double quotes - like "Josie"
36 - constant
37 - terminated with a null character
38
39 // Example of how it looks in memory
40 c++ is fun\0
41 -----
42
43 // Declaring c-style string variables
44 char MyName[]{"Josie"}; // Compiler will allocate 6 characters - one for the null
45 - BE CAREFUL - if you reassign the variable to a larger string you will go out of bounds
46
47 // Declaring without initializing
48 char myName[8];
49
50 We can't use assignment with these
51 myName = "Josie"; // This is invalid and will not work
52
53 -Instead, use strcpy();
54 strcpy(myName, "Josie"); // This will add null characters
55
56 // Other functions for cstyle strings
57 #include <cstring>
58 Copying, Concatenation, Comparison, Searching, etc.
59
60 strcpy(); // Copy
61 strcat(); // Concatenate
62 strcmp(); // Compare strings
63
64
65
66
67
68
69

```

```

70 // General purpose to convert C-style strings to other types
71 #include <cstdlib>
72
73 Can convert to integer, float, long, etc
74
75 strlen() actually returns a type size_t
76 This is an unsigned type that is dynamically configured when we use it
77 It can be different on different systems but it's always unsigned
78
79 // getline() function
80 cin.getline(fullName, 50); // Get the line and put into the variable, stopping at the
    numebr of characters
81
82 // C-Style string examples
83
84 #include <iostream>
85 #include <vector>
86 #include <string>
87
88 int main(){
89     // Always initialize variables
90     char firstName[20]{};
91     char lastName[20]{};
92     char fullName[50]{};
93     char temp[50]{};
94
95     // These uninitialized will display garbage
96     //char firstName[20];
97     //char lastName[20];
98     //char fullName[50];
99     //char temp[50];
100
101     std::cout << firstName; // Will display garbage
102
103     std::cout << "Enter first name: ";
104     std::cin >> firstName;
105
106     std::cout << "Enter last name:";
107     std::cin >> lastName;
108     std::cout << "-----" << std::endl;
109
110     std::cout << "Hello, " << firstName << ". Your first name has: " << strlen(firstName)
    ) << " characters" << std::endl;
111     std::cout << "Your last name is: " << lastName << ". Your last name has: " <<
    strlen(lastName) << " characters" << std::endl;
112
113     // Copies from the second argument to the first one
114     // Note that the compiler on my system threw errors and made me use strcat_s
115     strcpy_s(fullName, firstName); // Copy first name to full name
116     strcat_s(fullName, " "); // Copy a space to full name
117     strcat_s(fullName, lastName); // Copy last name to full name
118     std::cout << "Your full name is: " << fullName << std::endl;
119
120     // use getline() to get a full line of input
121     std::cout << "Enter your full name: ";
122     std::cin.getline(fullName, 50);
123     std::cout << "Your full name from getline is: " << fullName << std::endl;
124
125     std::cout << "-----" << std::endl;
126     strcpy_s(temp, fullName);
127     // If we get back a 0 that means they are the same
128     // If the string lexically comes before it will return -1
129     // If the string lexically comes after it will return 1
130     if (strcmp(temp, fullName) == 0) {
131         std::cout << temp << " and " << fullName << " are the same " << std::endl;
132     } else {
133         std::cout << temp << " and " << fullName << " are different " << std::endl;
134     }
135     std::cout << "-----" << std::endl;

```

```

136
137     // Change the name to all uppercase letters
138     for (size_t i{ 0 }; i < strlen(fullName); i++) {
139         if (isalpha(fullName[i])) {
140             fullName[i] = toupper(fullName[i]);
141         }
142     }
143
144     // Check it again
145     if (strcmp(temp, fullName) == 0) {
146         std::cout << temp << " and " << fullName << " are the same " << std::endl;
147     } else {
148         std::cout << temp << " and " << fullName << " are different " << std::endl;
149     }
150     std::cout << "-----" << std::endl;
151
152     return 0;
153 }
154
155 // C++ style strings
156 std::string is a class in the Standard Template Library
157 #include <string>
158 std namespace
159 contiguous in memory
160 dynamically sized
161 Works with input and output streams
162 lots of useful member functions
163 can use the familiar operators
164 can be converted to Cstyle strings if needed
165 safer - does bounds checking and etc
166
167 // Initializing strings
168 #include <string>
169
170 std::string s1;                //Empty
171 std::string s2{"Josie"};       // Josie
172 std::string s3{s2};           // Josie
173 // Number of characters to use
174 std::string s4{"Josie", 3};    // Jos
175 // Index start and how many to copy
176 std::string s5{s3, 0, 2};     // Jo
177 // Constructor
178 std::string s6(5, "J");       // JJJJJJ
179
180 // Assignment
181 std::string s1;
182 s1 = "C++ is amazing";
183 std::string s2;
184 s2 = s1;
185
186 // Concatenation
187 std::string part1{"C++"};
188 std::string part2{"Is a powerful"};
189
190 std::string sentence;
191 sentence = part1 + " " + part2 + " language";
192 // This will print C++ is a powerful language
193
194 sentence = "C++" + " is powerful"; // illegal - cannot put to C style string literals
195 together
196
197 // Accessing characters with [] and .at()
198 sentence[0];
199 sentence.at(0);
200
201
202
203

```

```

204 // Can also iterate over
205 for(char c: sentence){
206     std::cout << c << std::endl;
207 }
208
209 // If you use an integer type it will convert the chars to ints in the ascii table
210 for (int c: sentence){
211     std::cout << c << std::endl;
212 }
213
214 // Comparing strings
215 ==, !=, >, >=, <, <=
216
217 The objects are compared character by character lexically
218
219 Can compare:
220 two std::string objects
221 std::string object and C-style string literal
222 std::String object and C-Style string variable
223
224 CANNOT be used on two C-style literals
225
226 // Comparing examples
227 std::string s1{"Apple"}
228 std::string s2{"Banana"}
229 std::string s3{"Kiwi"}
230 std::string s4{"apple"}
231 std::string s5{s1}
232
233 s1 == s5 // True
234 s1 == s2 // False
235 s1 != s2 // True
236 s1 < s2 // True
237 s2 > s1 // True
238 s4 < s5 // False
239 s1 == "Apple" // True
240
241 // Extracting substrings
242 object.substr(start_index, length);
243
244 std::string s1{"This is a string"};
245
246 std::cout << sq.substr(0,4); //This
247 std::cout << sq.substr(5,2); //is
248 std::cout << sq.substr(10,5); //String
249
250 // Searching strings
251 object.find(searchString);
252 std::string s1{"This is a string"};
253 std::cout << s1.find("This"); // Finds it at 0
254 std::cout << s1.find("is", 4); // Starts looking at position 4
255 std::cout << s1.find("XX"); // string::npos - no position, not found
256
257 object.rfind(); // Same as above but in reverse
258
259 // Removing characters - erase() and clear()
260 Removes a substring of characters from a std::string
261
262 object.erase(start_index, length);
263
264 string s1{"This is a test"};
265
266 std::cout << s1.erase(0,5); // Deletes up to the is
267 std::cout << s1.clear(); // Empty string
268
269 // count the string length
270 .length();
271
272

```

```

273 // Compound concatenation operator
274 overloaded +=
275
276 s1 += " of C++"; // will return "This is a test of C++"
277
278 // Input and getline();
279 When you read input with cin, it only takes up to the first whitespace
280 std::string s1;
281 std::cin >> s1; // Hello There
282 // Only accepts to the first space
283 std::cout << s1 << std::endl; // Will only print Hello
284
285
286 // getline(inputstream, variableToStoreIn);
287
288 getline(cin, s1); // Will read the entire line until \n
289 std::cout << s1 << std::endl; // Will display hello there
290
291 // getline(inputStream, variableToStoreIn, delimiter);
292 getline(cin, s1, 'x'); will stop taking input at the delimiter
293
294 // C++ string examples
295 #include <iostream>
296 #include <vector>
297 #include <string>
298 #include <iomanip>
299
300 int main(){
301
302     std::string s0; // Will still be automatically initialized
303     std::string s1{ "Apple" };
304     std::string s2{ "Banana" };
305     std::string s3{ "Kiwi" };
306     std::string s4{ "apple" };
307     std::string s5{ s1 };
308     std::string s6{ s1, 0, 3 }; // first three characters of Apple
309     // Constructor style initialization
310     std::string s7( 10, 'X' ); // 10 X characters
311
312     //std::cout << s0 << std::endl;
313     //std::cout << s0.length() << std::endl;
314
315     // Initialization
316     std::cout << "\nInitialization" << "\n-----" << std::endl;
317     std::cout << "s0 is initialized to: " << s0 << std::endl;
318     std::cout << "s1 is initialized to: " << s1 << std::endl;
319     std::cout << "s2 is initialized to: " << s2 << std::endl;
320     std::cout << "s3 is initialized to: " << s3 << std::endl;
321     std::cout << "s4 is initialized to: " << s4 << std::endl;
322     std::cout << "s5 is initialized to: " << s5 << std::endl;
323     std::cout << "s6 is initialized to: " << s6 << std::endl;
324     std::cout << "s7 is initialized to: " << s7 << std::endl;
325
326     // Comparison
327     std::cout << "\nComparison" << "\n-----" << std::endl;
328     std::cout << std::boolalpha;
329     std::cout << s1 << " == " << s5 << ": " << (s1 == s5) << std::endl;
330     std::cout << s1 << " == " << s2 << ": " << (s1 == s2) << std::endl;
331     std::cout << s1 << " != " << s2 << ": " << (s1 != s2) << std::endl;
332     std::cout << s1 << " < " << s2 << ": " << (s1 < s2) << std::endl;
333     std::cout << s2 << " > " << s1 << ": " << (s2 > s1) << std::endl;
334     // Uppercase characters come before the lowercase ones in ascii table
335     std::cout << s4 << " < " << s5 << ": " << (s4 < s5) << std::endl;
336     std::cout << s1 << " == " << "Apple" << ": " << (s1 == "Apple") << std::endl;
337
338     // Assignment
339     std::cout << "\nAssignment" << "\n-----" << std::endl;
340     s1 = "Watermelon";
341     std::cout << "s1 is now: " << s1 << std::endl;

```

```

342     s2 = s1;
343     std::cout << "s2 is now: " << s2 << std::endl;
344
345     s3 = "Yaya";
346     std::cout << "s3 is now: " << s3 << std::endl;
347
348     s3[0] = 'W';
349     std::cout << "s3 is now: " << s3 << std::endl; // Changes to Wawa
350     s3.at(0) = 'Y';
351     std::cout << "s3 is now: " << s3 << std::endl; // Back to Yaya
352
353     // Concatenation
354     std::cout << "\nConcatenation" << "\n-----" << std::endl;
355     s3 = "Watermelon";
356     s3 = s5 + " and " + s2 + " juice"; // Apple and banana juice
357     std::cout << "s3 is now: " << s3 << std::endl;
358     //s3 = "nice " + " cold" + s5 + "juice"; // compiler error
359
360     // For loop
361     std::cout << "\nLooping" << "\n-----" << std::endl;
362     s1 = "Apple";
363     for (size_t i{ 0 }; i < s1.length(); i++) {
364         std::cout << s1.at(i) << std::endl; // or index style s1[i]
365     }
366
367     // Range-based for loop
368     for (auto letter : s1) {
369         std::cout << letter << std::endl;
370     }
371
372     // Substring
373     std::cout << "\nSubstring" << "\n-----" << std::endl;
374     s1 = "This is a test";
375     std::cout << s1.substr(0, 4) << std::endl; // This
376     std::cout << s1.substr(5, 2) << std::endl; // is
377     std::cout << s1.substr(10, 4) << std::endl; // test
378
379     // Erase
380     std::cout << "\nErase" << "\n-----" << std::endl;
381     s1 = "This is a test";
382     s1.erase(0, 5); // Erase the first five characters
383     std::cout << "s1 is now: " << s1 << std::endl;
384     s1.erase();
385     std::cout << "s1 is now: " << s1 << std::endl;
386
387     // getline()
388     std::cout << "\ngetline()" << "\n-----" << std::endl;
389     std::string fullName{};
390     std::cout << "Enter your full name: ";
391     getline(std::cin, fullName);
392     std::cout << "Your full name is: " << fullName << std::endl;
393
394     // find()
395     std::cout << "\nfind()" << "\n-----" << std::endl;
396     s1 = "The secret word is yaya";
397     std::string word{};
398     std::cout << "Enter the word to find: ";
399     std::cin >> word;
400     size_t position = s1.find(word);
401     if (position != std::string::npos) {
402         std::cout << "Found " << word << " at position: " << position << std::endl;
403     } else {
404         std::cout << "Sorry, that word was not found." << std::endl;
405     }
406     return 0;
407 }
408
409
410

```

```

411 //=====
412 // Section 10 Characters and strings Challenge
413 //=====
414 #include <iostream>
415 #include <vector>
416 #include <string>
417
418 int main(){
419     // Parse each letter, find the position in the main cipher
420
421     // Substitute the corresponding letter in the cipher
422
423     std::string mainAlphabet{"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"};
424     std::string cipherLetters{"zyxwvutsrqponmlkjihgfedcbaZYXWVUTSRQPONMLKJIHGFEDCBA"};
425     std::string phraseFromUser{"This is my test phrase"};
426     std::string swapString{};
427     //std::cout << "Please enter a phrase for the substitution:";
428     //getline(std::cin, phraseFromUser);
429
430     // Note when using size_t or unsigned int this threw an error.
431     // i >= 0 is always true for an unsigned value.
432     // Also got a narrowing conversion error with initialization syntax. Static cast it
433     //since I knew the values
434     //for (signed int i{ static_cast<signed int>(mainAlphabet.length() - 1) }; i >= 0;
435     //--i) {
436     //    std::cout << mainAlphabet.at(i) << " at " << i << std::endl;
437     //}
438
439     for (size_t i{ 0 }; i < phraseFromUser.length(); i++) {
440         // Get the character we need to substitute
441         char letterToSubstitute{ phraseFromUser[i] };
442         // Find what position that character is in the real alphabet
443         int position = mainAlphabet.find(letterToSubstitute);
444         std::cout << position << std::endl;
445         // Check that it exists in the real alphabet
446         if (position == std::string::npos) {
447             // and go back to the start of the loop if it does not
448             continue;
449         } else {
450             phraseFromUser.at(i) = cipherLetters.at(position);
451         }
452     }
453
454     std::cout << phraseFromUser;
455
456     for (size_t i{ 0 }; i < phraseFromUser.length(); i++) {
457         // Get the character we need to substitute
458         char letterToSubstitute{ phraseFromUser[i] };
459         // Find what position that character is in the real alphabet
460         int position = cipherLetters.find(letterToSubstitute);
461         std::cout << position << std::endl;
462         // Check that it exists in the real alphabet
463         if (position == std::string::npos) {
464             // and go back to the start of the loop if it does not
465             continue;
466         } else {
467             phraseFromUser.at(i) = mainAlphabet.at(position);
468         }
469     }
470
471     std::cout << phraseFromUser;
472     return 0;
473 }
474
475
476
477

```

```

478 //=====
479 // Section 10 Characters and strings Franks solution
480 //=====
481
482 #include <iostream>
483 #include <vector>
484 #include <string>
485
486 int main(){
487
488     std::string mainAlphabet{ "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz" };
489     std::string cipherLetters{ "zyxwvutsrqponmlkjihgfedcbaZYXWVUTSRQPONMLKJIHGFEDCBA" };
490     std::string secretMessage{};
491     std::cout << "Enter the message: ";
492     getline(std::cin, secretMessage);
493     std::string encryptedMessage{};
494     std::cout << "\nEncrypting Message..." << std::endl;
495
496     for (char c : secretMessage) {
497         size_t position = mainAlphabet.find(c);
498         if (position != std::string::npos) {
499             char newCharacter{ cipherLetters.at(position) };
500             encryptedMessage += newCharacter;
501         } else {
502             encryptedMessage += c;
503         }
504     }
505
506     std::cout << encryptedMessage << std::endl;
507
508     std::string decryptedMessage{};
509     for (char c : encryptedMessage) {
510         size_t position = cipherLetters.find(c);
511         if (position != std::string::npos) {
512             char newCharacter{ mainAlphabet.at(position) };
513             decryptedMessage += newCharacter;
514         } else {
515             decryptedMessage += c;
516         }
517     }
518     std::cout << decryptedMessage << std::endl;
519
520     return 0;
521 }
522
523

```