Introduction to Programming with C++

Working with Strings

Objective

- In this chapter, you'll learn:
  - How to create variables of type string
  - What operations are available with objects of type string, and how you use them
  - How you can search a string for a specific character or a substring
  - How you can modify an existing string
  - How you can work with strings containing Unicode characters
  - What a raw string literal is

A Better Class of String

- You've seen how you can use an array of elements type char to store a null-terminated (C-style) string.
- The cstring header provides a range of functions for working with C-style strings
- This includes capability for joining strings, searching a string, and comparing strings.
- All these operations depend on the null character being present to mark the end of a string.
- If it is missing or gets overwritten, many of these functions will march happily through memory

A Better Class of String

- That is, beyond the end of a string until a null character is found at some point, or some catastrophe stops the process.
- It often results in memory being arbitrarily overwritten.
- Using C-style strings is inherently unsafe and represents a security risk.
- Fortunately there's a better alternative.
- The string header defines the string type, which is much easier to use than a null-terminated string.
- The string type is defined by a class so it isn't one of the fundamental types.

A Better Class of String

- Type string is a compound type, which is a type that's a composite of several data items that are ultimately defined in terms of fundamental types of data.
- A string object contains the characters that make up the string it represents, and other data, such as number of characters in the string.
- Because the string type is defined in the string header, you must include this header when you're using string objects.
- The string type name is defined within the std namespace, so you need a using declaration to use the type name in its unqualified form.

Defining `string` Objects

- An object of type `string` contains a sequence of characters of type `char`, which can be empty.
- This statement defines a variable of type string that contains an empty string:

```
string empty; // An empty string
```

- This statement defines a string object that you refer to using the name empty.
- In this case empty contains a string that has no characters and so it has zero length.
- You can initialize a string object with a string literal when you define it:

```
string proverb {"Many a mickle makes a muckle."};
```

- proverb is a string object that contains the string literal shown in the initializer list.
- The string that's encapsulated by a string object doesn't have a string termination character.
- A string object keeps track of the length of the string that it represents, so no termination character is necessary.

Defining string Objects

- You can obtain the length of the string for a string object using its length() function, which takes no arguments:

```
std::cout << proverb.length(); // Outputs 29
```

- This statement calls the length() function for the proverb object and outputs the value it returns to cout.
- The record of the string length is maintained by the object itself.
- When you append one or more characters, the length is increased automatically by the appropriate amount and decreased if you remove characters.
- There are some other possibilities for initializing a string object.
- You can use an initial sequence from a string literal for instance:

```
string partLiteral{"Least said soonest mended.", 5}; // "Least"
```

Defining string Objects

- The second initializer in the list specifies the length of the sequence from the first initializer to be used to initialize the `partLiteral` object.
- You can't initialize a string object with a single character between single quotes
- You must use a string literal between double quotes, even when it's just one character.
- However, you can initialize a string with any number of instances of given character.
- You can define and initialize a sleepy time string object like this:

```
string sleeping {6, 'z'};
```

- The string object, sleeping, will contain "zzzzzz".

Defining string Objects

- The string length will be 6. If you want to define a string object that's more suited to a light sleeper, you could write this:

```
string lightSleeper {1, 'z'};
```

- This initialize lightSleeper with the string literal "z".
- A further option is to use an existing string object to provide the initial value.
- Given that you've defined proverb previously, you can define another object based on that:

```
string sentence {proverb};
```

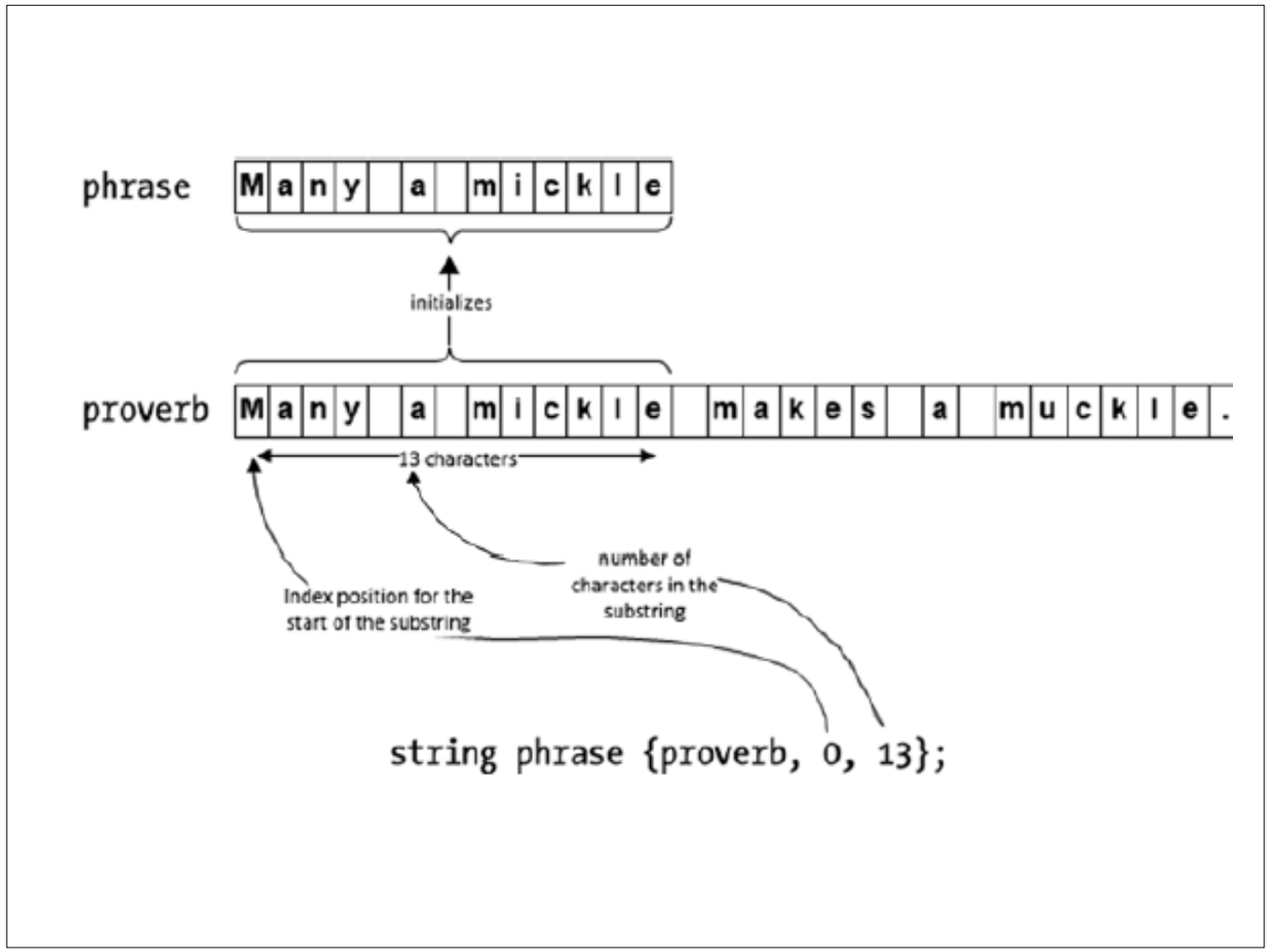
- The sentence object will be initialized with the string literal that proverb contains, so it too will contain `"Many a mickle makes a muckle."` and have a length of 29.
- You can reference characters within a string object using an index value starting from 0, just like an array.

Defining string Objects

- You can use a pair of index values to identify part of an existing string and use that to initialize a new string object
- For example:

```
string phrase {proverb, 0, 13};
```

- The first initializer in the initializer list is the source of the initializing string.
- The second is the index of the character in proverb that begins the initializing substring
- The third initializer in the list is the number of characters in the substring.
- Thus phrase will contain `"Many a mickle"`.


phrase
M a n y a m i c k l e
initializes
proverb
M a n y a m i c k l e m a k e s a m u c k l e .
13 characters
number of characters in the substring
Index position for the start of the substring
string phrase {proverb, 0, 13};

Just to summarize

- No initializer list (or an empty list):

```
string empty; // The string ""
```

- An initializer list containing a string literal:

```
string proverb {"Many a mickle makes a muckle."};//The
literal
```

- An initializer list containing an existing string object:

```
string sentence {proverb}; // Duplicates proverb
```

- An initializer list containing two initializers that are a string literal followed by the length of the sequence in the literal to be used to initialize the string object:

```
string partLiteral {"Least said soonest mended.", 5};// "Least"
```

Just to summarize

- An initializer list containing two initializers that are a repeat count followed by the character literal that is to be repeated in the string that initializes the string object:

```
string sleeping {6, 'z'}; // "zzzzzz"
```

- An initializer list containing three initializers that are an existing string object, an index specifying the start of the substring in the first initializer, and the length of the substring:

```
string phrase {proverb, 0, 13}; // "many a mickle"
```

Operations with String Objects

- A wide range of operations with string objects are supported.
- You can assign a string literal or another string object to a string object

```
string adjective {"hornswoggling"}; // Defines adjective
string word {"rubbish"}; // Defines word
word = adjective; // Modifies word
adjective = "twotiming"; // Modifies adjective
```

- You can join strings using the addition operator; the technical term for this is concatenation.
- You can concatenate the objects defined above:

```
string description {adjective + " " + word + " whippersnapper"};
```

- The third statement assigns the value of adjective, which is "hornswoggling", to word, so "rubbish" is replaced.
- The last statement assigns the literal, "twotiming" to adjective, so the original value "hornswoggling" is replaced.
- Thus, after executing these statements, word will contain "hornswoggling" and adjective will contain "twotiming".

Operations with String Objects

- Note that you can't concatenate two string literals using the + operator.
- One of the operands must always be an object of type string.
- The following statement, for example, won't compile:

```
string description { "whippersnapper" + " " + word};
```

See Ex7_01.cpp

Accessing Characters in a String

- You refer to a particular character in a string by using an index value between square brackets, just as you do with an array.
- The first character in a string object has the index value 0.
- You could refer to the third character in sentence, for example, as sentence[2].
- You can use such an expression on the left of the assignment operator, so you can replace individual characters as well as access them.
- The following loop changes all the characters in sentence to uppercase:

```
for(size_t i {} ; i < sentence.length() ; ++i)
   sentence[i] = std::toupper(sentence[i]);
```

Accessing Characters in a String

- This loop applies the `toupper()` function to each character in the string in turn and stores the result in the same position in the string.
- The index value for the first character is 0, and the index value for the last character is one less than the length of the string, so the loop continues as long as `i<sentence.length()` is true.
- A string object is a range, so you could also do this with the range-based for loop:

```
for (auto& ch : sentence)
   ch = std::toupper(ch);
```

- Specifying ch as a reference type allows the character in the string to be modified within the loop.
- This loop and the previous loop require the locale header to be included to compile.

See Ex07_2.cpp

Accessing Substrings

- You can extract a substring from a string object using its `substr()` function.
- The function requires two arguments.
- The first is the index position where the substring starts and the second is the number of characters in the substring.
- The function returns the substring as a `string` object.
- For example:

```
string phrase {"The higher the fewer."};
string word1 {phrase.substr(4, 6)}; // "higher"
```

- This extracts the six-character substring from phrase that starts at index position 4, so word1 will contain
- "higher" after the second statement executes.
- If the length you specify for the substring overruns the end of the string object, then the substr() function just returns an object contains the characters up to the end of the string.
- The following statement demonstrates this behavior:

```
string word2 {phrase.substr(4,100)}; // "higher the fewer."
```

- Of course, there aren't 100 characters in phrase, let alone in a substring.
- In this case the result will be that word2 will contain the substring from index position 4 to the end, which is "higher the fewer.".
- You could obtain the same result by omitting the length argument and just supplying the first argument that specifies the index of the first character of the substring:
- string word {phrase.substr(4)}; // "higher the fewer."
- This version of substr() also returns the substring from index position 4 to the end.
- If you omit both arguments to substr(), the whole of phrase will be selected as the substring.
- If you specify a starting index for a substring that is outside the valid range for the string object, an exception of type std::out_of_range will be thrown and your program will terminate abnormally—unless you've implemented some code to handle the exception.

Comparing Strings

- When you access a character using an index, the result is of type char, so you can use the comparison operators to compare individual characters.
- You can also compare entire string objects using any of the comparison operators.
- The comparison operators you can use are:

```
> >= < <= == !=
```

- You can use these to compare two objects of type string, or to compare a string object with a string literal or with a C-style string.

Comparing Strings

- The operands are compared character by character until either a pair of corresponding characters are different, or the end of either or both operands is reached.
- When a pair of characters differ, numerical comparison of the character codes determines which of the strings has the lesser value.
- If no differing character pairs are found and the strings are of different lengths, the shorter string is “less than” the longer string.
- Two strings are equal if they contain the same number of characters and all corresponding character codes are equal.
- Because you’re comparing character codes, the comparisons are obviously going to be case sensitive.

Comparing Strings

- You could compare two string objects using this if statement:

```
string word1 {"age"};
string word2 {"beauty"};
if(word1 < word2)
   std::cout << word1 << " comes before " <<
   word2 << "." << std::endl;
else
   std::cout << word2 << " comes before " <<
   word1 << "." << std::endl;
```

See Ex7_03.cpp

The compare() Function

- The compare() function for a string object can compare the object with another string object, or with a string literal, or with a C-style string.
- Here's an example of an expression that calls compare() for a string object, word, to compare it with a string literal:

```
word.compare("and");
```

- word is compared with the argument to compare().
- The function returns the result of the comparison as a value of type int.
- This will be a positive integer if word is greater than "and", zero if word is equal to "and", and a negative integer if word is less than "and".

Summary

- The essential points you have learned in this chapter are:
  - The std::string type stores a character string without a termination character.
  - The terminating null is unnecessary because a string object keeps track of the length of the string.
  - You can access and modify individual characters in a string object using an index between square brackets.
  - Index values for characters in a string object start at 0.
  - You can use the + operator to concatenate a string object with a string literal, a character, or another string object.
  - Objects of type string have functions to search, modify, and extract substrings.
  - You can store string objects in an array, or better still, in a sequence container such as a vector.

Reading Assignment

1. Searching & Modifing Strings(Page 198 - 207)
2. Strings of International (Page 208-210)
3. Raw String Literals (page 210)

Exercise

1. Write a program that reads and stores the first names of any number of students, along with their grades. Calculate and output the average grade, and output the names and grades of all the students in a table with the name and grade for three students on each line.
2. Write a program that reads text entered over an arbitrary number of lines. Find and record each unique word that appears in the text and record the number of occurrences of each word. Output the words and the number of occurrences of each word, three words and their counts per line. Words and counts should align in columns.

Exercise

3. Write a program that reads a text string of arbitrary length from the keyboard and prompt for entry of a word that is to be found in the string. The program should find and replace all occurrences of this word, regardless of case, by as many asterisks as there are characters in the word. It should then output the new string. Only whole words are to be replaced. For example, if the string is "Our house is at your disposal." and the word that is to be found is “our,” then the resultant string should be: "*** house is at your disposal." and not "*** house is at y*** disposal."
4. Write a program that prompts for input of two words and determines whether one is an anagram of the other.
5. Write a program that reads a text string of arbitrary length from the keyboard followed by a string containing one or more letters. Output a list of all the whole words in the text that begin with any of the letters, upper or lowercase.