【C++】Day35(2)

Class	C++
Date	@January 7, 2022
Material	
Series Number	
Summary	Additional string Operations

【Ch9】Sequential Container

9.5 Additional string Operations

9.5.1 Other Ways to Construct strings

The following table introduces several other ways to construct a string:

n, len2 and pos2 are all unsigned values	
`string s(cp, n);`	s is a copy of the first n characters in the array to which cp points. That array must have at least n characters.
`string s(s2, pos2);`	s is a copy of the characters in the `string` s2 starting at the index pos2. Undefined if `pos2 > s2.size()`.
`string s(s2, pos2, len2);`	s is a copy of `len2` characters from s2 starting at the index pos2. Undefined if `pos2 > s2.size()`. Regardless of the value of `len2`, copies at most `s2.size() - pos2` characters.

```
const char *cp = "Hello World!!!"; //null-terminated array
char noNull[] = { 'H', 'i' }; //not null terminated
string s1(cp); //copy up to the null in cp; s1 == "Hello World!!!"
string s2(noNull, 2); //copy 2 characters from noNull; s2 == "Hi"
string s3(noNull); //undefineds: noNull not null terminated
string s4(cp + 6, 5); //copy 5 characters starting at cp[6]; s4 == "World"
string s5(s1, 6, 5); //copy 5 characters starting at s1[6]; s5 == "World!!!"
string s6(s1, 6); //copy from s1[6] to end of s1; s6 == "World!"
string s7(s1, 6, 20); //ok, copies only to end of s1; s7 == "World!!!"
string s8(s1, 16); //throw an out_of_range error
```

Ordinarily when we create a string from a `const char*`, the array to which the pointer points must be null terminated; characters are copied up to the null.

If we also pass a count, the array does not have to be null terminated.

If we do not pass a count and there is no null, or if the given count is greater than the size of the array, the operation is undefined.

When we copy from a string, we can supply an optional starting position and a count. The starting position must be less than or equal to the size of the given string.

If the position is greater than the size, then the constructor throws an `out_of_range` exception.

When we pass a count, that many characters are copied, starting from the given position. Regardless of how many characters we ask for, the library copies up to the size of the string, but not more.

The substr Operation

Table 9.12. Substring Operation

`s.substr(pos, n)`	Return a `string` containing `n` characters from `s` starting at `pos`. `pos` defaults to 0. `n` defaults to a value that causes the library to copy all the characters in `s` starting from `pos`.

```
string s("hello world");
string s2 = s.substr(0, 5); //s2 == "hello"
string s3 = s.substr(6); //s3 == "world"
string s4 = s.substr(6, 11); //s4 == "world"
string s5 = s.substr(12); //throw and out_of_range error
```

Exercise

Exercise 9.41: Write a program that initializes a `string` from a `vector<char>`.

```
std::vector<char> vec = {'a', 'b', 'c'};
std::string str(vec.cbegin(), vec.cend());
```

Exercise 9.42: Given that you want to read a character at a time into a `string`, and you know that you need to read at least 100 characters, how might you improve the performance of your program?

We can call `str.reserve(100)` to preallocate memory for 100 characters.

9.5.2 Other Ways to Change a String

In addition to the version of `insert` and `erase` that take iterators, string provides versions that take an index.

The index indicates the starting element to erase or the position before which to insert the given values:

```
s.insert(s.size(), 5, '!'); //insert five '!' at the end of s
s.erase(s.size() - 5, 5); //erase the last five characters from s
```

The string library also provides versions of insert and assign that takes C-style character arrays.

For example, we can use a null-terminated character array as the value to insert or assign into a string:

```
const char *cp = "Stately, plump Buck";
s.assign(cp, 7); //s == "Stately"
s.insert(s.size(), cp + 7);
```

The append and replace Functions

The string class defines two additional members, `append` and replace, that can change the contents of a string.

`s.insert(pos, args)`	Insert characters specified by *args* before *pos*. *pos* can be an index or an iterator. Versions taking an index return a reference to `s`; those taking an iterator return an iterator denoting the first inserted character.
`s.erase(pos, len)`	Remove `len` characters starting at position `pos`. If `len` is omitted, removes characters from `pos` to the end of the `s`. Returns a reference to `s`.
`s.assign(args)`	Replace characters in `s` according to *args*. Returns a reference to `s`.
`s.append(args)`	Append *args* to `s`. Returns a reference to `s`.
`s.replace(range, args)`	Remove *range* of characters from `s` and replace them with the characters formed by *args*. *range* is either an index and a length or a pair of iterators into `s`. Returns a reference to `s`.

***args* can be one of the following; `append` and `assign` can use all forms**
`str` must be distinct from `s` and the iterators `b` and `e` may not refer to `s`

`str`	The `string str`.
`str, pos, len`	Up to `len` characters from `str` starting at `pos`.
`cp, len`	Up to `len` characters from the character array pointed to by `cp`.
`cp`	Null-terminated array pointed to by pointer `cp`.
`n, c`	`n` copies of character `c`.
`b, e`	Characters in the range formed by iterators `b` and `e`.
initializer list	Comma-separated list of characters enclosed in braces.

***args* for `replace` and `insert` depend on how *range* or *pos* is specified.**

`replace` (pos,len,*args*)	`replace` (b,e,*args*)	`insert` (pos,*args*)	`insert` (iter,*args*)	*args* can be
yes	yes	yes	no	`str`
yes	no	yes	no	`str, pos, len`
yes	yes	yes	no	`cp, len`
yes	yes	no	no	`cp`
yes	yes	yes	yes	`n, c`
no	yes	no	yes	`b2, e2`
no	yes	no	yes	*initializer list*

The append operation is a shorthand way of inserting at the end:

```
string s("C++ Primer"), s2 = s;
s.insert(s.size(), " 4th Edition"); //s == "C++ Primer 4th Edition"
s2.append(" 4th Edition"); //s == s2
```

The replace operations are a shorthand way of calling `erase` and `insert`:

```
s.erase(11, 5); //s == "C++ Primer Edition"
s.insert(11, "5th"); //s == "C++ Primer 5th Edition"
s2.replace(11, 3, "5th"); //starting at position 11, erase 3 characters and insert "5th"
```

We can insert a larger or smaller string:

```
s.replace(11, 5, "Fifth");
```