【C++】Day eleven

Class	C++
Date	@November 29, 2021
Material	
Series Number	
Summary	

【Ch3】C-Style String

3.5.4 C-Style Character String

Warning: Although C++ supports C-style strings, they should not be used by C++ programs. C-style strings are a surprisingly rich source of bugs and are the root cause of many security problems.

C-style strings are not a type. Instead, they are a convention for how to represent and use character strings. Strings that follow this convention are stroed in character arrays and are null terminated.(the last character is '\0'.)

C Library String Functions

The Standard C library provides a set of functions that operate on C-style strings. These functions are defined in the `cstring` header, which is the C++ version of the C header `string.h`.

Table 3.8. C-Style Character String Functions

`strlen(p)`	Returns the length of `p`, *not counting the null.*
`strcmp(p1, p2)`	Compares `p1` and `p2` for equality. Returns `0` if `p1 == p2`, a positive value if `p1 > p2`, a negative value if `p1 < p2`.
`strcat(p1, p2)`	Appends `p2` to `p1`. Returns `p1`.
`strcpy(p1, p2)`	Copies `p2` into `p1`. Returns `p1`.

Note: The pointers passed to these routines must point to null-terminated arrays

Comparing Strings

Comparing two C-style strings is done quite differently from how we compare library strings. When we compare two library strings, we use the normal relational or equality oeprator:

```
string s1 = "Abc";
string s2 = "Adc";
if(s1 < s2) //true
```

Using these operators on similar defined C-style strings compares the pointer values, not the strings themselves:

```
const char ca1[] = "A string example";
const char ca2[] = "A different string";
if( ca1 < ca2) //undefined: compares two unrelated addresses
```

To copmare the strings, rather than the pointer values, we can call `strcmp`. That function returns 0 if the strings are equal, or a positive or negative value, depending on whether the first string is larger or smaller than the second:

```
if(strcmp(ca1, ca2) < 0) //same effect as string comparison s1 < s2
```

Caller is Responsible for Size of a Destination String

Concatenating or copying C-style strings is also very different from the same operations on library strings.

We use `strcat` and `strcpy`. However, to use these functions, we must pass an aray to hold the resulting string. The array we pass must be large enough to hold the generated string, including the null character at the end. The code we show here is fraught with potential for serious error:

```
//disastrous if we miscalculated the size of largeStr
strcpy(largeStr, ca1); //copes ca1 into largeStr
strcat(largeStr, " "); //adds a space at the end of largeStr
strcat(largeStr, ca2); //concatenates ca2 onto largeStr
```

3.5.5 Interfacting to Older Code

Mixing Library strings and C-style Strings

We can use a null-terminated character array anywhere that we can use a string literal:

- We can use a null-terminated character array to initialize or assign a string
- We can use a null-terminated character array as one operand(but not both operands) to the string addition operator or as the right-hand operand in the string compound assignment `+=` operator.

The reverse functionality is not provided: There is no direct way to use a library `string` when a C-style string is required. For example, there is no way to initialize a character pointer from a string. There is, however, a string member function named `c_str` that we can often use to accomplish what we want:

```
char *str = s; //error: can't initialize a char* from a string
const char *str = s.c_str(); //ok
```

The name `c_str` indicates that the function returns a C-style character string. That is, it returns a pointer to the beginning of a null-terminated character aray that holds the same data as the caracters in the string. The type of the pointer is `const char*`, which prevents us from changing the contents of the array.

Warning: If a program needs continuing access to the contents of the array returned by `str()`, the program must copy the array returned by `c_str`.

Using an Array to Initialize a vector

We cannot use a vector to initialize an array. However, we can use an array to initialize a vector. To do so, we specify the address of the first element and one past the last element that we wish to copy:

```
int int_arr[] = {0, 1, 2, 3, 4};
//ivec has five elements; each is a copy of the corresponding element in int_arr
vector<int> ivec(begin(int_arr), end(int_arr));
```

The two poniters used to construct `ivec` mark the range of values to use to initialize the elements in ivec.

The specified range can be a subset of the array:

```
//copies three element: int_arr[1], int_arr[2], int_arr[3]
vector<int> subVec(int_arr + 1, int_arr + 4);
```