Advanced C

Strings and Vectors in C++

Th.S Nguyen Minh Anh

Phenikaa University

Last Update: 7th February 2023

Outline

Library string Type

- A string is a variable-length sequence of characters.
- To use the string type, we must include the string header.

```
#include <string>
using std::string;
```

Defining and Initializing strings

The most common ways to initialize `strings`:

```
string s1; // default initialization; s1 is the empty string
string s2(s1); // s2 is a copy of s1
string s2 = s1; // equivalent to s2(s1), s2 is a copy of s1
string s3 = "saka"; // s3 is a copy of the string literal
string s3("saka"); // equivalent to s3 = "saka"
string s4(10, 'c'); // s4 is cccccccccc
```

Outline

The most common string operations

`os << s`	Writes s onto output stream os. Returns os.
`is >> s`	Reads whitespace-separated string from is into s. Returns is.
`getline(is, s)`	Reads a line of input from is into s. Returns is.
`s.empty()`	Returns true if s is empty; otherwise returns false.
`s.size()`	Returns the number of characters in s.
`s[n]`	Returns a reference to the char at position n in s; positions start at 0.
`s1 + s2`	Returns a string that is the concatenation of s1 and s2.
`s1 = s2`	Replaces characters in s1 with a copy of s2.
`s1 == s2`	The strings s1 and s2 are equal if they contain the same characters.
`s1 != s2`	Equality is case-sensitive.
$<, <=, >, >=$	Comparisons are case-sensitive and use dictionary ordering.

Reading and Writing strings

- The string input operator reads and discards any leading whitespace (e.g., spaces, newlines, tabs). It then reads characters until the next whitespace character is encountered.

```
int main()
{
  string s; // empty string
  cin >> s; // read a whitespace-separated string into s
  cout << s << endl; // write s to the output
  return 0;
}
```

Question: If the input to this program is `"  Hello World!  "`, the output?

Reading and Writing strings

- The string input operator reads and discards any leading whitespace (e.g., spaces, newlines, tabs). It then reads characters until the next whitespace character is encountered.

```
int main()
{
  string s; // empty string
  cin >> s; // read a whitespace-separated string into s
  cout << s << endl; // write s to the output
  return 0;
}
```

Question: If the input to this program is `"  Hello World!  "`, the output? `"Hello"`

Reading and Writing strings

- Like the input and output operations on the built-in types, the string oper ators return their left-hand operand as their result. Thus, we can chain together multiple reads or writes:

```
string s1, s2;
cin >> s1 >> s2; // read first input into s1, second into s2
cout << s1 << s2 << endl; // write both strings
```

Question: If we give this version of the program the same input `"  Hello World!  "`, our output would be?

Reading and Writing strings

- Like the input and output operations on the built-in types, the string oper ators return their left-hand operand as their result. Thus, we can chain together multiple reads or writes:

```
string s1, s2;
cin >> s1 >> s2; // read first input into s1, second into s2
cout << s1 << s2 << endl; // write both strings
```

Question: If we give this version of the program the same input `"  Hello World!  "`, our output would be? `"HelloWorld!"`

Reading and Writing strings

Reading an Unknown Number of strings

```
int main()
{
  string word;
  while (cin >> word) // read until end-of-file
   cout << word << endl; // write each word followed by a new line
  return 0;
}
```

Using getline to Read an Entire Line

If we **do not want to ignore the whitespace** in our input, we can use the **getline** function instead of the >> operator. The getline function:

- takes an input stream and a string
- reads the given stream up to and including the first newline and stores what it read—not including the newline—in its string argument
- After getline sees a newline, even if it is the first character in the input, it stops reading and returns.
- If the first character in the input is a newline, then the resulting string is the empty string.

Using getline to Read an Entire Line

Like the input operator, **getline returns its istream argument**. As a result, we can use getline as a condition.
For example, we can rewrite the previous program that wrote one word per line to write a line at a time instead:

```
int main()
{
  string line;
  // read input a line at a time until end-of-file
  while (getline(cin, line))
   cout << line << endl;
  return 0;
}
```

The string empty and size Operations

- The **empty** function returns a bool indicating whether the string is empty.

```
// read input a line at a time and discard blank lines
while (getline(cin, line))
  if (!line.empty())
   cout << line << endl;
```

- The **size** member returns the length of a string (i.e., the number of charac- ters in it).

```
string line;
// read input a line at a time and print lines that are longer than 80
    characters
while (getline(cin, line))
  if (line.size() > 80)
   cout << line << endl;
```

Comparing strings

- The equality operators (== and !=) test whether two strings are equal or unequal, respectively.
- The relational operators $<, <=, >, >=$ test whether one string is less than, less than or equal to, greater than, or greater than or equal to another in a lexicographical manner.

```
string str = "Hello";
string phrase = "Hello World";
string slang = "Hiya";
```

Question: `str` vs `phrase` vs `slang`?

Assignment for strings

We can assign one string object to another:

```
string st1(10, 'c'), st2; // st1 is ccccccccc; st2 is an empty string
st2 = "Bukayo Saka"; // st2 now is assigned to literal "Bukayo Saka"
st1 = st2; // assignment: replace contents of st1 with a copy of st2
        // both st1 and st2 are now the empty string
```

Adding Two strings

```
string s1 = "hello, ", s2 = "world\n";
string s3 = s1 + s2; // s3 is hello, world\n
s1 += s2; // equivalent to s1 = s1 + s2
```

Adding Literals and strings

The string library lets us convert both character literals and character string literals to strings.

```
string s1 = "hello", s2 = "world";
string s3 = s1 + ", " + s2 + '\n';
```

```
string s4 = s1 + ", ";
string s5 = "hello" + ", ";
string s6 = s1 + ", " + "world";
string s7 = "hello" + ", " + s2;
```

Adding Literals and strings

The string library lets us convert both character literals and character string literals to strings.

```
string s1 = "hello", s2 = "world";
string s3 = s1 + ", " + s2 + '\n';
```

```
string s4 = s1 + ", "; // ok: adding a string and a literal
string s5 = "hello" + ", "; // error: no string operand
string s6 = s1 + ", " + "world"; // ok: each + has a string operand
string s7 = "hello" + ", " + s2; // error: cant add string literals
```

Warning: For historical reasons, and for compatibility with C, string literals are not standard library strings. It is important to remember that these types differ when you use string literals and library strings.

Outline

Processing Every Character? Use Range-Based `for`

To iterates through the elements in a given sequence and performs some operation on each value in that sequence

```
for (declaration : expression)
   statement
```

- `expression` is an object of a type that represents a sequence
- `declaration` defines the variable that we'll use to access the underlying elements in the sequence

As a string represents a sequence of characters

```
string str("some string");
// print the characters in str one character to a line
for (auto c : str)  // for every char in str
   cout << c << endl; // print the current char followed by a newline
```

Using a Range for to Change the Characters in a string

If we want to change the value of the characters in a string, we must define the loop variable as a reference type

```
string s("Hello World!!!"); // convert s to uppercase
for (auto &c : s) // for every char in s
   c = toupper(c); // c is a reference, so the assignment changes the
       char in s
cout << s << endl;
```

Processing Only Some Characters?

The following example uses the **subscript operator** to print the first character in a string:

```
if (!s.empty()) // make sure there's a character to print
   cout << s[0] << endl; // print the first character in s
```

or to print the first n characters:

```
if (!s.empty()) // make sure there's a character to print
   for (int i = 0; i < n; ++i)
      cout << s[i] << endl;
```

Outline

Outline

Vector

- **a collection of objects**, all of which have the **same type**.
- every object in the collection has an associated **index**, which gives access to that object.
- often referred to as a **container** because it "contains" other objects

Vector

To use a vector, we must include the appropriate header.

```
#include <vector>
using std::vector;
```

The additional information we supply is the **type** of the objects the vector will hold:

```
vector<int> ivec; // ivec holds objects of type int
vector<Sales_item> Sales_vec; // holds Sales_items
vector<vector<string>> file; // vector whose elements are vectors
```

Defining vectors

- We can default initialize a vector, which creates an **empty vector** of the specified type:

```
vector<string> svec; // default initialization; svec has no
    elements
```

- We can also supply initial value(s) by copying elements from another vector.

```
vector<int> ivec; // initially empty
// give ivec some values
vector<int> ivec2(ivec); //copy elements of ivec into ivec2
vector<int> ivec3 = ivec; //copy elements of ivec into ivec3
vector<string> svec(ivec2); //error: svec holds strings, not ints
```

List Initializing a vector

We can **list initialize** a vector from a list of **zero or more** initial element values enclosed in **curly braces**:

```
vector<string> articles = {"a", "an", "the"};
vector<string> v1{"a", "an", "the"}; // list initialization
vector<string> v2("a", "an", "the"); // error
```

Creating a Specified Number of Elements

We can also initialize a vector from a count and an element value

```
vector<int> ivec(10, -1); // ten int elements, each initialized to -1
vector<string> svec(10, "hi!"); // ten strings; each element is "hi!"
```

Common ways to Initialize a vector

Table 3.4: Ways to Initialize a `vector`

`vector<T> v1`	`vector` that holds objects of type `T`. Default initialization; `v1` is empty.
`vector<T> v2(v1)`	`v2` has a copy of each element in `v1`.
`vector<T> v2 = v1`	Equivalent to `v2(v1)`, `v2` is a copy of the elements in `v1`.
`vector<T> v3(n, val)`	`v3` has `n` elements with value `val`.
`vector<T> v4(n)`	`v4` has `n` copies of a value-initialized object.
`vector<T> v5{a,b,c...}`	`v5` has as many elements as there are initializers; elements are initialized by corresponding initializers.
`vector<T> v5 = {a,b,c...}`	Equivalent to `v5{a,b,c...}`.

Outline

Adding Elements to a vector

The push_back operation takes a value and **"pushes"** that value as a new last element onto the **"back" of the vector**.

```
vector<int> v2; // empty vector
for (int i = 0; i != 100; ++i)
   v2.push_back(i); // append sequential integers to v2
// at end of loop v2 has 100 elements, values 0 . . . 99
```

Using the same approach, we can read and store an unknown number of values in text

```
   // read words from the standard input and store them as elements in a
       vector
   string word;
   vector<string> text; // empty vector
   while (cin >> word)
      text.push_back(word); // append word to text
```

Questions

Create a vector of n first prime numbers
Create a vector containing numbers satisfy certain conditions

Other vector operations

Table 3.5: `vector` Operations

`v.empty()`	Returns `true` if `v` is empty; otherwise returns `false`.
`v.size()`	Returns the number of elements in `v`.
`v.push_back(t)`	Adds an element with value `t` to end of `v`.
`v[n]`	Returns a reference to the element at position `n` in `v`.
`v1 = v2`	Replaces the elements in `v1` with a copy of the elements in `v2`.
`v1 = {a,b,c...}`	Replaces the elements in `v1` with a copy of the elements in the comma-separated list.
`v1 == v2` `v1 != v2`	`v1` and `v2` are equal if they have the same number of elements and each element in `v1` is equal to the corresponding element in `v2`.
`<, <=, >, >=`	Have their normal meanings using dictionary ordering.

Outline

Exercises