【C++】Day29(2)

Class	C++
Date	@December 31, 2021
Material	
Series Number	
Summary	String Streams

【Ch8】The IO Library

8.3 string Streams

The `sstream` header defines three types to support in-memory IO; these types read from or write to a string as if the string were an IO stream.

The `istringstream` type reads a string, `ostringstream` writes a string, and `stringstream` reads and writes the string.

Like the `fstream` types, the types defined in `sstream` inherit from the types we have used from the `iostream` header. In addition to the operations they inherit, the types defined in `sstream` add members to manage the string associated with the stream.

The added operations are listed in the following table. They may be called on stringsteram objects but no on the other IO types.

Table 8.4. File Modes

`in`	Open for input
`out`	Open for output
`app`	Seek to the end before every write
`ate`	Seek to the end immediately after the open
`trunc`	Truncate the file
`binary`	Do IO operations in binary mode

8.3.1 Using an istringstream

An `istringstream` is often used when we have some work to do on an entire line, and other work to do with individual words within a line.

As one example, assume we have a file that lists people and their associated phone numbers.

morgan 2015552368 8625550123
drew 9735550130
lee 6095550132 2015550175 8005550000

Each record in this file starts with a name, which is followed by one or more phone numbers. We will start by defining a simple class:

```
struct PersonInfo {
  string name;
  vector<string> numbers;
};
```

Our program will read the data file and build up a vector of `PersonInfo`.

```
string line, word; //will hold a line and word from input, respectively
vector<PersonInfo> people; //will hold all hte records from the input
while(getline(cin, line)) {
  PersonInfo person; //create an object to hols this record's data
  std::istringstream record(line); //bind record to the current line
  record >> person.name; //read the name
  while(record >> word) //read the phone numbers
    person.numbers.push_back(word); //and store them
  people.push_back(person); //append this record to people
}
```

8.3.2 Using ostringstreams

An `ostringstream` is useful when we need to build up our output a little at a time but do not want to print the output until later.

See the following code for an example. We want to print the info of people if all of the numbers are valid, print error if not:

```
for(const auto &entry : people) {
  ostringstream formatted, badNums; //objects created on each loop
  for(const auto &nums : entry.numbers) {
    //valid defined elsewhere
    if(valid(nums)) {
      formatted << " " << nums;
    } else {
      badNums << " " << nums;
    }
  }
  if(badNums.str().empty())
    os << entry.name << " " << formatted.str() << endl;
  else
    cerr << "Input Error: " << entry.name << " invalid number(s)" << badNums.str() << endl;
}
```

Exercise

Exercise 8.14: Why did we declare `entry` and `nums` as `const auto &`?

By using reference, we avoid copy every object in vectors. By using `const`, we avoid accidentally changing the values of elements in vectors.