【C++】Day28(2)

Class	C++
Date	@December 30, 2021
Material	
Series Number	
Summary	

【Ch8】The IO Library

8.2 File Input and Output

The `fstream` header defines three types to support file IO:

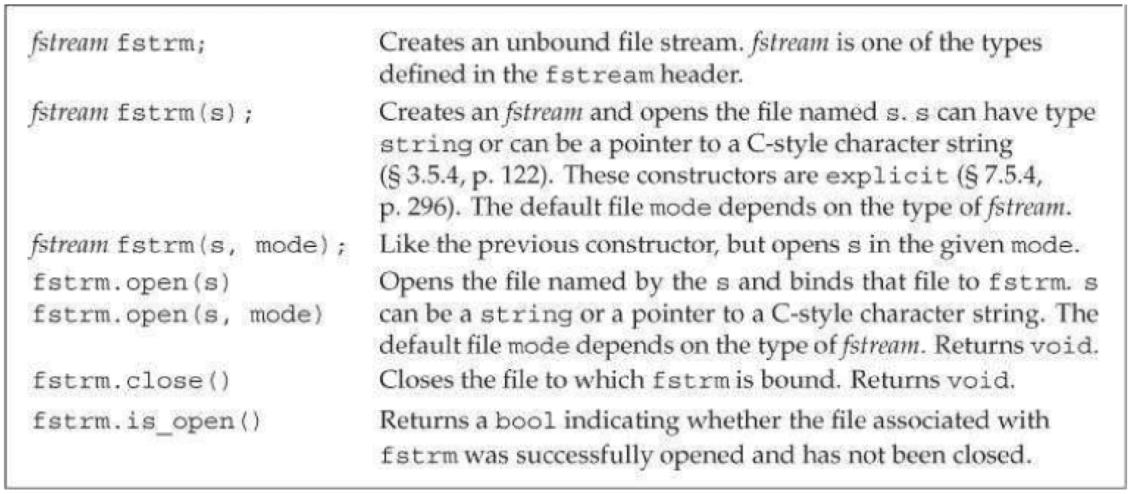
1. `ifstream` to read from a given file
2. `ofstream` to write to a given file
3. `fstream` which reads and writes a given file

We use the IO operators (`<<` and `>>`) to read and write files, we can use `getline` to read an `ifstream`.

In addition to the behaviour that they inherit from the iostream types, the types defined in `fstream` add members to manage the file associated with the stream.

The following operations can be called on objects of `fstream`, `ifstream`, or `ofstream` but not on the other IO types.

Table 8.3. fstream-Specific Operations



fstream `fstrm;`	Creates an unbound file stream. *fstream* is one of the types defined in the `fstream` header.
fstream `fstrm(s);`	Creates an *fstream* and opens the file named `s`. `s` can have type `string` or can be a pointer to a C-style character string (§ 3.5.4, p. 122). These constructors are `explicit` (§ 7.5.4, p. 296). The default file `mode` depends on the type of *fstream*.
fstream `fstrm(s, mode);`	Like the previous constructor, but opens `s` in the given `mode`.
`fstrm.open(s)` `fstrm.open(s, mode)`	Opens the file named by the `s` and binds that file to `fstrm`. `s` can be a `string` or a pointer to a C-style character string. The default file `mode` depends on the type of *fstream*. Returns `void`.
`fstrm.close()`	Closes the file to which `fstrm` is bound. Returns `void`.
`fstrm.is_open()`	Returns a `bool` indicating whether the file associated with `fstrm` was successfully opened and has not been closed.

8.2.1 Using File Stream Objects

When we want to read or write a file, we define a file stream object and associate that object with the file. Each file stream class defines a member function named `open` that does whatever system-specific operations are required to locate the given file and open it for reading or writing as appropriate.

When we create a file stream, we can (optionally) provide a file name. When we supply a file name, `open` is called automatically:

```
ifstream in(ifile); //construct an ifstream and open the given file
ofstream out; //output file stream that is not associated with any file.
```

This code defines `in` as an input stream that is initialized to read from the file named by the string argument file.

It defines `out` as an output stream that is not yet associated with a file.

With the new standard, file names can be either library strings or C-style character arrays.

Using an fstream in Place of an iostream&

We can use an object of an inherited type in places where an object of the original type is expected. This fact means that functions that are written to take a reference(or pointer) to one of the iostream types can be called on behalf of the corresponding `fstream` (or `sstream`) type.

That is, if we have a function that takes an `ostream&`, we can call that function passing it an `ofstream` object, and similarly for `istream&` and `ifstream`.

See the following for an example, we want to read data from cin and save the data into a file called Out.txt:

```
Sales_data total;
ofstream out(argv[1]);
//read data from cin and save it onto total
if(read(cin, total)) {
  Sales_data trans;
  while(read(cin, trans)) {
    if(total.isbn() == trans.isbn()) {
      total.combine(trans);
    } else {
      //save data of total onto the file output stream
      print(out, total) << endl;
      total = trans;
    }
  }
  print(out, total) << endl;
}
else {
  std::cerr << "No data" << endl;
}
```

If we run it in shell using the following code:

```
prompt> .\a.exe Out.txt
```

The open and close Members

When we define an empty file stream object, we can subsequently associate that object with a file by calling `open`:

```
ifstream in(ifile); //construct an ifstreamand open the given file
ofstream out; //output file stream that is not associated with any file
out.open(ifile + ".copy"); //open the specified file
```

If a call to open fails, `failbit` is set. Because a call to `open` might fail, it is usually a good idea to verify that the open succeeded:

```
if(out) //check that the open succeeded
  //the open succeeded, so we can use the file
```

Once a file stream has been opened, it remains associated with the specified file.

Indeed, calling open on a file stream that is already open will fail and set `failbit`.

To associate a file stream with a different file, we must first close the existing file. Once the file is closed, we can open a new one:

```
in.close(); //close the file
in.open(ifile + "2"); //open another file
```

Automatic Construction and Destruction

Consider a program whose `main` function takes a list of files it should process. Such a program might have a look like the following:

```
//for each file passed to the program
for(auto p = argv + 1; p < argv + argc; ++p) {
  ifstream input(*p); //create input and open the file
  if(input) { //if the file is ok, "process" thsi file
    process(input);
  } else {
    cerr << "couldn't open: " + string(*p);
  }
}
```

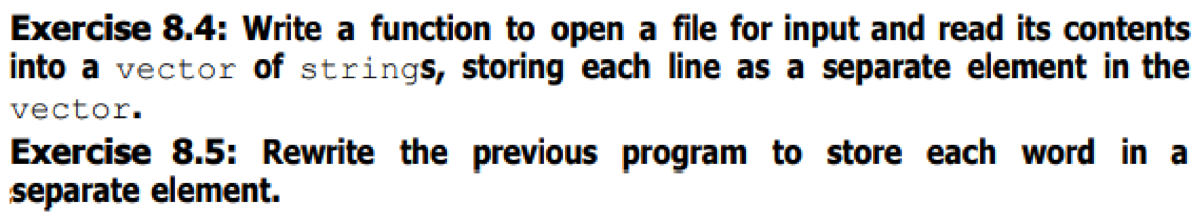
Each iteration constructs a new `ifstream` object named input and opens it to read the given file.

As usual, we check that the open succeeded. If so, we pass that file to a function that will read and process the input. If not, we print an error message and continue.

Because input is local to the while, it is created and destroyed on each iteration. When an fstream object goes out of scope, the file it is bound to is automatically closed. On the next iteration, input is created anew.

Note: When an `fstream` object is destroyed, close is called automatically.

Exercise



Exercise 8.4: Write a function to open a file for input and read its contents into a `vector` of `string`s, storing each line as a separate element in the `vector`.

Exercise 8.5: Rewrite the previous program to store each word in a separate element.

```
//打印vector中内容
void showVec(vector<string> vec) {
  for(auto str : vec)
    cout << str << endl;
}

std::istream &func(std::istream &is, vector<std::string> &vec) {
  string buffer;
  //从文件中读出一行内容
  //若想要每个element，则使用is >> buffer
  while(getline(is, buffer)) {
    vec.push_back(buffer);
  }
  showVec(vec);
  return is;
}

int main(int argc, char *argv[]) {
  std::ifstream input(argv[1]);
  if(!input)
    std::cerr << "Wrong input" << endl;

  vector<string> vec;
  func(input, vec);
  return 0;
}
```