



中國人民大學
RENMIN UNIVERSITY OF CHINA

程序设计荣誉课程

6 字符串与输入输出

授课教师：孙亚辉

本章内容

1. string

2. IO

1. string

C++98标准添加了string类，可以使用string类型的变量（对象）而不是字符数组来存储字符串。

- 头文件<string>
- string类定义隐藏了字符串的数组性质，使程序员能够像处理普通变量那样处理字符串
- string类使用起来比数组简单，同时提供了将字符串作为一种数据类型的表示方法

示例

```
1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4
5  int main()
6  {
7      char charr1[20];           // create an empty array
8      char charr2[20] = "jaguar"; // create an initialized array
9      string str1;               // create an empty string object
10     string str2 = "panther";    // create an initialized string
11     cout << "Enter a kind of feline: ";
12     cin >> charr1;
13
14     cout << "Enter another kind of feline: ";
15     cin >> str1; // use cin for input
16
17     cout << "Here are some felines:\n";
18     cout << charr1 << " " << charr2 << " "
19     << str1 << " " << str2 // use cout for output
20     << endl;
21     cout << "The third letter in " << charr2 << " is "
22     << charr2[2] << endl;
23     cout << "The third letter in " << str2 << " is "
24     << str2[2] << endl; // use array notation
25     return 0;
26 }
```

```
yahui@Yahui:/media/sf_VM$ g++ test.cpp
yahui@Yahui:/media/sf_VM$ ./a.out
Enter a kind of feline: ocelot
Enter another kind of feline: tiger
Here are some felines:
ocelot jaguar tiger panther
The third letter in jaguar is g
The third letter in panther is n
```

- 可以使用C-风格字符串来初始化string对象
- 可以使用cin来将键盘输入存储到string对象中
- 可以使用cout来显示string对象
- 可以使用数组表示法来访问存储在string对象中的字符

注意：通过using namespace std或using std::string引入string类

string的一些特点

- string对象和字符数组的主要区别是：可将string对象声明为简单变量，而不是数组，如
 - string str1;
 - string str2 = "panther";
- 程序能够自动处理string的大小
 - str1的声明创建一个长度为0的string对象，当程序将输入读取到str1中时，将自动调整str1的长度
 - **注意**：此处，长度是指其内部所存储的字符串的长度，而不是string类对象本身的大小
 - 示例（Ubuntu 18.04-x64）

```
1. string str1;  
2. cout << sizeof(str1) << " " << str1.length() << endl;  
3. cin >> str1; // 输入tiger  
4. cout << sizeof(str1) << " " << str1.length() << endl;
```

或size()

输出结果：

32 0
32 5

- **string**对象的大小与其内部储存的字符串的长度无关。

```
1. string str1;  
2. cout << sizeof(str1) << " " << str1.length() << endl;  
3. cin >> str1; // 输入ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890  
4. cout << sizeof(str1) << " " << str1.length() << endl;
```

- 输入字符串长度为36，输出结果为

```
32  0  
32  36
```

- 发生改变的只是内部字符串数据的大小，**string**对象本身的大小没有变化（**string**是一个类，其对象存储了指向字符串存储地址的指针等内容）。

- 使用string，程序员无需预先指定字符串的最大长度（如char charr1[20]），因此在处理具有未知大小的字符串时更方便。

```
1. int main() {
2.     char charr1[20]; // create an empty array
3.     string str1; // create an empty string object
4.     cout << "Enter a kind of feline: ";
5.     cin >> charr1;

6.     cout << "Enter another kind of feline: ";
7.     cin >> str1; // use cin for input

8.     cout << &charr1 << " " << &str1 << endl;
9.     cout << charr1 << " " << str1 << endl;
10.    return 0;
11. }
```

- 若分别输入IamAveryLongString1234567890和ThisCanBeAVeryLongStringABCDEFGH，会发生什么？

- 字符数组charr1限长20，但输入数据长度为28，发生了栈溢出，修改了栈上其他数据，导致运行错误

```
Enter a kind of feline: IamAveryLongString1234567890
Enter another kind of feline: ThisCanBeAVeryLongStringABCDEFGG
0x7ffffcbf708f0 0x7ffffcbf708d0
IamAveryLongString1234567890 ThisCanBeAVeryLongStringABCDEFGG
*** stack smashing detected ***: <unknown> terminated
Aborted (core dumped)
```

- 若去掉字符数组charr1，只保留str1，且提供一个足够长的输入，结果如下：
 - 未发生运行错误

```
Enter another kind of feline: 1234567890ABCDEFWFGHIJKLMNOPQRSTUVWXYZ0987654321
0x7ffffebd36320
1234567890ABCDEFWFGHIJKLMNOPQRSTUVWXYZ0987654321
```


字符串初始化

- 带初值的字符串string对象初始化：
 - `string str1("hello world");`
 - `string str2{"hello world"};`
 - `string str3 = "hello world";`
 - `string str4 = {"hello world"};`
 - `string str5 = ("hello world");`
- 以上5种形式均可，从汇编代码也能看出，五种初始化方式，调用的是相同的构造函数。

赋值、拼接和附加（1）

- 使用字符数组时，不能将一个数组赋给另一个数组，不能将两个字符数组简单的“+”起来形成一个新的字符数组

```
1. char charr1[] = "hello";  
2. char charr2[] = "world";  
3. char charr3[] = charr1;  
4. char charr4 = charr1 + charr2;
```

- 编译错误：

```
H.cpp: In function 'int main()':  
H.cpp:15:19: error: array must be initialized with a brace-enclosed initializer  
char charr3[6] = charr1;  
                  ^~~~~~  
H.cpp:16:23: error: invalid operands of types 'char [6]' and 'char [6]' to binary 'operator+'  
char charr4 = charr1 + charr2;  
              ~~~~~~^~~~~~
```

赋值、拼接和附加（2）

- 使用string类时
 - 可以将一个string对象赋给另一个string对象
 - 可以使用运算符“+”将两个string对象合并起来
 - 合并或附加的数据也可以是C-风格字符串（字符数组）
 - 还可以使用运算符“+=”将字符串附加到string对象的末尾

```
1. string str1("hello");
2. string str2{"world"};
3. char   charr1[] = "good";
4. string str3 = str1;
5. string str4 = str1 + str2;
6. string str5 = str3 + charr1;
7. str5 += " morning";
8. cout << str3 << " " << str4 << endl;
9. cout << str5 << endl;
```

- 输出结果 hello helloworld
 hellogood morning

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      string str1("hello");
8      string str2{"world"};
9      char charr1[] = "good";
10     string str3 = str1;
11     string str4 = str1 + str2;
12     string str5 = str3 + charr1;
13     str5 += " morning";
14     cout << str3 << " " << str4 << endl;
15     cout << str5 << endl;
16 }
```

```
yahui@Yahui:/media/sf_VM$ g++ test.cpp
yahui@Yahui:/media/sf_VM$ ./a.out
hello helloworld
hellogood morning
```

string类IO

- 使用`cin>>`可以将输入数据放入C-风格字符数组或`string`对象中，使用`cout<<`可以显示两种风格的字符串
- 使用`getline`每次读取一行而不是一个单词时，使用的句法不同

```
1. char   charr1[20];  
2. string str1;  
3. cin.getline(charr1, 20);  
4. std::getline(cin, str1);  
  
5. cout << charr1 << endl;  
6. cout << str1 << endl;
```

- 第3行：`getline`是`istream`类的一个成员函数，两个参数分别表示目标数组和最大接收长度
- 第4行：`getline`不是类方法，而是一个全局方法。以`cin`作为参数，指出到哪里去查找输入，字符串长度无需手动确定

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      char charr1[20];
8      string str1;
9      cin.getline(charr1, 20);
10     std::getline(cin, str1);
11
12     cout << charr1 << endl;
13     cout << str1 << endl;
14 }
```

```
yahui@Yahui:/media/sf_VM$ g++ test.cpp
yahui@Yahui:/media/sf_VM$ ./a.out
3rs
eqed
3rs
eqed
```

string类的一些其他方法

- **const char * c_str():** 返回string对象所存储数据的C-风格字符串指针，如
 - `const char *s = str1.c_str();`

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      string str1 = "tiger";
8      const char *s = str1.c_str();
9      cout << s << endl;
10 }
```

```
yahui@Yahui:/media/sf_VM$ g++ test.cpp
yahui@Yahui:/media/sf_VM$ ./a.out
tiger
```

string类的一些其他方法

- `bool empty() const`: 检查是否`size() == 0`
- `void clear()`: 删除字符串中所有的字符

```
1. string str1;  
2. string str2 = "hello";  
3. cout << str1.empty() << " " << str2.empty() << endl;  
4. str2.clear();  
5. cout << str1.empty() << " " << str2.empty() << endl;
```

— 输出结果

```
1 0  
1 1
```


字符串存取

- `string`类提供`[]`运算符和`at()`方法用于存取字符串的元素
 - **差别**：`at()`方法执行边界检查（更安全），而`[]`运算符不进行边界检查（更快）
 - 通过`[]`操作符或`at()`方法更改指定位置的元素

```
1. string str = "hello";  
2. str[1] = 'E';  
3. str.at(3) = 'X';  
4. cout << str << endl;
```

输出结果：
hE1Xo

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      string str = "hello";
8      cout << str[5] << endl;
9  }
```

可以运行

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      string str = "hello";
8      cout << str.at(5) << endl;
9  }
```

```
yahui@Yahui:/media/sf_VM$ g++ test.cpp
yahui@Yahui:/media/sf_VM$ ./a.out
terminate called after throwing an instance of 'std::out_of_range'
  what():  basic_string::at: __n (which is 5) >= this->size() (which is 5)
)
Aborted (core dumped)
```

at()方法执行边界检查（更安全），
而[]运算符不进行边界检查（更快）

front()和back()

- C++11标准增加了front()和back()方法，分别用于访问string的第一个和最后一个元素

```
1. string str = "hello";  
2. cout << str.front() << " " << str.back() << endl;  
3. str.front() = 'H';  
4. str.back() = 'X';  
5. cout << str << endl;
```

- 输出结果 h o
 HellX

子字符串

- `substr(pos, n)`返回一个从`pos`开始、复制`n`个字符（或到字符串尾部）的一个字符串

```
1. string str = "hello";  
2. cout << str.substr(1) << endl;  
3. cout << str.substr(1, 3) << endl;
```

- 输出结果 `ello`
 `ell`

字符串搜索

`string`类提供了多种搜索函数

- `find(str, pos = 0)`: 返回当前对象中从`pos`位置开始，第一次出现`str`的位置；如果搜索失败，则返回`string::npos`
- `rfind(str, pos = npos)`: 返回当前对象中在`pos`之前最后一次出现`str`的位置；搜索失败则返回`string::npos`
- `find_first_of`: 搜索给定字符串中的字符首次出现的位置
- `find_last_of`: 搜索给定字符串中的字符出现的最后位置
- `find_first_not_of`: 搜索第一个不位于子字符串中的字符
- `find_last_not_of`: 搜索的是最后一个没有在字符串中出现的字符

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      string str = "That is a funny hat."; // string长度20, char[21],因为最后是\0
8
9      cout << str.find("ha") << endl;
10     cout << str.find("ha", 2) << endl;
11     if (string::npos == str.find('A')) cout << "Fail" << endl << endl;
12
13     cout << str.rfind("ha") << endl;
14     cout << str.rfind("ha", 15) << endl << endl;
15
16     cout << str.find_first_of("fat") << endl;
17     cout << str.find_last_of("fat") << endl;
18     cout << str.find_first_not_of("This") << endl;
19     cout << str.find_last_not_of("That.") << endl;
20
21     return 0;
22 }

```

```

yahui@Yahui:/media/sf_VM$ g++ test.cpp
yahui@Yahui:/media/sf_VM$ ./a.out

```

```

1
16
Fail

```

```

16
1
2
18
2
15

```

字符串解析1

```
test.cpp
test.cpp
1  #include<vector>
2  #include<string>
3  #include<iostream>
4  using namespace std;
5
6  std::vector<std::string> parse_string(std::string parse_target, std::string delimiter) {
7      std::vector<std::string> Parsed_content;
8      size_t pos = 0;
9      std::string token;
10     while ((pos = parse_target.find(delimiter)) != std::string::npos) {
11         token = parse_target.substr(0, pos);
12         Parsed_content.push_back(token);
13         parse_target.erase(0, pos + delimiter.length());
14     }
15     Parsed_content.push_back(parse_target);
16
17     return Parsed_content;
18 }
19
20 int main()
21 {
22     std::string s = "sfgdssddd";
23     auto xx = parse_string(s, "gd");
24     std::cout << xx[0] << "|" << xx[1] << std::endl;
25 }
```

字符串解析2

```
23  std::vector<std::string> parse_substring_between_pairs_of_delimiters
24  (std::string& parse_target, std::string delimiter1, std::string delimiter2) {
25
26      /*this pair of delimiters must be different*/
27      std::vector<std::string> results;
28      std::vector<std::string> Parsed_content1 = parse_string(parse_target, delimiter1);
29      for (int i = 0; i < Parsed_content1.size(); i++) {
30          if (Parsed_content1[i].find(delimiter2) != std::string::npos)
31          {
32              std::vector<std::string> Parsed_content2 = parse_string(Parsed_content1[i], delimiter2);
33              results.push_back(Parsed_content2[0]);
34          }
35      }
36      return results;
37  }
38
39  int main()
40  {
41      std::string s = "s(fr)dgfx(sgd)";
42      auto xx = parse_substring_between_pairs_of_delimiters(s, "(", ")");
43      std::cout << xx[0] << "|" << xx[1] << std::endl;
44  }
```


字符串比较

- `string`类提供了`compare`函数进行字符串比较，返回`int`值
- 另外，重载比较运算符`==`、`<`、`<=`、`>`、`>=`、`!=`，可用于在`string`对象与`string`对象、`string`对象与C-风格字符串、C-风格字符串与`string`对象之间进行比较
- `s1 < s2` : length of `s1` is shorter than `s2` or first mismatched character is smaller (ASCLL值).
- `s1 > s2` : length of `s1` is longer than `s2` or first mismatched character is larger.
- `<=` and `>=` have almost same implementation with additional feature of being equal as well.

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      string str1 = "hello";
8      string str2 = "world";
9
10     cout << str1.compare(str2) << " " << str2.compare(str1) << " " << str1.compare(str1) << endl;
11     cout << (str1 < str2) << " " << (str1 < str1 + "s") << " " << (str1 != str1) << " " << (str1 != str2) << endl;
12
13
14     return 0;
15 }

```

```

yahui@Yahui:/media/sf_VM$ g++ test.cpp
yahui@Yahui:/media/sf_VM$ ./a.out
-1 1 0
1 1 0 1

```

子串替换

- `string`类提供了一系列`replace`方法，用于将当前对象中指定起始/结束位置的区间替换为给定的内容
 - 以下以一个方法为例：将字符串`str`中第7位开始，长度为5的子串（`right`）替换为`left`

```
1. string str = "Take a right turn at Main Street";  
2. cout << str.replace(7, 5, "left") << endl;
```

- 输出结果

Take a left turn at Main Street

本章内容

1. string

2. IO

2.10

- C++程序把输入（Input）和输出（Output）看作字节流
 - 输入时，程序从输入流中抽取字节
 - 输出时，程序将字节插入输出流
- 对于文本输入/输出流，每个字节代表一个字符；对于二进制输入/输出流，每个字节代表一个二进制数值
- “流”充当了程序和流源或流目标之间的桥梁，使C++程序可以以相同的方式对待来自键盘的输入和来自文件的输入
- 以输入为例，管理输入包含两步
 - 将流与输入去向的程序关联起来
 - 将流与文件（流的来源）关联起来
- 读写数据时，先将数据放置于缓冲区，直至缓冲区数组发生溢出或强制执行刷新操作，才会导致缓冲区中的数据被写出到目的地或读入程序。

使用cout进行输出

- 我们常使用cout对象结合<<操作符执行输出
 - cout对象是ostream类的一个实例化对象
- ostream类除了<<操作符外，还提供了put()和write()方法，分别用于显示字符和字符串

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      cout.put('w').put('U');
8      cout.write("hello", 5) << endl; // 参数为字符串及串长度
9  }
```

```
yahui@Yahui:/media/sf_VM$ g++ test.cpp
yahui@Yahui:/media/sf_VM$ ./a.out
wUhello
```

- 刷新缓冲区

- 由于ostream类对cout对象处理的输出会进行缓冲，所以输出不会立即发送到目标地址，而是存储在缓冲区中，直到缓冲区填满。
- 多数C++实现会在输入即将发生时刷新缓冲区
- 可以通过flush或endl控制符刷新缓冲区（后者插入一个换行符）

```
1. cout << "Hello, good-looking!" << flush;  
2. cout << "Wait just a moment, please." << endl;
```

- 也可以直接调用全局函数flush()来刷新缓冲区

```
1. flush(cout);
```

ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);

cin cout先把要输入输出的东西存入缓冲区，再输入输出，导致效率低，而上述语句可以来打消iostream的输入输出缓存，可以节省许多时间，使其效率与scanf与printf相差无几。

```
test.cpp ×
test.cpp > main()
1  #include <iostream>
2  #include <chrono>
3  using namespace std;
4
5  int main()
6  {
7      ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
8      auto begin = std::chrono::high_resolution_clock::now();
9      for (int i = 0; i < 1e3; i++) {
10         cout << i;
11     }
12     auto end = std::chrono::high_resolution_clock::now();
13     double runtime = std::chrono::duration_cast<std::chrono::nanoseconds>(end - begin).count() / 1e6;
14     cout << " runtime: " << runtime << "ms" << endl;
15 }
```


用cout进行格式化

- 有时我们需要对输出数据进行格式化，比如使用十六进制输出整数、保留特定位数的小数
- 在C语言中，使用printf输出时，可以使用格式化字符对输出进行控制，如

```
1. printf(“%x %.2f”, i32, f32); // 示例: 2d 1.23
```

- 在C++中使用cout输出时，可通过控制符调整显示效果
 - 使用dec、hex和oct控制符，控制整数以十进制、十六进制还是八进制显示

```
1. cout << std::hex << i32 << endl;
```

- 注意：完成设置后，程序后续将持续使用该设置进行输出，直到格式状态被设置为其他选项为止。如

```
1. cout << std::hex << i32 << endl;  
2. cout << i32 << endl;
```

第2行虽然没有明确设置hex控制符，但是延续了第1行的输出格式

- 调整字段宽度

- 通过ostream类的成员函数width()可以调整输出内容的宽度
- 注意：与前一页的格式空字符不同，width()方法只影响将显示的下一个项目，然后字段宽度将恢复为默认值

```
1. cout.width(6);  
2. cout << 45 << "-" << 45 << "-" << 45 << endl;
```

- 输出结果

45-45-45

4个空格

- 等价于printf输出形式

```
1. printf("%6d", i32);
```

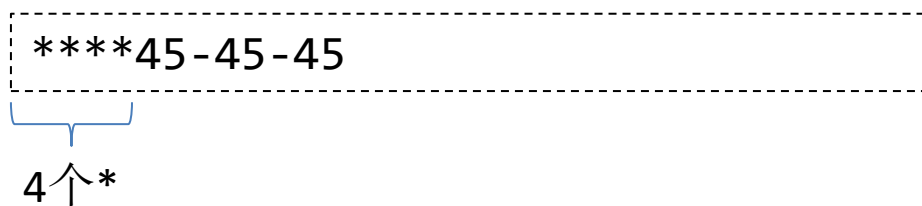
- 填充字符

- 通过ostream类的成员函数fill()可以用指定字符来填充字段中未被使用的部分
- 默认使用空格填充

```
1. cout.fill('*');  
2. cout.width(6);  
3. cout << 45 << "-" << 45 << "-" << 45 << endl;
```

- 输出结果

****45-45-45



4个*

- 新的填充字符将一直有效，直到更改它为止

- 设置浮点数的显示精度
 - 默认模式下，精度指的是显示的总位数
 - 通过ostream类的成员函数precision()可以用指定显示进度
 - 与fill()类似，新的精度设置将一直有效

```
1. cout << f32 << endl;  
2. cout.precision(2);  
3. cout << f32 << endl;
```

- 输出结果

```
1.23456  
1.2
```

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  /*https://www.cplusplus.com/reference/string/string/compare/*/
6
7  int main()
8  {
9      printf("%x %.2f\n", 425, 21.1);
10
11     cout << std::hex << 425 << endl;
12     cout << 425 << endl;
13
14     cout << std::dec;
15     cout.precision(3);
16     cout.fill('*');
17     cout.width(6);
18     cout << 425 << "-" << 425.1 << "-" << 425 << endl;
19     printf("%6d\n", 425);
20
21     return 0;
22 }

```

```

yahui@Yahui:/media/sf_VM$ g++ test.cpp
yahui@Yahui:/media/sf_VM$ ./a.out
1a9 21.10
1a9
1a9
***425-425-425
    425

```

setf

- **setf**成员函数可用来控制数据的显示格式
- 相应的格式常量在**ios_base**类中定义，下标列举了其中一部分

常量	含义
boolalpha	输入输出的bool值为true和false
showbase	对于输出，使用C++基数前缀（0、0x）
showpoint	显示末尾的小数点
uppercase	对于16进制输出，使用大写字母，E表示法
showpos	在（十进制）正数前加上“+”

```
1. cout << b << " " << i32 << " " << f32 << endl;
2. cout.setf(ios_base::boolalpha);
3. cout.setf(ios_base::showpos);
4. cout << b << " " << i32 << " " << f32 << endl;

5. cout.setf(ios_base::showbase);
6. cout.setf(ios_base::uppercase);
7. cout << b << " " << std::hex << i32 << " " << f32 << endl;
```

输出结果:

1 45 1.23456

true +45 +1.23456

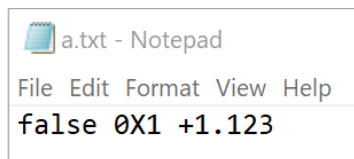
true 0X2D +1.23456

文件输入和输出

- C++的I/O类软件包处理文件输入和输出的方式与处理标准输入和输出的方式非常类似
- 写入文件：创建一个ofstream对象，并使用ostream方法，如<<操作符或write()等方法
- 读取文件：创建一个ifstream对象，并使用istream方法，如>>操作符或get()等方法
- C++在头文件<fstream>中定义了多个类，包括用于文件输入的ifstream类和用于文件输出的ofstream类
 - 这些类都是从头文件<iostream>中的类派生而来的，它们的对象可以使用前面介绍过的方法

示例

- 创建一个ofstream对象来管理输出流
- 将该对象与特定的文件关联起来
- 以使用cout的方式使用该对象



```
yahui@Yahui:/media/sf_VM$ g++ test.cpp
yahui@Yahui:/media/sf_VM$ ./a.out
false 0X1 +1.123
```

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  using namespace std;
5
6  int main()
7  {
8      bool b;
9      int i32 = 1;
10     float f32 = 1.123;
11     ofstream fout;
12     fout.open("a.txt");
13     fout.setf(ios_base::boolalpha);
14     fout.setf(ios_base::showpos);
15     fout.setf(ios_base::showbase);
16     fout.setf(ios_base::uppercase);
17     fout << b << " " << std::hex << i32 << " " << f32 << endl;
18     fout.close();
19
20     ifstream fin("a.txt");
21     string str;
22     std::getline(fin, str);
23     fin.close();
24     cout << str << endl;
25     return 0;
26 }
```

检查文件是否成功打开

- 可使用如下方式检查试图打开文件时是否成功

```
1. ifstream fin("a.txt");  
2. if (fin.fail()) return -1;  
3. if (!fin) return -1;  
4. if (!fin.is_open()) return -1;
```

- 同样地，这些方式也能被用在ofstream对象上，检查输出的目标文件是否成功打开

一行一行地读取文件

```
test.cpp  X
test.cpp
1  #pragma once
2  #include <fstream>
3  #include <string>
4  #include <iostream>
5
6  void read_file_line_by_line(std::string file_name) {
7      std::string line_content;
8      std::ifstream myfile(file_name); // open the file
9      if (myfile.is_open()) { // if the file is opened successfully
10         int count = 0;
11         while (getline(myfile, line_content)) // read file line by line
12         {
13             count++;
14             std::cout << line_content << std::endl;
15         }
16         myfile.close(); //close the file
17         std::cout << "Total Line Num: " << count << std::endl;
18     }
19     else {
20         std::cout << "Unable to open file " << file_name << std::endl
21             << "Please check the file location or file name." << std::endl; // throw an error message
22         getchar(); // keep the console window
23         exit(1); // end the program
24     }
25 }
26
27 int main(){
28     read_file_line_by_line("test.cpp");
29 }
```

文件模式

- 文件模式描述的是文件将被如何使用：读、写、追加
- 将流与文件关联时，可提供指定文件模式的第二个参数

```
1. ifstream fin("a.txt", mode1);  
2. ofstream fout;  
3. fout.open("b.txt", mode2);
```

- 下表列出了C++文件I/O相应的模式

Constant	Meaning
<code>ios_base::in</code>	Open file for reading.
<code>ios_base::out</code>	Open file for writing.
<code>ios_base::ate</code>	Seek to end-of-file upon opening file.
<code>ios_base::app</code>	Append to end-of-file.
<code>ios_base::trunc</code>	Truncate file if it exists.
<code>ios_base::binary</code>	Binary file.

续写文件:

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  using namespace std;
5
6  int main()
7  {
8      bool b;
9      int i32 = 1;
10     float f32 = 1.123;
11     ofstream fout;
12     fout.open("a.txt");
13     fout.setf(ios_base::boolalpha);
14     fout.setf(ios_base::showpos);
15     fout.setf(ios_base::showbase);
16     fout.setf(ios_base::uppercase);
17     fout << b << " " << std::hex << i32 << " " << f32 << endl;
18     fout.close();
19
20     fout.open("a.txt", ios::out|ios::app);
21     fout << b << " " << std::hex << i32 << " " << f32 << endl;
22     fout.close();
23 }
```

 a.txt - Notepad

File Edit Format View Help

false 0X1 +1.123

false 0X1 +1.123

读写二进制文件（用处：读写二进制文件比读写非二进制文件快很多）：

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  #include <vector>
5  using namespace std;
6
7  void binary_save_vector_of_vectors(std::string path, const vector<vector<int>> &myVector)
8  {
9      std::ofstream FILE(path, std::ios::out | std::ofstream::binary);
10
11     // Store size of the outer vector
12     int s1 = myVector.size();
13     FILE.write(reinterpret_cast<const char *>(&s1), sizeof(s1));
14
15     // Now write each vector one by one
16     for (auto &v : myVector)
17     {
18         // Store its size
19         int size = v.size();
20         FILE.write(reinterpret_cast<const char *>(&size), sizeof(size));
21
22         // Store its contents
23         FILE.write(reinterpret_cast<const char *>(&v[0]), v.size() * sizeof(int));
24     }
25     FILE.close();
26 }
```

```

28 void binary_read_vector_of_vectors(std::string path, vector<vector<int>> &myVector)
29 {
30     ifstream FILE(path, std::ios::in | std::ifstream::binary);
31
32     int size = 0;
33     FILE.read(reinterpret_cast<char *>(&size), sizeof(size));
34     if (!FILE)
35     {
36         std::cout << "Unable to open file " << path << endl
37         << "Please check the file location or file name." << endl; // throw an error message
38         exit(1); // end the program
39     }
40     myVector.resize(size);
41     for (int n = 0; n < size; ++n)
42     {
43         int size2 = 0;
44         FILE.read(reinterpret_cast<char *>(&size2), sizeof(size2));
45         int f;
46         for (int k = 0; k < size2; ++k)
47         {
48             FILE.read(reinterpret_cast<char *>(&f), sizeof(f));
49             myVector[n].push_back(f);
50         }
51     }
52 }

```

```

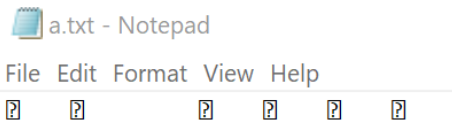
54  int main()
55  {
56      vector<vector<int>>> x(2), y;
57      x[0].push_back(0);
58      x[0].push_back(1);
59      x[1].push_back(2);
60      x[1].push_back(3);
61
62      binary_save_vector_of_vectors("a.txt", x);
63      binary_read_vector_of_vectors("a.txt", y);
64
65      cout << y[1][1] << endl;
66  }

```

```

yahui@Yahui:/media/sf_VM$ g++ test.cpp
yahui@Yahui:/media/sf_VM$ ./a.out
3

```



读写二进制文件较普通读写文件快不少（大数据处理），
但缺点是保存的数据文件不具有可读性

随机存取

- 随机存取指在文件的任何位置进行存取。
- 以下将以读写模式打开文件——创建一个**fstream**对象
 - 从**iostream**类派生而来，继承了两个缓冲区，一个用于输入，一个用于输出，并能同步这两个缓冲区的处理
 - 即：当程序读写文件时，它将协调地移动输入缓冲区中的输入指针和输出缓冲区中的输出指针
- **fstream**类提供两个方法**seekg**（**get**）和**seekp**（**put**），分别用于移动输入缓冲区指针和输出缓冲区指针

示例

- 以前文创建的a.txt为例

```
1. char charr[5];
2. fstream finout("a.txt", ios_base::in | ios_base::out);
3. string str;
4. //获取文件现有内容（一行），并输出到屏幕
5. std::getline(finout, str);
6. cout << str << endl;
7. // 从文件头部移动5个字节的位置，并写入“ABCD\0”
8. finout.seekp(5, ios_base::beg); // ios_base::beg: 文件开始的位置
9. finout << "ABCD\0";
10. // 从文件头部移动5个字节的位置，并读入最多5个字节到charr中，并输入到屏幕
11. finout.seekg(5, ios_base::beg);
12. finout >> charr; // 遇到\0停止读入
13. cout << charr << endl;
14. // 获取文件现有内容（一行），并输出到屏幕
15. finout.seekg(0, ios_base::beg);
16. std::getline(finout, str);
17. cout << str << endl;
```

输出结果:

true 0X2D +1.23456

ABCD

true ABCD +1.23456

<pre> 1 #include <iostream> 2 #include <fstream> 3 #include <string> 4 using namespace std; 5 6 int main() 7 { 8 string file_name = "a.txt"; 9 ofstream myfile(file_name, ios_base::out); 10 myfile << "this is the first line." << endl; 11 myfile << "this is the second line." << endl; 12 myfile.close(); 13 14 fstream finout(file_name, ios_base::in ios_base::out); 15 string line_content; 16 int line_id = 1; 17 // 每次使用getline, 读指针便移到下一行开头, 如果没有读取则return FALSE 18 while (getline(finout, line_content)) // read file line by line 19 { 20 if (line_id == 1) 21 { 22 cout << line_content << endl; //获取文件现有内容 (一行), 并输出到屏幕 23 finout.seekp(5, ios_base::beg); //从文件头开始偏移量为5的位置 24 finout << "IS"; // 写指针写新内容 (覆盖旧内容) 25 26 char str[8]; 27 finout.seekg(5, ios_base::beg); 28 finout >> str; // 从文件头部移动5个字节的位置, 并读入最多8个字节到charr中, 并输入到屏幕 (遇到empty \0 char停止) 29 cout << " " << str << " " << endl; 30 31 // 获取文件现有内容 (一行), 并输出到屏幕 32 finout.seekg(0, ios_base::beg); 33 getline(finout, line_content); 34 cout << line_content << endl; 35 } </pre>	<pre> yahui@Yahui:/media/sf_VM\$ g++ test.cpp yahui@Yahui:/media/sf_VM\$./a.out this is the first line. IS this IS the first line. this is the second line. a.txt is opened successfully Print a.txt: this IS the first line. this IS the second line. Print a.txt over </pre>
---	---

```

37     if (line_id == 2)
38     {
39         cout << line_content << endl;    //获取文件现有内容（一行），并输出到屏幕
40         finout.seekp(-25, ios_base::cur);
41         // 从文件尾(ios_base::cur当前位置)开始偏移量为-25的位置 (Linux末尾 \n; VC编译器末尾 \0 \n, 应该是-26);
42         finout << "this IS"; // 写指针写新内容（覆盖旧内容）
43     }
44
45     line_id++;
46 }
47 finout.close(); //close the file
48
49 // print file
50 if (1)
51 {
52     ifstream myfile2(file_name);
53     if (myfile2.is_open() == true)
54     {
55         cout << file_name << " is opened successfully" << endl;
56         cout << "Print " << file_name << ":" << endl;
57         string line_content;
58         while (getline(myfile2, line_content)) // read file line by line
59         {
60             cout << line_content << endl;
61         }
62         cout << "Print " << file_name << " over" << endl;
63     }
64 }
65
66
67
68 return 0;
69 }

```

```

yahui@Yahui:/media/sf_VM$ g++ test.cpp
yahui@Yahui:/media/sf_VM$ ./a.out
this is the first line.
|IS|
this IS the first line.
this is the second line.
a.txt is opened successfully
Print a.txt:
this IS the first line.
this IS the second line.
Print a.txt over

```

内核格式化

- C++库提供了sstream族，使用相同的接口提供程序和string对象之间的I/O
 - 可以将用于cout的ostream方法将格式化信息写入到string对象中，并使用istream方法来读取string对象中的信息
- 读取string对象中的格式化信息或将格式化信息写入string对象中被称为**内核格式化**（incore formatting）
- 头文件<sstream>定义了一个ostringstream类
 - 创建一个ostringstream对象，可以将信息写入其中，它将存储这些信息
 - 可以将可用于cout的方法用于ostringstream对象
 - 格式化文本进入缓冲区，在需要的情况下将动态分配内存以增大缓冲区。成员函数str()返回相应的字符串对象

示例

```
1. ostream outstr;  
2. double price = 380.0;  
3. char * ps = " for a copy of the ISO/EIC C++ standard!";  
  
4. outstr.precision(2);  
5. outstr << std::fixed; // 表示浮点输出应该以固定点或小数点表示法显示  
6. outstr << "Pay only CHF " << price << ps << endl;  
  
7. string str = outstr.str();  
8. cout << str << endl;
```

- 输出结果

Pay only CHF 380.00 for a copy of the ISO/EIC C++ standard!

- 同样地，可以创建 `istringstream`，使用 `istream` 类似的方法读取数据
 - `istringstream` 对象可以使用 `string` 对象进行初始化
 - 以上页最后的 `str` 为例

```
1. istringstream instr(str);  
2. string istr;  
3. std::getline(instr, istr);  
4. cout << istr << endl;
```

- 输出结果

Pay only CHF 380.00 for a copy of the ISO/EIC C++ standard!

```

1  #include <iostream>
2  #include <fstream>
3  #include <sstream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9      ostringstream outstr;
10     double price = 380.0;
11     char *ps = (char *)" for a copy of the ISO/EIC C++ standard!";
12     outstr.precision(2);
13     outstr << std::fixed;
14     outstr << "Pay only CHF " << price << ps << endl;
15     string str = outstr.str();
16     cout << str << endl;
17
18     istreamstringstream instr(str);
19     string istr;
20     std::getline(instr, istr);
21     cout << istr << endl;
22
23     return 0;
24 }

```

```

yahui@Yahui:/media/sf_VM$ g++ test.cpp
yahui@Yahui:/media/sf_VM$ ./a.out
Pay only CHF 380.00 for a copy of the ISO/EIC C++ standard!

Pay only CHF 380.00 for a copy of the ISO/EIC C++ standard!

```


作业

- YOJ-618. string与IO (1)
- YOJ-619. string与IO (2)
- YOJ-620. string与IO (3)
- YOJ-621. string与IO (4)