

目录

- 基本 I/O
- 格式化输出
- 字符串
- 字符串 I/O
- 命名空间

目录

- 基本 I/O
- 格式化输出
- 字符串
- 字符串 I/O
- 命名空间

C vs. C++ - 输出

```
#include <stdio.h>
```

```
int main() {  
    printf("hello, world\n");  
}
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main() {  
    cout << "hello, world" << endl;  
}
```

C vs. C++ - 输入

```
#include <stdio.h>
```

```
int main() {  
    int a, b;  
    scanf("%d%d", &a, &b);  
    printf("%d\n", a + b);  
}
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main() {  
    int a, b;  
    cin >> a >> b;  
    cout << (a + b) << endl;  
}
```

<iostream>

- C++ 标准输入输出定义在头文件 <iostream> 中，使用输入输出流库需要包含此头文件
- 标准库中有 4 个 I/O 相关对象：
 - 处理输入的 istream 对象 cin,
 - 处理输出的 ostream 对象 cout
 - 另外两个 ostream 对象 cerr 和 clog
- 也可以通过引入头文件 <cstdio> 或 <stdio.h> 使用 printf 和 scanf

标准输入流 cin (1)

- 流提取符 >>, 以空白字符或输入结束字符为终止
 - 常见的“空白字符”包括空格、换行 \n、回车 \r 和制表符 \t
- 输入结束 (End-Of-File, EOF) 字符: 在 Windows 的命令行中用 Ctrl + Z 输入, 在 Linux 的命令行中用 Ctrl + D 输入, 在 MacOS 的命令行中用 Command + D 输入

标准输入流 cin (2)

- 读入一个字符

```
char ch;  
cin.get(ch);
```

- 放回一个字符，可能会有问题，不推荐使用

```
cin.unget();
```

标准输入流 cin (3)

- 删除连续的空白字符

```
cin >> std::ws;
```

- 忽略从当前位置开始到 \n（包括）的字符

```
cin.ignore(  
    std::numeric_limits<std::streamsize>::max(),  
    '\n'  
);
```


标准输入流 cin (4) – 示例 (1)

- 输入 10 个数字

```
int nums[10];  
for (int i = 0; i < 10; i++) {  
    cin >> nums[i];  
}
```

- 输入未知个数的数字并求和

```
int sum = 0;  
while (cin >> n) {  
    sum += n;  
}
```

>> 实际上会转换成函数 `operator>>` 的调用，函数的返回值为 `istream` 对象，而 `istream` 对象可以转换为 `bool` 值，因而可以出现在 `while` 的条件中

标准输入流 cin (4) – 示例 (2)

- 读取未知个数的字符

```
char ch;  
while (cin.get(ch)) {  
    // do something ...  
}
```

get 的返回值也为 istream 对象

标准输入流 cin (4) – 示例 (3)

- 读取输入的两个坐标，输入格式为 (0,0),(1,1)

```
char c;  
int x1, y1, x2, y2;  
cin >> c >> x1 >> c >> y1 >> c >> c  
    >> c >> x2 >> c >> y2 >> c;
```

标准输出流 cout

- 流插入符 <<
- `std::endl` 相当于换行符 '\n'

C++ I/O 方式的优点 (1)

- 类型安全

- printf 和 scanf 等函数通过格式化占位符和可变参数实现，无法检查参数的类型和个数，尽管可以通过 -Werror* 标志让编译器检查，但并不优雅，C++ 通过函数重载解决了参数类型的问题，通过 >> 和 << 返回 istream 和 ostream 对象从而允许级联调用解决了可变参数的问题

- 更加易用

- 保障类型安全，减少心智负担
- 引用代替指针

*: 此标志可用于 GCC 和 Clang

C++ I/O 方式的优点 (2)

```
#include <cstdio>

int main() {
    double pi = 3.14;
    // %d 用于输出 int
    std::printf("%d\n", pi);
}
```

输出 -855011016

C++11



arithmetic types (1)

```
istream& operator>> (bool& val);
istream& operator>> (short& val);
istream& operator>> (unsigned short& val);
istream& operator>> (int& val);
istream& operator>> (unsigned int& val);
istream& operator>> (long& val);
istream& operator>> (unsigned long& val);
istream& operator>> (long long& val);
istream& operator>> (unsigned long long& val);
istream& operator>> (float& val);
istream& operator>> (double& val);
istream& operator>> (long double& val);
istream& operator>> (void*& val);
```

C++ I/O 方式的缺点

- 使字符串支离破碎

```
cout << n_apples << " apple(s), "  
      << n_peaches << " peach(es)" << endl;
```

- 相对于 printf 和 scanf 性能稍差

```
iostream version:          21.9 seconds  
scanf version:             6.8 seconds  
iostream with sync_with_stdio(false): 5.5 seconds
```

- 见 <https://stackoverflow.com/a/12762166>

目录

- 基本 I/O
- 格式化输出
- 字符串
- 字符串 I/O
- 命名空间

cout 格式化输出 (1)

- 需要包含头文件 `<iomanip>`

cout 格式化输出 (2) – 示例 (1)

- 输出不同进制

```
cout << showbase  
      << hex << 26 << ' ' // 十六进制  
      << oct << 26;        // 八进制  
// 输出: 0x1a 032
```

- 浮点数输出指定精度

```
cout << setprecision(5) << 3.1415926535;  
// 输出: 3.1416
```

cout 格式化输出 (2) – 示例 (2)

- 输出指定宽度、右对齐

```
cout << "" << setw(6) << right << 10 << "";  
// 输出 '      10'
```

- 输出年月日

```
int year = 2021, month = 3, day = 26;  
cout << year << '-'  
      << setw(2) << setfill('0') << month << '-'  
      << setw(2) << setfill('0') << day;  
// 输出: 2021-03-26
```

目录

- 基本 I/O
- 格式化输出
- 字符串
- 字符串 I/O
- 命名空间

<string>

- 使用 std::string 类需要包含头文件 <string>
- 跟 Java 不同，string 是可以修改内容的

```
#include <iostream>
#include <string>

using namespace std;

int main() {
    string s = "hello, world";
    s[0] = 'H'; s[7] = 'W';
    cout << s << endl;
}
```

string 的相关操作 (1) – I/O

- 读入，以空白字符或 EOF 作为结束标志

```
cin >> s;
```

- 读入一行，以换行符（默认）或指定的字符（称为 delimiter）作为结束标志，delimiter 会被读取但不会出现在 s 中

```
getline(cin, s);           // 以换行符为结束标志  
getline(cin, s, ',');       // 以 , 为结束标志
```

string 的相关操作 (2) – I/O

```
#include <iostream>
#include <string>

using namespace std;

int main() {
    cout << "Enter: ";
    string s1, s2, s3;
    cin >> s1 >> s2 >> s3;
    cout << "'" << s1 << "\", "
         << "'" << s2 << "\", "
         << "'" << s3 << "'" << endl;
}
```

```
Enter:      Magic C++   Wow!
"Magic", "C++", "Wow!"
```

```
#include <iostream>
#include <string>

using namespace std;

int main() {
    string s1, s2, s3;
    getline(cin, s1);
    getline(cin, s2, ',');
    getline(cin, s3);
    cout << "'" << s1 << "'" << endl;
    cout << "'" << s2 << "'" << endl;
    cout << "'" << s3 << "'" << endl;
}
```

```
hello, world
    indented line,      many blanks
"hello, world"
"    indented line"
"    many blanks      "
```

注意！

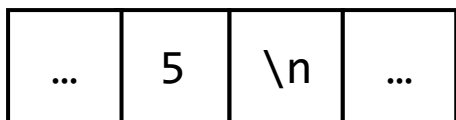
```
#include <iostream>
#include <string>

using namespace std;

int main() {
    int n;
    string line;
    cin >> n;
    getline(cin, line);
    cout << "n = " << n << endl;
    cout << "line = \"" << line << "\"\" << endl;
}
```

输入 5 然后回车：

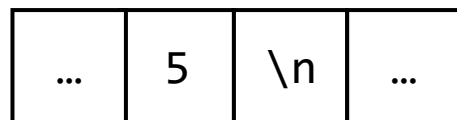
```
5
n = 5
line = ""
```

输入缓冲区

```
cin >> n;  
getline(cin, line);
```

1

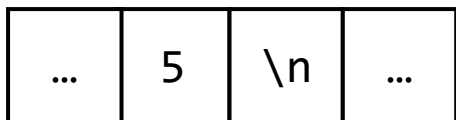


输入缓冲区

遇到 \n, 结束!

```
cin >> n; // n == 5  
getline(cin, line);
```

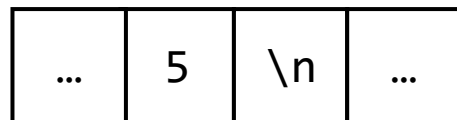
2



输入缓冲区

```
cin >> n;  
getline(cin, line);
```

3



输入缓冲区

再次遇到 \n, 结束!

```
cin >> n;  
getline(cin, line);
```

4

注意 getline 默认使用 \n 作为结束标志

如何解决？

- 使用 `std::ws`!

```
int main() {  
    int n;  
    string line;  
    cin >> n;  
    cin >> ws;  
    getline(cin, line);  
    cout << "n = " << n << endl;  
    cout << "line = \"" << line << "\"" << endl;  
}
```

```
5  
hello, world  
n = 5  
line = "hello, world"
```

string 的相关操作 (3) – 长度

- 获取字符串长度（以字节为单位）： `str.size()` 和 `str.length()`，含义相同
- `str.empty()` 判断 `str` 是否为空字符串 ""

string 的相关操作 (4) – 获取 char

- `str[index]`
 - $0 \leq \text{index} \leq \text{str.length}()$
 - `index == str.length()` 返回末尾的 `\0`, 不应该修改!
- `str.at(int index)`
 - $0 \leq \text{index} < \text{str.length}()$
- `str.front()` 获取第一个字符
- `str.back()` 获取最后一个非 `\0` 字符



```
string str = "12345678";  
cout << str[2] << endl;  
char c = str[5];  
cout << c << endl;  
cout << str.at(3) << endl;
```

```
3  
6  
4  
请按任意键继续. . .
```

string 的相关操作 (5) – 连接

- `s1 = s2 + s3`
- `s1.append(s2)` 或 `s1 += s2`

string 的相关操作 (6) – 其他

- 子串
 - `string s2 = s.substr(pos, n);` // **与 Java 不同:** 从 pos 开始取 n 个 char 组成新的字符串
- 比较
 - `<`、`<=`、`>`、`>=`、`==`、`!=`
 - `s1.compare(s2)` 相等时返回 0; `s1 < s2` 时返回 -1; `s1 > s2` 时返回 1

string 的相关操作 (7) – 与数值互转

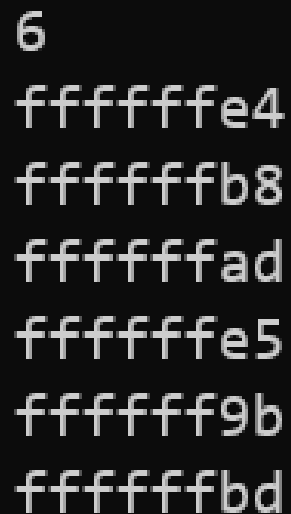
- 字符串转换为 int
 - `int v = std::stoi(str);`
- 转换为 long 使用 `stol`、转换为 long long 使用 `stoll`、转换为 float 使用 `stof`、转换为 double 使用 `stod`
- 数值（整型、浮点型）转换为字符串使用 `to_string`
 - `string s = std::to_string(42);`

注意！

- 跟 Java 不同，C++ 的 string 几乎是一个字节容器

字节容器

```
// 使用 Linux 上的 Vim 编写
// 如果使用其他平台
// 可能会有不同结果
string s = "中国";
cout << s.length() << endl;
for (int i = 0; i < s.length(); i++) {
    cout << hex << int(s[i]) << endl;
}
```



```
6
fffffffef4
fffffffb8
fffffffad
fffffffef5
ffffff9b
ffffffbd
```

UTF-8 编码, 中: E4B8AD, 国: E59BBD

'\0' 会特殊对待

```
char bytes[] = {  
    'a', 'b', 'c', '\0',  
    'd', 'e', 'f'  
};  
string s(bytes, 0, 7);  
cout << s.length() << endl;  
cout << s << endl;
```



3
abc

目录

- 基本 I/O
- 格式化输出
- 字符串
- 字符串 I/O
- 命名空间

字符串 I/O

- 包含 `<sstream>` 头文件
- `stringstream`、`istringstream` 和 `ostringstream`
 - `stringstream` 可以输入也可以输出
 - `istringstream` 操作方式类似于 `cin`
 - `ostringstream` 操作方式类似于 `cout`

字符串 I/O

```
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

int main() {
    istringstream ss("hello, world");
    // or
    // string input = "hello, world";
    // istringstream ss(input);
    string s;
    ss >> s;
    cout << s << endl;
}
```

输出为 hello,

```
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

int main() {
    ostringstream oss1;
    oss1 << "hello, world";
    ostringstream oss2("bye, ");
    oss2 << "world";
    // ios_base::ate 表示追加
    ostringstream oss3("bye, ", ios_base::ate);
    oss3 << "world";
    cout << oss1.str() << endl;
    cout << oss2.str() << endl;
    cout << oss3.str() << endl;
}
```

输出为

```
hello, world
world
bye, world
```

字符串 I/O

```
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

int main() {
    stringstream ss;
    ss << "hello, ";
    string s;
    ss >> s;
    ss << "world";
    cout << "s = \"" << s << "\", ss = \"" << ss.str() << "\"" << endl;
}
```

输出为

```
s = "hello,", ss = "hello, world"
```

目录

- 基本 I/O
- 格式化输出
- 字符串
- 字符串 I/O
- 命名空间

使用命名空间 std

- cin 和 cout 是 C++ 标准库内置**对象**而不是关键字
- 标准库的所有标识符都在命名空间 std 中

```
using namespace std; // 直接使用 cin、cout
```

```
using std::cin; // 直接使用 cin、cout, 而来自标准库的其他符号  
using std::cout; // 需要加上 std:: 前缀
```


资源和工具

- 资源

- <https://en.cppreference.com/w/>
- <https://zh.cppreference.com/w/%E9%A6%96%E9%A1%B5>
- 注意 C++ 标准的版本, 下图表示从 C++11 开始可用

std::stoi, std::stol, std::stoll

Defined in header <string>

int	stoi(const std::string& str, std::size_t* pos = nullptr, int base = 10);	(1)	(since C++11)
int	stoi(const std::wstring& str, std::size_t* pos = nullptr, int base = 10);		
long	stol(const std::string& str, std::size_t* pos = nullptr, int base = 10);	(2)	(since C++11)
long	stol(const std::wstring& str, std::size_t* pos = nullptr, int base = 10);		
long long	stoll(const std::string& str, std::size_t* pos = nullptr, int base = 10);	(3)	(since C++11)
long long	stoll(const std::wstring& str, std::size_t* pos = nullptr, int base = 10);		

- 工具

- 在线 C++ 编辑器 <http://cpp.sh/>
- 在线 C++ 调试器 <https://www.onlinegdb.com/>