

CS100 Recitation 6

从 C 到 C++：输入输出、字符串与动态容器

Speaker jiangye

CS100 Teaching Assistant



上海科技大学

ShanghaiTech University

今天我们要回答什么？

1. C++ 的输入输出为什么不需要格式串和取地址？
2. `std::string` 与 C 风格字符串有什么本质区别？
3. `std::vector` 如何替我们管理动态数组？
4. 容器增长时，哪些指针、引用和迭代器会失效？

从 C 到 C++，不是换一个文件后缀，而是把资源管理交给对象。

目录

1 从 C 程序到 C++ 程序

2 std::string：字符串成为对象

3 std::vector：动态数组成为容器

4 资源管理、增长与失效

从 C 程序到 C++ 程序

同一件事，两种写法

C：printf 输出字符串

```c
#include <stdio.h>

int main(void) {
  printf("Hello, C!\n");
}
```

C++：向输出流插入字符串

```cpp
#include <iostream>

int main() {
  std::cout << "Hello, C++!\n";
}
```

- C++ 程序通常使用 .cpp 后缀，入口仍然是 main。
- std::cout 是标准输出流对象，<< 在这里是插入运算符。

立志成才 报国裕民

编译器与语言标准


```
gcc -std=c17 -Wall -Wextra -Wpedantic hello.c -o hello
g++ -std=c++17 -Wall -Wextra -Wpedantic hello.cpp -o hello
```

- 本课程使用 **C17** 与 **C++17**；编译 C++ 时使用 `g++` 或 `clang++`。
- 编译器会根据源文件后缀选择语言，但构建命令中仍应明确指定标准。

C++ 不是“带类的 C”

两种语言共享大量语法，却具有不同的类型规则、标准库和惯用写法。不要假定每份 C 程序都能直接作为 C++ 编译。

立志成才 报国裕民

输入输出：对象与运算符


```cpp
#include <iostream>

int main() {
  int a;
  int b;
  if (!(std::cin >> a >> b)) {
    std::cerr << "expected two integers\n";
    return 1;
  }
  std::cout << a + b << '\n';
}
```

- `std::cin >> x` 从输入流提取一个值，`std::cout << x` 向输出流插入一个值。
- 运算符返回流本身，因此可以连续书写；流的转换结果可用于检查输入是否成功。

立志成才 报国裕民

为什么不再需要格式串和取地址？

C：调用者描述类型与地址

```c
int x;
double y;
scanf("%d%lf", &x, &y);
printf("%d %f\n", x, y);
```

C++：运算根据对象类型选择

```cpp
int x;
double y;
std::cin >> x >> y;
std::cout << x << ' ' << y << '\n';
```

- `scanf` 依赖格式串；`std::cin` 知道 `x` 的类型，并通过引用完成写入。
- 输出同样不需要%d/%s；插入运算根据表达式类型进行处理。

方向记忆：数据从 `cin` 流向变量，从变量流向 `cout`。

立志成才 报国裕民

std::endl 不只是换行


```cpp
std::cout << "next line\n"; // 写入换行符
std::cout << std::endl;     // 换行，并刷新输出缓冲区
```

- '\n' 写入换行符；std::endl 写入换行符并**刷新输出缓冲区**。
- 普通逐行输出优先使用'\n'；需要立刻显示或刷新日志时再使用 std::endl。

刷新是额外语义，不是每次换行都必须做的事情。

立志成才 报国裕民

名字住在哪里：namespace std


```cpp
std::cout << "fully qualified\n";

using std::cout;
cout << "introduce one name\n";

using namespace std; // 不推荐依赖这种写法
cout << "the whole namespace is visible\n";
```

- 标准库的大部分名字位于命名空间 `std` 中，避免与用户代码重名。
- 最明确的写法是 `std::cout`；也可用 `using std::cout` 只引入一个名字。
- 避免依赖 `using namespace std`；**不要在头文件的全局作用域中写任何这类 using 声明或指令**。

立志成才 报国裕民

C 标准库在 C++ 中的入口


```cpp
#include <cstdio>

int main() {
  int answer = 42;
  std::printf("answer=%d\n", answer);
}
```

C 头文件	C++ 对应头文件
<stdio.h>	<cstdio>
<string.h>	<cstring>
<ctype.h>	<cctype>
<math.h>	<cmath>

<cxxx> 把相应设施放入 std，并可能同时提供全局名字。

<string> 与 <cstring> 不是一回事。

旧代码中的 C 头文件仍可能可用；新的 C++ 代码优先使用对应 <cxxx> 形式。

立志成才 报国裕民

兼容不等于完全相同

规则	C17	C++17
布尔类型	包含 <stdbool.h> 后使用 bool	bool、true、false 内建
字符串字面量类型	char[N]；修改仍是 UB	const char[N]
字符字面量'a'	类型为 int	类型为 char
声明 int f()	参数信息未指定	不接受参数
逻辑与关系表达式	结果为 int	结果为 bool
void * 到对象指针	允许隐式转换	必须显式转换

把.c 改名为.cpp，得到的是另一门语言中的程序。

立志成才 报国裕民

迁移第一段代码：按顺序检查

1. 源文件与编译器匹配：`.cpp`、`g++`、`-std=c++17`。
2. 每个使用的设施都有对应头文件：`<iostream>`、`<string>`、`<vector>` 等。
3. 标准库名字写全，或只引入需要的名字。
4. 不机械混用两套接口：本讲优先使用 iostream、`std::string` 与 `std::vector`。
5. 保留 C 中学到的检查：输入失败、边界、对象生命周期和未定义行为仍然重要。

先用小而完整的 C++ 程序建立正确习惯，再逐步迁移复杂代码。

std::string：字符串成为对象

从字符数组到字符串对象

	C 风格字符串	std::string
表达	以 NUL 结尾的字符数组	标准库字符串对象
长度	strlen 逐字符查找	s.size() 直接查询
拷贝	字符数组不能直接赋值	s2 = s1 复制内容
连接	strcat 要求目标容量足够	+、+= 管理所需存储
释放	若动态分配，调用者负责 free	对象析构时自动释放其资源

- 头文件是 <string>，不是 <string.h> 或 <cstring>。
- std::string 表达的是**字符串值**，不是指向首字符的裸指针。

自动管理内存不等于没有边界、生命周期和无效访问。

立志成才 报国裕民

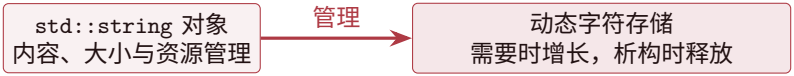
定义、初始化与拷贝


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

int main() {
  std::string greeting{"Hello"};
  std::string copy = greeting;
  greeting += ", C++!";
  std::cout << greeting << '\n';
  std::cout << copy << '\n';
  std::cout << "size=" << greeting.size() << '\n';
}
```

- `std::string s` 得到空串；`std::string s(7, 'a')` 得到七个`'a'`。
- 拷贝构造和拷贝赋值复制内容；修改一个对象不会改变另一个对象。

对象替我们管理什么？



- 不再手动维护末尾 NUL、用 `realloc` 扩容或配对 `free`。
- 关注字符串内容；具体成员布局与增长策略属于实现细节。

对象本身与其管理的字符存储不是同一个概念；不要把示意图当作内存布局保证。

立志成才 报国裕民

长度与空串：size()、empty()


```cpp
std::string empty;
std::string text{"hello"};
std::cout << empty.size() << ' ' << empty.empty() << '\n';
std::cout << text.size() << ' ' << text.empty() << '\n';
```

- `s.size()` 返回字符单元数，类型通常是 `std::size_t`；不是 `strlen`，更不是 `sizeof`。
- `s.empty()` 直接表达“是否为空”，通常比 `s.size() == 0` 更清晰。

对 UTF-8 文本，`size()` 统计的是字节数，不一定是人类感知的字符个数。

立志成才 报国裕民

连接：+ 与 +=


```cpp
std::string hello{"hello"};
std::string result = hello + " world" + "!";
result += " C++";

const char *cstr = "hello";
// std::string bad = cstr + " world"; // 错误：两侧都不是string
```

- 二元 + 生成新字符串；两侧至少一个操作数必须是 std::string。
- + 左结合：hello + " world" + "!" 先得到字符串，再继续连接。
- += 直接追加到左侧对象；不需要调用 strcat 或手动检查目标容量。

比较的是内容，不是地址


```cpp
std::string a{"apple"};
std::string b{"apply"};
bool same = (a == b);   // false
bool before = (a < b);  // true: 'e' < 'y'
```

- 使用 ==、!=、<、<=、>、>= 直接比较内容。
- 顺序比较是字典序，不需要调用 strcmp；赋值也复制内容而非别名。

不要比较 `s1.c_str() == s2.c_str()`：那比较的是两个指针值。

立志成才 报国裕民

下标访问与边界


```cpp
std::string s{"abc"};
char first = s[0];
char second = s.at(1);
s[0] = 'A';
// s[10] = 'x';   // 未定义行为
// s.at(10) = 'x'; // 抛出std::out_of_range
```

- 有效下标范围是 $[0, \texttt{s.size()})$；`s[i]` 不检查越界，越界是未定义行为。
- `s.at(i)` 在越界时抛出 `std::out_of_range` 异常。
- `front()`、`back()`、`pop_back()` 作用于非空字符串；空串上使用是未定义行为。

遍历字符串：基于范围的 for


```cpp
std::string text{"Hello C++"};
for (char c : text) {
  if (std::isupper(static_cast<unsigned char>(c)) != 0)
    std::cout << c;
}
```

- 读作“对 text 中的每个 char c”，不必手动维护下标。
- std::isupper 的参数要先转换为 unsigned char，避免负 char 值导致未定义行为。

这里的 c 是当前字符的副本；修改 c 不会修改 text。

立志成才 报国裕民

字符串输入：读一个词，还是读一行？

两种提取方式

```cpp
std::string word;
std::cin >> word; // 跳过分隔空白，读到下一个空白

std::string line;
std::getline(std::cin, line); // 从当前位置读到换行符
```

完整的逐行输入示例

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

int main() {
  std::string name;
  if (!std::getline(std::cin, name))
    return 1;
  std::cout << "Hello, " << name << "!\n";
}
```

- `cin >> word` 跳过前导空白，读到下一个空白为止；空白不存入字符串。
- `getline(cin, line)` 从当前位置读到换行符；换行符被取出但不存入字符串。
- 混用时若前一次输入留下换行符，下一次 `getline` 可能立即读到空串或本行剩余内容。

立志成才 报国裕民

与 C 接口交接：c_str()


```cpp
void legacy_print(const char *text);

std::string s{"hello"};
legacy_print(s.c_str());
// 不修改返回的字符数组，也不要把指针保存到s失效之后。
```

- `s.c_str()` 提供指向 NUL 结尾字符数组的指针，便于调用只读 C 接口。
- 不要通过该指针修改内容，也不要在字符串失效或被后续非 const 成员操作使指针失效后继续使用。

优先传递和使用 `std::string`；只在 API 边界处转换为 C 风格字符串。

立志成才 报国裕民

std::vector：动态数组成为容器

从手工动态数组到 std::vector

	C 中的手工方案	std::vector
存储	malloc/realloc	容器按需要管理
元素个数	单独保存 size 变量	v.size()
当前容量	单独保存 capacity 变量	v.capacity()
追加元素	检查容量、扩容、写入	v.push_back(x)
释放	每条成功与失败路径配对 free	析构时自动释放

- 头文件是 <vector>；元素在容器中连续存储。
- 它适合表达“大小可变化的一列同类型元素”。

自动化的是存储管理，不是算法复杂度、边界检查或失效规则。

立志成才 报国裕民

std::vector 是类模板


```
std::vector<int> numbers;
std::vector<std::string> words;
std::vector<double> samples;
```

- vector<T> 中的 T 是元素类型；给出模板参数后才得到具体类型。
- std::vector<int> 与 std::vector<double> 是不同类型。
- std::vector 本身不是可直接声明对象的完整元素类型。

```
std::vector<std::vector<int>> rows{
    {1, 2, 3},
    {4, 5},
    {6}
};
```

“vector of vector”可以表达每行长度不同的二维结构。

初始化：圆括号与花括号不一样


```cpp
std::vector<int> numbers;
std::vector<std::string> words;
std::vector<double> samples;

std::vector<int> a{2, 3, 5, 7}; // 4个元素
std::vector<int> b(10);         // 10个0
std::vector<int> c(10, 42);     // 10个42
std::vector<int> d{10};         // 1个元素，值为10
```

- `vector<int>(10)` 创建 10 个值初始化元素，即 10 个零。
- `vector<int>{10}` 创建 1 个元素，其值为 10。
- `vector<int>(10, 42)` 创建 10 个 42；花括号形式优先表示元素列表。

先确认你想表达的是“元素列表”还是“元素个数”。

立志成才 报国裕民

拷贝：复制整列元素


```cpp
std::vector<int> a{1, 2, 3};
std::vector<int> b = a;
b[0] = 99;
std::cout << a[0] << ' ' << b[0] << '\n'; // 1 99
```

- 拷贝构造与拷贝赋值复制所有元素，不需要手动写循环。
- 修改 `b[0]` 不会改变 `a[0]`；两个对象分别拥有自己的元素存储。

如果只是读取较大的对象，后续学习引用后可避免不必要的拷贝。

立志成才 报国裕民

大小、空容器与清空


```cpp
std::vector<int> v{2, 3, 5, 7};
std::cout << v.size() << ' ' << v.empty() << '\n'; // 4 0
v.clear();
std::cout << v.size() << ' ' << v.empty() << '\n'; // 0 1
```

- `v.size()` 是元素个数；`v.empty()` 判断是否为空。
- `v.clear()` 移除全部元素，使大小变为 0；不要手写循环逐个 `pop_back()`。
- `clear()` 不保证释放已分配存储，因此不保证 `capacity()` 变为 0。

立志成才 报国裕民

追加、访问首尾与删除末尾


```cpp
std::vector<int> values{2, 3, 5, 7};
values.push_back(11);
values.front() = 1;
++values.back();
std::cout << "size=" << values.size() << " front=" << values.front()
          << " back=" << values.back() << '\n';
for (int value : values)
  std::cout << value << ' ';
std::cout << '\n';
values.pop_back();
```

- `push_back(x)` 在末尾添加元素；`pop_back()` 删除末尾元素。
- `front()` 与 `back()` 返回首、尾元素的引用，因此可以修改原元素。

空容器上调用 `front()`、`back()` 或 `pop_back()` 都是未定义行为。

立志成才 报国裕民

读入 n 个整数：不再手写扩容


```cpp
#include <iostream>
#include <vector>

int main() {
  int n;
  if (!(std::cin >> n) || n < 0)
    return 1;
  std::vector<int> values;
  for (int i = 0; i < n; ++i) {
    int x;
    if (!(std::cin >> x))
      return 1;
    values.push_back(x);
  }
  for (std::size_t i = values.size(); i > 0; --
      i) {
    if (i != values.size())
      std::cout << ' ';
    std::cout << values[i - 1];
```

- 每次成功读入后调用 `push_back`，容器自动管理元素存储。
- 逆序输出使用有效下标 `i - 1`；无符号下标循环在 `i == 0` 时结束。

遍历容器：基于范围的 for


```cpp
for (int x : numbers)
  std::cout << x << '\n';

for (const std::string &s : words)
  std::cout << s << '\n';
```

- `int x : numbers` 逐个复制元素，适合廉价复制且只需读取的元素。
- `const std::string &s : words` 通过常量引用读取，避免复制每个字符串。

遍历期间不要改变同一个容器的大小；失效问题在下一章专门讨论。

立志成才 报国裕民

下标访问与边界检查


```cpp
for (std::size_t i = 0; i != v.size(); ++i)
  std::cout << v[i] << '\n';

// v[v.size()] = 42;   // 未定义行为
// v.at(v.size()) = 42; // 抛出std::out_of_range
```

- 有效下标范围是 `[0, v.size())`；`v[i]` 不检查边界，越界是未定义行为。
- `v.at(i)` 检查边界，越界时抛出 `std::out_of_range`。
- 下标与大小比较时使用 `std::size_t`，避免有符号／无符号比较告警。

标准库接口具有统一性

操作	std::string	std::vector<T>
元素个数	s.size()	v.size()
是否为空	s.empty()	v.empty()
清空	s.clear()	v.clear()
首／尾元素	s.front()、s.back()	v.front()、v.back()
末尾添加／删除	s.push_back(c)、s.pop_back()	v.push_back(x)、v.pop_back()
下标／检查访问	s[i]、s.at(i)	v[i]、v.at(i)

学会一个容器的基本词汇，可以把同样的意图迁移到另一个容器。

立志成才 报国裕民

C++17 CTAD：让编译器推导元素类型


```cpp
std::vector a{2, 3, 5, 7}; // std::vector<int>
std::vector b{3.14, 6.28}; // std::vector<double>
std::vector c(10, 42);     // std::vector<int>
// std::vector d(10);      // 错误：无法推导元素类型
```

- Class Template Argument Deduction（类模板参数推导）利用初始值推导模板参数。
- 信息不足时不能推导；例如只有元素个数，编译器不知道元素类型。

初学时写出 `std::vector<int>` 更直观；CTAD 是便利语法，不是新的容器。

立志成才 报国裕民

资源管理、增长与失效

vector 如何管理存储与增长？



`size() == 4`：已有元素 `capacity() == 8`：可容纳

对象管理一片连续元素存储；增长时可能搬迁，析构时自动释放，不需要 `free`。

每次只增加 1 个容量：第 $i+1$ 个元素到来时复制原有 i 个元素，n 次追加总复制量

$$\sum_{i=0}^{n-1} i = O(n^2)$$

容量按倍数增长（$0 \to 1 \to 2 \to 4 \to \cdots$）：扩容总复制量

$$1 + 2 + \cdots + 2^{m-1} = O(n)$$

均摊到 n 次 `push_back`，每次 $O(1)$；单次扩容仍昂贵，增长因子由实现选择。

- `size()` 是元素个数；`capacity()` 是当前存储可容纳的元素个数。容量大于大小不代表额外元素存在，不能通过下标访问。
- 已知大致元素个数时可用 `reserve(n)` 提前请求容量，减少重分配；但它**不创建元素**，`v[0]` 仍是未定义行为。

观察实际增长：输出随实现而变化


```cpp
#include <iostream>
#include <vector>

int main() {
  std::vector<int> values;
  for (int i = 0; i != 17; ++i) {
    values.push_back(i);
    std::cout << values.size() << ' ' << values.capacity() << '\n';
  }
}
```

- 每次追加后打印 `size` 与 `capacity`，观察何时发生容量增长。
- 可检查不变量：大小依次为 1 到 17，容量不减且始终不小于大小。

不要把某次运行得到的容量序列写成所有编译器都保证的规则。

重新分配会使哪些东西失效?


```cpp
std::vector<int> values{1, 2, 3};
int &last = values.back();
values.push_back(4); // 若重新分配，last失效
std::cout << last;   // 若last已失效：未定义行为
```

- 若 push_back 触发重新分配，元素被转移到新存储，旧存储被释放。
- 指向旧元素的**指针、引用和迭代器全部失效**；再使用就是未定义行为。
- 是否失效取决于是否重新分配，因此错误可能时隐时现，不能靠一次运行证明安全。

立志成才 报国裕民

遍历时追加：从错误写法到安全写法

错误：范围 for 依赖迭代器

```
for (int x : values) {
  if (x % 2 == 0)
    values.push_back(x + 1); // 可能使遍历迭代器失效
}
```

重新分配会使遍历所用的迭代器失效。

一种安全写法：固定原范围

```
const std::size_t original_size = values.size();
for (std::size_t i = 0; i != original_size; ++i) {
  int x = values[i];
  if (x % 2 == 0)
    values.push_back(x + 1);
}
```

先复制当前元素再追加；不保存可能失效的引用。

不要在基于范围的 `for` 遍历容器时改变该容器的大小。

立志成才 报国裕民

迁移到 C++ 后，仍然要保留哪些 C 语言直觉？

1. **类型：**对象知道类型，不代表转换和初始化规则可以忽略。
2. **边界：**`size()` 给出有效范围；`[]` 不检查，空容器没有首尾元素。
3. **生命周期：**对象自动析构；保存到对象失效后的指针或引用仍然危险。
4. **失效：**容器增长可能搬迁元素；不要继续持有旧位置的访问句柄。
5. **复杂度：**`push_back` 是均摊 $O(1)$，单次扩容仍需移动元素。

优先让标准库拥有资源，把精力留给真正需要推理的边界与算法。

立志成才 报国裕民

带走这四个改写动作

- **换工具：**使用 C++17 编译器和 `g++`，明确包含所需标准库头文件。
- **换表达：**用 `std::cin/std::cout` 表达输入输出，名字按需从 `std` 引入。
- **换字符串：**用 `std::string` 表达字符串值，拷贝、比较和连接直接作用于内容。
- **换动态数组：**用 `std::vector` 管理存储；仍要检查边界、空容器与失效规则。

标准库减少了手工管理，却没有取消类型、生命周期和未定义行为。

立志成才 报国裕民

Thank You