目录

命令行编译运行方法

这是一个非常实用的技能，尤其是在考试、机房环境或者服务器上工作时。

绝大多数考试和OJ（Online Judge）环境使用的是 **GCC/G++** 编译器。如果你在 Windows 上安装了 MinGW，或者在 Linux/macOS 环境下，使用的都是这一套指令。

以下是针对单文件编译的极简指南。

一、 GCC/G++ (最通用，适用于 Linux/Mac/Windows MinGW)

1. C 语言 (使用 `gcc`)

假设你的源代码文件名为 `hello.c`。

- **最简单的编译命令：**

```
gcc hello.c
```

  - **结果**：默认生成可执行文件。
    - Windows 下是 `a.exe`
    - Linux/Mac 下是 `a.out`

- **指定输出文件名 (推荐)：**

```
gcc hello.c -o hello
```

  - **结果**：生成名为 `hello` (Windows下是 `hello.exe`) 的可执行文件。
  - **记忆方法**：`-o` 后面紧跟 Output 名字。

- **链接数学库 (重要)：**
  如果你的 C 代码里用了 `math.h` 中的函数（如 `sqrt`, `pow`），在 Linux 下有时会报错 `undefined reference`，需要加上 `-lm`：

```
gcc hello.c -o hello -lm
```

2. C++ 语言 (使用 `g++`)

假设你的源代码文件名为 `main.cpp`。

- **最简单的编译命令：**

```
g++ main.cpp
```

- **指定输出文件名 (推荐)：**

```
g++ main.cpp -o main
```

- **开启常用警告 & 指定标准 (考试防坑必备):**

```
g++ main.cpp -o main -Wall -std=c++11
```

  - `-Wall`：开启所有警告（Warning All）。如果你写了未初始化的变量，编译器会提示你，这在考试查错时**非常有用。**
  - `-std=c++11`：指定使用 C++11 标准（也可以是 `c++14`, `c++17`）。有些旧的编译器默认标准很老，导致一些新语法（如 `auto`，`vector` 初始化）报错。

3. 如何运行程序

编译成功后，你需要运行生成的文件：

- **Windows (CMD/PowerShell):**

```
hello.exe
// 或者
.\hello.exe
```

- **Linux / macOS:**

```
./hello
// 或者是 ./a.out
```

注意：Linux/Mac 下必须加 `./` 表示当前目录，否则系统找不到文件。

二、 MSVC (Windows Visual Studio 原生编译器)

如果你发现机房电脑里装了 Visual Studio，但是打不开 IDE，或者你只能用命令行，你可以使用微软的编译器 `cl`。

前提：你必须打开 **"Developer Command Prompt for VS 20xx"** (在开始菜单里搜)，普通的 CMD 通常找不到 `cl` 命令。

- **编译 C/C++:**

```
cl main.cpp
```

  - **结果**：它会自动生成 `main.obj` 和 `main.exe`。
  - **运行**：直接输入 `main.exe` 即可。

三、 考试/上机 极简速查表 (Cheat Sheet)

建议背下这一行 **“万能命令”**，既能查错又能指定文件名：

语言	源代码	编译命令 (推荐)	运行 (Win)	运行 (Lin/Mac)
C	`test.c`	`gcc test.c -o test -Wall`	`test.exe`	`./test`

语言	源代码	编译命令 (推荐)	运行 (Win)	运行 (Lin/Mac)
C++	`test.cpp`	`g++ test.cpp -o test -Wall -std=c++11`	`test.exe`	`./test`

四、 常见问题 (Troubleshooting)

1. **Command not found / 不是内部或外部命令**
    - 原因：环境变量没配好。
    - 解决：
        - 如果是 Windows 且装了 VS，去开始菜单找 "Developer Command Prompt"。
        - 如果是 Windows 且装了 Dev-C++ 或 CodeBlocks，去它们的安装目录找 `MinGW\bin` 文件夹，把这个路径加到系统环境变量 Path 里（或者直接在这个 bin 目录下写代码编译）。
2. **undefined reference to ...**
    - 原因：链接错误。
    - 解决：C语言用数学函数记得加 `-lm`；如果是 C++ 代码错误使用了 `gcc` 编译，请改用 `g++`。
3. **编译成功了，但是运行没反应**
    - 解决：一定要看清楚生成的文件名是什么（默认是 `a.exe` / `a.out`），并且在命令行里手动输入文件名回车运行。

祝你上机顺利！只要记住 `g++ xxx.cpp -o xxx` 基本就能应付 99% 的情况。

读取和输出

1. 标准流 I/O (`iostream`)

这是 C++ 最常用、最推荐的方式，具有**类型安全**（不需要像 `printf` 那样指定 `%d`）和**可扩展性**的特点。

需要头文件：`#include <iostream>`

(1) 基本输出 `std::cout`

使用 `<<`（流插入运算符）将数据发送到标准输出。

```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 10;
    string s = "Hello";

    // 自动识别类型
    cout << s << ", world! Number: " << a << "\n";

    // endl 会换行并刷新缓冲区（速度较慢），"\n" 仅换行（速度快）
    cout << "End of line" << endl;
    return 0;
}
```

功能	printf 写法	cout 写法 (需 `<iomanip>`)
保留 2 位小数	`%.2f`	`cout << fixed << setprecision(2) << x;`
宽度 5 (右对齐)	`%5d`	`cout << setw(5) << x;`
宽度 5 (左对齐)	`%-5d`	`cout << left << setw(5) << x;`
补零 (宽度 3)	`%03d`	`cout << setfill('0') << setw(3) << x;`
输出字符串	`%s`	直接输出 (如果是 string)

(2) 基本输入 `std::cin`

使用 `>>`（流提取运算符）从标准输入读取数据。

- **特点**：会自动跳过空白字符（空格、制表符、换行符）。

```cpp
#include <iostream>
using namespace std;

int main() {
    int x;
    double y;
    // 如果输入 "10 3.14"，cin 会分别读取
    cin >> x >> y;
    cout << "x=" << x << ", y=" << y << endl;
    return 0;
}
```

(3) 读取整行 `std::getline`

`cin >> s` 读到空格就会停止。如果想读取包含空格的一整行（例如地址或句子），需要用 `getline`。

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    string str;
    // 读取一行，直到遇到换行符
    getline(cin, str);
    cout << "Input content: " << str << endl;
    return 0;
}
```

> **注意坑点**：如果先用 `cin >>` 读取数字，再用 `getline`，需要处理缓冲区里残留的换行符。
>
> ```cpp
> int n; cin >> n;
> cin.ignore(); // 忽略掉输入 n 后的那个换行符
> string s; getline(cin, s);
> ```

---

2. C 风格 I/O (`cstdio`)

这是继承自 C 语言的方式。在**算法竞赛**或需要**严格格式控制**时非常常用，因为它的速度通常比未经优化的 `cin/cout` 快。

需要头文件：`#include <cstdio>` (或 `#include <stdio.h>`)

(1) 格式化输出 `printf`

使用占位符来指定格式。

- `%d`: 整数
- `%lld`: `long long`
- `%f` / `%.2f`: 浮点数 / 保留两位小数
- `%s`: C 风格字符串 (`char*`)，如果是 `std::string` 需要用 `.c_str()`。

- 使用 `%nd`，`n` 代表最小宽度。
    - 默认是**右对齐**（前面补空格）。
    - 加负号 `%-nd` 表示**左对齐**（后面补空格）

```cpp
#include <cstdio>

int main() {
    int a = 123;
    double b = 3.14159;

    // 保留2位小数
    printf("Integer: %d, Float: %.2f\n", a, b);
    return 0;
}
```

(2) 格式化输入 `scanf`

同样使用占位符，注意变量前通常需要加 `&`（取地址符），数组名除外。

```cpp
#include <cstdio>

int main() {
    int a;
    long long b;
    // 读取直到遇到 EOF（文件结束符）
    // scanf 返回成功读取变量的个数
    while (scanf("%d %lld", &a, &b) != EOF) {
        printf("Read: %d %lld\n", a, b);
    }
    return 0;
}
```

---

3. I/O 性能优化 (算法竞赛专用)

默认情况下，`cin/cout` 为了兼容 `scanf/printf`，会进行同步，导致速度较慢。在处理百万级数据输入输出时，可以使用以下代码提速：

```cpp
#include <iostream>
using namespace std;

int main() {
    // 1. 关闭与 stdio 的同步
    ios::sync_with_stdio(false);
    // 2. 解除 cin 和 cout 的绑定（防止 cin 前自动刷新 cout）
    cin.tie(nullptr);

    // 之后只能混用 cin/cout，不能混用 scanf/printf
    int n;
    cin >> n;
```

```
    cout << n << "\n"; // 尽量用 "\n" 代替 endl，endl 会强制刷新缓冲区
    return 0;
}
```

4. 字符串读写

字符串的读写是 C++ 中最容易踩坑的地方，尤其是**混用 C 风格字符串（`char[]`）和 C++ `std::string`，以及混用不同的读取函数**时。

针对你的要求，我将按照**功能场景**（读单词 vs 读整行）来总结 `cin`、`scanf`、`fgets` 以及 `cout` 的用法。

---

1. 变量类型区别

在讲读取之前，必须明确你要读到哪里去：

- `std::string`：C++ 专用字符串，可变长，极其好用。（**推荐**）
- `char s[100]`：C 风格字符数组，长度固定，需要注意溢出。（老式代码或底层操作用）

---

2. 场景一：读取一个单词（遇到空格、回车、Tab 就停止）

输入示例：`Hello World`

(1) 使用 `cin`（支持 `string` 和 `char[]`）

最常用的方式，会自动跳过前面的空白符。

```
string s;      // 或者 char s[100];
cin >> s;      // 读入 "Hello"，剩下的 "World" 还在缓冲区
cout << s;     // 输出 Hello
```

(2) 使用 `scanf`（仅支持 `char[]`）

需要使用 `%s`。

- **注意**：`scanf` 不能直接读进 `std::string`，必须先读进 `char[]`。
- **缺点**：不安全，如果输入太长会爆内存（缓冲区溢出）。

```
char s[100];
scanf("%s", s); // 读入 "Hello"
// printf("%s", s);
```

---

3. 场景二：读取一整行（包含空格）

输入示例：`Hello World`

(1) 使用 `std::getline` (配合 `std::string`) —— 最推荐

这是 C++ 读取一行最标准的方法。它会读取直到换行符的所有字符，并**丢弃换行符**。

```
string s;
getline(cin, s); // 读入 "Hello World"
cout << s;
```

(2) 使用 `cin.getline` (配合 `char[]`)

这是 `cin` 的成员函数，用于读取到字符数组中。

```
char s[100];
cin.getline(s, 100); // 读入 "Hello World"，自动处理 '\0'
```

(3) 使用 `fgets` (配合 `char[]`) —— C 语言中最安全的方式

`fgets` 是设计用来替代不安全的 `gets` 的。

- **特点 1**：它会把换行符 `\n` 也读进去！(除非行太长超过了大小)。
- **特点 2**：必须指定最大长度。

```
char s[100];
// 从标准输入(stdin)读取，最多读 99 个字符 + 1 个结尾符
fgets(s, 100, stdin);

// 输入 "Hello"，s 的内容实际是 "Hello\n\0"
// 很多时候我们需要手动把那个 \n 去掉：
int len = strlen(s);
if(s[len-1] == '\n') s[len-1] = '\0';
```

(4) 使用 `scanf` 的正则用法 (配合 `char[]`) —— 竞赛常用黑科技

`scanf` 可以通过 `%[^\n]` 来实现“读到换行符为止”。

```
char s[100];
scanf("%[^\n]", s); // 读入 "Hello World"，遇到 \n 停止
```

---

4. 重点：输出 (Output)

(1) 输出 `std::string`

- 只能用 `cout`。
- 如果要用 `printf`，必须转换成 C 风格：`s.c_str()`。

```
string s = "Hello";
cout << s << endl;          // 正确
printf("%s\n", s.c_str());  // 正确，必须加 .c_str()
// printf("%s", s);         // 错误！！会乱码或崩溃
```

(2) 输出 `char s[]`

- `cout`、`printf`、`puts` 都可以。

```
char s[] = "Hello";
cout << s << endl;
printf("%s\n", s);
puts(s); // puts 会自动在末尾加一个换行符
```

---

5. 巨坑预警：混用时的“回车残留”

这是新手最容易遇到的 Bug。
当你先用 `cin` 或 `scanf` 读取一个**数字**，紧接着用 `getline` 或 `fgets` 读取**字符串**时，程序会“跳过”字符串的输入。

原因：`cin >> n` 读走了数字，但把回车符 `\n` 留在了缓冲区。紧接着的 `getline` 一看缓冲区里有个 `\n`，以为读完了一行空行，于是直接结束。

错误示例：

```
int n;
string s;
cin >> n;         // 输入 "10回车"
getline(cin, s); // 直接读取了 n 后面的那个回车，s 变为空字符串
```

解决方法：在读取字符串前，把那个回车“吃掉”。

```
int n;
string s;
cin >> n;

// 方法 A: C++ 风格 (推荐)
cin.ignore();

// 方法 B: C 风格
// getchar();

getline(cin, s); // 现在正常了
```

总结速查表

需求	变量类型	推荐代码	备注
读单词 (空格停止)	`string`	`cin >> s;`	简单方便
读单词 (空格停止)	`char[]`	`scanf("%s", s);`	注意缓冲区溢出
读整行 (含空格)	`string`	`getline(cin, s);`	**最推荐**，自动处理内存
读整行 (含空格)	`char[]`	`cin.getline(s, 100);`	方便，不留换行符
读整行 (含空格)	`char[]`	`fgets(s, 100, stdin);`	安全，但**保留换行符**
读整行 (含空格)	`char[]`	`scanf("%[^\n]", s);`	竞赛常用，不读换行符

一句话建议:
如果是写 C++，尽量全程使用 `std::string` + `cin/cout` + `getline`，可以避免 90% 的字符串处理错误。

String 常见操作

`std::string` 是 C++ 中最强大的工具之一，本质上它是一个封装好的**动态字符数组**（类似于 `vector<char>`）。

为了方便记忆，我把常用函数分为 **5 类**：基本查询、增删改、查找截取、转换、以及算法配合。

---

1. 基本信息与清空

函数	描述	示例
`size()` 或 `length()`	返回字符串长度	`s.size()` (这两个完全通用，习惯用哪个都行)
`empty()`	判断是否为空	`if (s.empty()) ...` (比 `s.size() == 0` 更直观)
`clear()`	清空字符串	`s.clear();` (变为空串 `""`)

---

2. 增加与修改 (拼凑)

函数/操作符	描述	示例
`+` 或 `+=`	**最常用**：拼接字符串或字符	`s += " World";` 或 `s = s + 'A';`
`push_back(c)`	在末尾添加**一个字符**	`s.push_back('a');`
`pop_back()`	删除末尾**一个字符**	`s.pop_back();`

> **注意**：`+=` 支持 string 和 char，而 `push_back` 只支持 char。

---

3. 截取、查找、替换 (做题核心)

这是算法题中最容易出 Bug 的地方，尤其是参数的含义。

(1) `substr` (截取子串)

- **语法**：`s.substr(开始索引, 截取长度)`
- **注意**：第二个参数是**长度**，不是结束索引！如果不写第二个参数，默认截取到末尾。

```
string s = "0123456789";
string sub1 = s.substr(2, 3); // 从下标2开始，取3个 -> "234"
string sub2 = s.substr(5);    // 从下标5开始，取到底 -> "56789"
```

(2) `find` (查找)

- **语法**：`s.find(子串)`
- **返回**：如果找到，返回**第一个字符的下标**；如果没找到，返回 `string::npos`。
- `string::npos`：这是一个特殊常量（通常是 -1 或极大的无符号数），用来表示"不存在"。

```
string s = "hello world";
int pos = s.find("wor");

if (pos != string::npos) {
    cout << "Found at: " << pos << endl; // 输出 6
} else {
    cout << "Not found" << endl;
}
```

- 还有 `rfind()`：从右往左找。（也返回的是首字母的位置）

(3) `insert` 和 `erase` (插入与删除)

- `s.insert(下标, "字符串")`：在指定位置插入。
- `s.erase(下标, 长度)`：删除指定片段。

```
string s = "hello";
s.insert(1, "123");   // "h123ello"
s.erase(1, 3);        // 从下标1开始删3个 -> 回到 "hello"
```

---

4. 类型转换 (数字 <-> 字符串)

做模拟题（比如大整数处理、回文数判断）时必用。

函数	描述	示例
`to_string(num)`	数字转字符串	`string s = to_string(123);` -> `"123"`
`stoi(s)`	字符串转 int	`int a = stoi("123");`
`stoll(s)`	字符串转 long long	`long long b = stoll("123456789012");`
`stod(s)`	字符串转 double	`double c = stod("3.14");`

---

5. 配合 STL 算法 (排序、反转)

`string` 支持迭代器，所以可以直接丢给 `algorithm` 库里的函数处理。
需要 `#include <algorithm>`。

```
string s = "bcad";

// 1. 排序 (字典序)
```

```
sort(s.begin(), s.end());
// s 变为 "abcd"

// 2. 反转
reverse(s.begin(), s.end());
// s 变为 "dcba"

// 3. 去重 (配合 erase)
// 比如 "aabbcc" -> "abc"
auto last = unique(s.begin(), s.end()); // unique 只能去除相邻重复，所以通常先 sort
s.erase(last, s.end());
```

---

6. 字符访问与比较

- **访问**：直接像数组一样用 `s[i]`。
- **比较**：直接用 `==`, `!=`, `<`, `>`。它是按照**字典序**比较的。

```
string s1 = "apple";
string s2 = "banana";
if (s1 < s2) cout << "apple 在 banana 前面"; // True
if (s1 == "apple") ...
```

总结速查

做题最常用的其实就这几个：

1. `s += c` (拼接)
2. `s.size()` (长度)
3. `s.substr(start, len)` (截取，记准参数!)
4. `s.find() != string::npos` (查找)
5. `to_string()` / `stoi()` (转换)
6. `sort(s.begin(), s.end())` (排序)

转义字符

在 C++ 中，编译器会把双引号 `"` 当作字符串的开头或结尾，把反斜杠 `\` 当作转义的开始。

如果你想在屏幕上直接输出这些具有“特殊语法含义”的字符，就需要**转义（Escape）**。

C++ 的转义符是 **反斜杠** `\`。

1. 核心保留字符（必须转义）

最常见的情况是你需要输出引号或反斜杠本身。

字符	含义	如何输出 (转义写法)	代码示例	输出结果
`"`	双引号	`\"`	`cout << "她说:\"你好\"";`	`她说:"你好"`
`\`	反斜杠	`\\`	`cout << "C:\\Windows";`	`C:\Windows`
`'`	单引号	`\'`	`cout << '\'';` (仅在char中必须)	`'`

> **注意**：在 `string` 双引号里写单引号 `'` 其实**不用**转义（`"I'm ok"` 是合法的），但在 `char` 里写单引号必须转义（`char c = '\''`）。

2. 常用控制字符（不可见字符）

这些字符在键盘上打不出来，或者代表特定的动作，也需要用 `\` 开头。

转义字符	含义	作用
`\n`	**换行 (Newline)**	光标移到下一行开头 (最常用)
`\t`	**制表符 (Tab)**	光标移到下一个对齐位 (通常用于列对齐)
`\0`	**空字符 (Null)**	字符串的结束标志 (编程时很关键)
`\r`	回车 (Return)	光标回到**当前行**开头 (常用于覆盖当前行输出)
`\b`	退格 (Backspace)	删除前一个字符

3. 特殊情况：`printf` 中的 `%`

如果你习惯用 C 语言风格的 `printf`，有一个特殊的保留字符：百分号 `%`。
因为 `%` 被用来做格式占位符（如 `%d`），所以想输出它自己，需要写**两个**。

- `cout`：直接写即可 `cout << "100%";`
- `printf`：必须写两遍 `%%`。

```
int n = 50;
printf("进度: %d%%", n);
// 输出: 进度: 50%
```

4. 进阶技巧：原始字符串 (Raw String Literal)

如果你需要输出**大量的**反斜杠（比如文件路径、正则表达式），写一堆 `\\` 会看得眼花缭乱。

C++11 引入了 **Raw String**，格式是 `R"( 内容 )"`。括号里的所有字符（包括 `\` 和 `"`）都会被原样输出，不需要转义。

```
// 普通写法 (很难看，容易数错反斜杠)
string path1 = "C:\\Program Files\\MyGame\\";

// Raw String 写法 (所见即所得)
string path2 = R"(C:\Program Files\MyGame\)";

cout << path2 << endl;
// 输出: C:\Program Files\MyGame\
```

总结速查

1. 想输出 `"` → 写 `\"`
2. 想输出 `\` → 写 `\\`
3. 想输出 `%` (仅printf) → 写 `%%`
4. 想换行 → 写 `\n`