速通系列

C++

入门

EthernosStudio

速通系列

C++

入门

Ethernos Studio 知识开放计划

dhjs0000 编

下载地址

扫描二维码或进入https://dhjs0000.github.io/EP/release.html

发布编号：43260118001
信息查询：
扫描二维码或进入https://dhjs0000.github.io/EP/query.html

主编：dhjs0000

本书仅供学习研究使用。

感谢您使用本书。本书遵循知识共享“署名-非商业性使用-相同方式共享”协议（BY-NC-SA）。您可以在遵守以下条款的前提下自由使用本书内容：

署名：您必须注明原作者“Ethernos Studio”及本书名称。

非商业性使用：您不得将本书内容用于任何商业目的。

相同方式共享：如果您基于本书内容进行改编、转换或再创作，则必须基于完全相同的BY-NC-SA协议分发您的贡献成果。

本书作者不对因使用本书内容而产生的任何直接或间接损失承担责任。本协议的解释权归原作者所有。如您使用本书内容，即表示您已阅读、理解并同意遵守此协议的全部条款。

前言

这不是一本“从入门到精通”的废话合集，也不是把《C++ Primer》嚼碎了喂给你的复读机。

《速通系列》的存在只有一个目的：让零基础的你在最短时间内写出能跑的C++代码。

我是一名2012年出生的OIer，Ethernos Studio创始人。我教过QQ群里的开发者配置Git代理，在GitHub维护着vsenv和ECL，也写过《速通系列Python 2026》。我深知初学者最怕什么：不是语法难，而是教程废话太多；不是概念深，而是学了半天写不出一个能用的程序。

这本书的特点是：

1．零基础，真从零开始 你不需要会任何编程。。不谈计算机历史，不扯设计理念，直接上代码。

2．实用主义，能跑就行 先教会你写A+B问题，再解释什么是变量。先让你用vector写程序，再讲内存管理。每一章末尾都是“实战：用今天学的知识解决XX问题”，而不是“习题：请阐述面向对象三大特征”。

3．不废话，一页一个知识点 `int a = 10;` 是什么意思？占半页。剩下半页是代码示例和常见错误。没有“从1946年ENIAC讲起”，没有“我们生活的世界充满了对象”。每节不超过5分钟，适合在晚自习或地铁上碎片化学习。

这本书不会教你模板元编程，也不会深入内存模型。但它会让你在30天内写出能通过入门级OI的代码，在60天内能独立开发一个命令行工具，在90天内能读懂GitHub上一部分C++项目的源码。

速通不是捷径，而是把弯路的坑提前填平。

打开IDE，抄下第一段代码，按下F5。你的C++旅程从下一页开始。

目录

第 1 章
基本入门

本章节内容大纲

1. C++环境的下载安装与配置
2. 你的第一个C++程序
3. 变量基础
4. 基本数据类型

自1844年莫尔斯电码划破时空，人类便开始了用形式化符号精确表达意图的探索。从打卡孔到汇编指令，从B到C，编程语言的进化史，本质上是一部不断降低人与机器沟通成本的文明史。C++诞生于贝尔实验室的1985年，如同一位承前启后的翻译官，既保留了与硬件对话的犀利，又赋予了构建庞然大物的优雅。如今，从火星车的轨迹计算到游戏引擎的粒子渲染，从高频交易的微秒博弈到操作系统的内核调度，C++的语法规则正悄然定义着数字世界的运行逻辑。

或许不久的将来，当通用人工智能真正理解人类意图，我们仍需通过严谨的语法结构向硅基伙伴下达不可二义的指令。语法，这门看似枯燥的符号契约，实则是程序员与编译器之间的君子协定——你遵守规则，它便还你效率。

当今社会，代码已成为继文字之后的第五大发明。生活中，每一次扫码支付、每一帧3D动画、每一导航路径，背后都是语法规则在静默支撑。我们享受着语法精确性带来的便利，却往往忽视其存在的价值。

1.1

C++环境的下载安装与配置

1.1.1 下载安装C++

C++并非一门"开箱即用"的语言。三十余年演进史催生了MSVC、GCC、Clang等编译器家族，它们对标准的支持度、优化策略乃至错误提示风格都各不相同——这本是工程自由的体现，却成了初学者第一道隐形门槛。我们无意在此探讨工具链哲学，实用主义要求我们快速跨过配置泥潭，直接进入代码世界。

Dev-C++并非功能最强大、界面最现代的IDE，但它将编译器、编辑器、调试器整合为开箱即用的绿色软件，不产生注册表污染，也不需配置环境变量。

第一步：下载安装包

Dev-C++ 官方原版已经是远古版本，但我们还有Embarcadero的社区版本。

打开浏览器，访问：

https://sourceforge.net/projects/embarcadero-devcpp/

点击Download按钮，下载 Embarcadero_Dev-Cpp_6.3_TDM-GCC_9.2_Setup.zip（约 70MB）。

第二步：解压和安装（一定要解压，不要直接双击exe）

将下载的 .7z 文件解压到任何路径，例如：

E:\DevCppInstaller\

然后进入解压目录，双击exe文件，等待安装程序加载完成，然后会弹出这个提示： 在一般情况下，直接点击OK就行了。

图1-1 安装界面

图1-2 输入目标文件夹

点了OK之后，同意许可证协议，也就是“我接受”按钮。在选择组建的页面，确保“选定安装的类型”为“Full”，然后点击下一步。

这一步是关键，在“目标文件夹”的输入框，最好要是不带中文和空格的路径，否则编译器可能会出现问题。

点击“安装”按钮，等待安装完成即可。

第三步：启动 Dev-C++

进入解压目录，双击 devcpp.exe。首次启动会提示选择语言，选 简体中文。

第四步：验证编译器

点击菜单 **工具 > 编译**选项，应看到：

编译器类型：TDM-GCC 9.2.0 64-bit Release

则安装成功。若提示错误，请重新检查路径。

第五步：创建第一个项目

点击 **文件 > 新建 > 项目**，选 **"Console Application"**

语言选 C++，项目名称填 hello

保存位置选 **E:\DevCpp\Projects**（自行创建文件夹）

将默认代码替换为：

```
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

按 F11 编译运行，若弹出黑框显示 Hello, World!，则环境配置完成。

1.1.2 配置C++

Dev-C++ 默认使用 C++98 标准，这会导致现代代码无法编译。本节将其升级为 C++14，并启用基础警告。

第一步：切换语言标准

点击 **工具 > 编译选项 > 代码生成/优化 > 代码生成 > 语言标准(-std)**，在下拉菜单中选择 GNU C++14。

为什么选择C++14？ 因为这是 OI 竞赛与工程实践的最低平衡点——支持 auto 类型推导、基于范围的 for 循环，又不会引入 C++17/20 的复杂特性。

第二步：开启编译警告

在 编译选项 对话框中，切换到 "编译器" 选项卡，在 "加入以下命令" 文本框填入：

```
-Wall -Wextra -Wconversion -Wshadow
```

这些标志的含义：

`-Wall` ：启用所有基础警告（未使用变量、类型不匹配等）

`-Wextra` ：补充额外警告（如 `int` 与 `bool` 隐式转换）

`-Wconversion` ：警告可能丢失精度的类型转换

`-Wshadow` ：警告变量名遮蔽（培养命名规范）

第三步：设置编辑器

点击 **工具 > 编辑器选项 > 显示**：

字体：选择 默认、**Cascadia Code**、consolas、JetBrain Mono（如果有）、Courier New 其一你感觉最顺眼的即可，字号 12

语法高亮：颜色选 Classic（白底黑字最护眼），图标可以按自己喜好调整。

第四步：规范文件编码

Dev-C++ 默认使用 GBK 编码，需强制改为 UTF-8 避免中文乱码。

新建文件时，点击 **文件 > 另存为**，在保存对话框底部将编码选为 UTF-8。此后每个新文件都必须执行此操作。

第五步：验证配置

将以下代码抄到 main.cpp，按 F11 编译：

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

int main() {
    std::vector<int> nums = {1, 2, 3};
    for(auto x : nums) {  // C++11 范围 for
        std::cout << x << std::endl;
    }
    return 0;
}
```

若编译无误，则配置成功。若编译报错，返回第一步检查语言标准。

1.2

你的第一个C++程序

1.2.1 cout输出

cout 是 C++ 内置的**输出**工具，专门用来把文字、数字等内容显示到屏幕上。你可以把它理解成一个已经连接好的“打印机”，只要告诉它要“打印”什么，它就会自动打印到纸条上面。

使用cout进行输出的格式如下：

cout << 要输出的内容1 << ... << 要输出的内容n;

```
1 // 这是一个C++程序
2 #include <iostream>
3 using namespace std;
4 /* 本程序是用来输出“你好，世界！”的
5     这是一个简单的C++程序 */
6 int main() {
7     cout << "你好，世界!" << endl; // 使用cout输出
8     return 0;
9 }
```

其中，参与cout语句的运算符 << 是作为输入运算符的，由于cout和它的运算符都在系统提供的头文件”iostream”中，因此需要使用#include 进行导入。

关于 << 运算符： << 在这里不是数学里的“小于小于”，而是 C++ 中数据流向的符号。这个 << 符号可以这么理解：就比如你有一个棒球发射器（也就是cout），这个 << 符号就代表这把一个棒球放入棒球发射器的那个筒子里，语句结束就是让棒球开始把这些文字“发射”到屏幕上。

程序运行结果如下：

```
你好，世界!
```

代码逐行解释：

第一行：以两个斜线”//”开始，表示这一行是注释，注释的意思就是这一行的内容是给人看的，不是给机器看的。在编译时，编译器会自动把注释里的内容当成空气。为了提高

程序的可读性，就需要为比较难以理解的代码添加合适的注释。

在C++中，除了使用”//”进行注释，还可以使用”/* … */” 进行多行注释。使用”//”表示在这一行内，这两个斜线后面的所有内容都是注释，都需要忽略，所以也被称为单行注释。而在”/* … */”中，只要是被/*和*/括在一起的所有内容都被当成注释，这也被称为块注释。第4到第5行就使用了块注释这种注释风格，这种多行注释除了使用块注释还可以使用多个单行注释：

```
// 本程序是用来输出“你好，世界！”的
// 这是一个简单的C++程序
```

第二行：是编译器的预处理指令，将文件iostream的所有代码嵌入该指令所在的位置。编译预处理指令以#符号开始，编译预处理语句不是C++语句，所以后面也不能加分号。

第三行：使用std命名空间，关于命名空间的内容，后面再专门介绍。

第六行：C++程序的主函数，程序入口一定是一个名为main的函数，关于函数的内容后面章节有专门介绍。main函数的类型只能是int或signed，函数使用“{”开始，“}”结束。

函数是由各个语句构成的，每一个语句以“;”结束，这个程序一共有2条语句。其中第一行使用cout输出字符串，cout是一个输出流对象，通过<<输入运算符，把文字“你好，世界!”输入进cout里，其中endl是换行符号，表示后面的输出都要在下一行。

最后的return 0 表示这个函数的返回值是0，与函数的int类型一致。

1.2.1 cin输入

cin 是 C++ 内置的输入工具，专门用来从键盘读取用户输入的数据。你可以把它理解成一个连接着键盘的“扫描仪”，它会等待你输入内容并按回车，然后把读到的一个一个的内容按照空格分别存放到变量里。

```
#include <iostream>
using namespace std;
int main() {
    int a,b;
    cout << “输入两个整数，空格分开:”;
    cin >> a >> b;
    cout << a << b; // 使用cout输出两个变量的值
    return 0;
}
```

使用cin进行输入的格式如下：

cin >> 接收数据的地方1 >> ... >> 接收数据的地方n;

其中，参与cin语句的运算符 >> 是作为提取运算符的，由于cin和它的运算符都在系统提供的头文件”iostream”中，因此需要使用#include 进行导入。

程序运行后，首先输出了”输入两个整数，空格分开” 然后输入 11 45，按下回车(Enter)键，程序运行结果：

输入两个整数，空格分开:11 45

1145

逐行解释：

第四行：这个程序定义了两个整型变量(int) a和b，cin是一个输入流对象，通过提取运算符把cin里的数据提取给各个变量。

第五行：输入的两个数据分别存放进变量a与b中，按空格分隔。

第六行：将a的内容和b的内容依次输出到屏幕，此时注意没有空格分隔。

练习1.2

1. 以下选项中，不是注释文本的是（）

A. // 这里是注释内容

B. /* 这里是注释内容
   这里还是注释内容 */

C. # 这里是注释内容

D. // 这里是注释内容
// 这里还是注释内容

2. cin与cout在系统提供的名为（）的头文件。

A. std

B. stdio

C. iostream

D. system

3. 以下哪段代码是错误的（）

A. cout << “Hello, World”;

B. cin << a << b;

C. cout << a << b;

D. cin >> x;

4. 尝试根据所学内容，自行写一个C++程序，要求是输出两行任意文字。

答案：C C B 略

1.3

变量基础

1.3.1 变量是什么

变量是内存中一块有名字的存储空间，用于保存程序运行时可变的数据。在C++中，使用变量前必须先声明类型再命名，这被称为强类型约束。

基本语法

类型名 变量名 = 初始值;

示例：四种基础类型

编译与运行

```
#include <iostream>

int main() {
    // 整数类型
    int age = 13;
    // 浮点数类型（小数）
    double height = 1.75;
    // 字符类型（单引号括起）
    char grade = 'A';
    // 布尔类型（真/假）
    bool isStudent = false;
    // 输出变量值
    std::cout << "年龄：" << age << std::endl;
    std::cout << "身高：" << height << "米" << std::endl;
    std::cout << "等级：" << grade << std::endl;
    std::cout << "是学生吗：" << isStudent << std::endl;
    return 0;
}
```

在 Dev-C++ 中新建项目，命名为 variables

将上述代码抄到 main.cpp

按 F11，预期输出：

```
年龄：13
身高：1.75米
等级：A
是学生吗：1
```

命名规则：

1. 只能包含字母、数字、下划线，且不能以数字开头
2. 区分大小写：age 和 Age 是两个变量
3. 不能使用关键字（如 int, return, if）
4. 建议：变量名用小驼峰（变量名多个单词直接连起来，第一个单词首字母不大写，从第二个单词开始，首字母需要大写，如关于学生名字的变量就用studentName），常量用全大写，多单词下划线分隔（MAX_SCORE）

变量赋值与修改

```
int score = 0;       // 声明并初始化
score = 100;         // 赋值（修改）
score = score + 5;   // 自增
```

常见错误

1. 错误：'age' was not declared in this scope
   解决：忘记声明变量，检查 int age; 是否在当前作用域
2. 错误：invalid conversion from 'double' to 'int'
   解决：int a = 3.14; 会丢失小数部分，应写为 double a = 3.14;
3. 错误：redeclaration of 'int age'
   解决：同一作用域内变量名重复，检查是否复制粘贴时未改名
4. 警告：'grade' is used uninitialized
   解决：变量声明后未赋值就使用，务必初始化（= 0 或 = 0.0）

1.3

基本数据类型

计算机的内存不是无限大的，因此程序中使用的任何数据都会在内存中占用一定的空间，而不同的数据类型也会占用不同的内存空间，有大有小。而且不同的数据类型也有自己的特性。就像现实中的各种容器，有大杯子小杯子，还有碗之类的。如果使用盆子放盐就会浪费空间，用小杯子放篮球显然不行，用竹篮打水——一场空。

在计算机中也是如此，因此才需要不同的数据类型。

1.3.1 整数（整型数据）

在C++中，为了存放恰当大小的整型数字，系统提供了四种不同的数据类型，有：整型int、短整型short(或short int)、长整型long(或long int)、长长（双长）整型long long。

其中，他们占用的内存关系是这样的：

整型 < 短整型 < 长整型 < 长长（双长）整型

计算机最小的存储单位是字节，GNU C++规范没有给出每种数据类型占多少字节，但给出了每种数据类型占用存储空间的最小值：

1. short 至少2字节
2. int 至少2字节，且不能少于short所使用的字节
3. long 至少4字节，且不能少于int所使用的字节
4. long long 至少8字节，且不能少于long所使用的字节

同一种数据类型，不同的系统、不同的架构所占用的空间可能会不太一样，但都需要遵守上面的规则。占用空间越大表示能存储的东西就越多，在表1-1中展示了在64位操作系统中不同数据类型的取值范围。

表1-1 基本整型数据及其取值范围

数据类型	取值范围
short	-32,768 ~ 32,767 (-2^{15}~2^{15}-1)
int	-2,147,483,648 ~ 2,147,483,647(-2^{31}~2^{31}-1)
long	-2,147,483,648 ~ 2,147,483,647(-2^{31}~2^{31}-1)
long long	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807 (-2^{63}~2^{63}-1)

在现实中，有些数字往往不存在负数，如年龄、人数等，因此我们也可以把负数部分省下来，给整数腾出更多空间。C++也为此提供的对应的数据类型，在各种整型前面加

unsigned就行，它叫无符号。顾名思义，它表示这个数字没有符号（也就是正还是负），每种整型前面都能加unsigned。在表1-2中展示了在64位操作系统中的不同无符号整型和它们的取值范围。

表1-2 无符号整型数据及其取值范围

数据类型	取值范围
unsigned short	0~65,535(0~2^{16}-1)
unsigned int	0~4,294,967,295(0~2^{32}-1)
unsigned long	0~4,294,967,295(0~2^{32}-1)
unsigned long long	0~18,446,744,073,709,551,615(0~2^{64}-1)

1.3.2 浮点数（浮点/实型数据）

我们在数学中，不仅仅只需要整数，还需要小数。而浮点数就是计算机中表示小数的方法。

C++提供了三种浮点数类型，分别是：float（浮点数）、double（双精度浮点数）、long double（长双精度浮点数）。同样的，虽然GNU C++规范没有给出每种数据类型占多少字节，但给出了每种数据类型占用存储空间的最小值：

1. float 至少4字节
2. double 至少6字节，且不能少于float所使用的字节
3. long double不能少于double所使用的字节

1.3.3 字符（字符数据）

在计算机中，还有类型用于表示单个字符。C++提供了一种字符类型：char（字符）。

在C++中，char字符类型通常通过ASCII码或Unicode码从数字转换成人眼能看到的字符。数字，英文字母，基本的符号一般在ASCII中表示，图1-1是一个基本的ASCII码图。

图1-3 ASCII字符代码表

在图中左侧“ASCII非打印控制字符”是不会实际显示的字符，用于一些控制操作和操作占位符，而右侧的“ASCII打印字符”就是可以在屏幕上显示的字符。

运行以下代码观察结果：

```
#include <iostream>
using namespace std;
int main() {
    char a = 97;
    char b = 98;
    char H = 72;
    cout << a << b << H;
}
```

输出结果应为：abH
与ASCII字符代码表上的对应相等。

1.3.4 布尔值（布尔/真假数据）

在现实中，还有些数据往往只有两种状态，特别是表示某个事物的状态是否是什么。在C++中，bool占一个字节，只有两种取值 true 和 false。表示真或假。

多知道一点——计算机如何存储数据

计算机系统将现实世界的信息映射为二进制位流的过程，严格遵循预先定义的数据表示规范。各类数据类型的存储方案均围绕位模式的含义解释与内存空间的布局展开。

一、 整数的存储：补码

整型存储的核心在于高效、无歧义地表示正数、零和负数，并支持高效的算术运算。现代计算机采用二进制补码作为标准。

1．存储格式

位宽固定：一般是8、16、32、64位，与CPU字长或指令集对齐。

最高位为符号位：0表示非负，1表示负。

值域：对于n位有符号整型，其表示范围为 $[-2^{n-1}, 2^{n-1}-1]$；无符号整型范围为 $[0, 2^{n-1}]$。

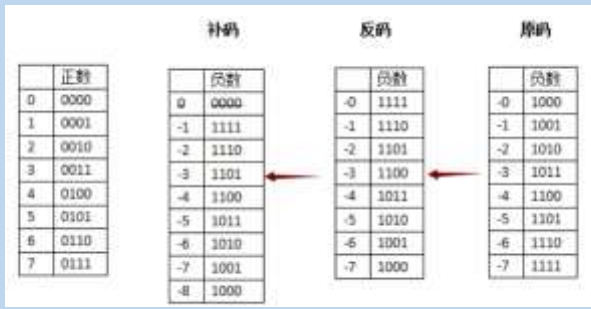

图1-4 补码、反码与原码

2．补码的生成与本质

设位宽为n。

非负数X（含零）：其补码为X的二进制原码，左侧以0填充至n位。

负数-X：其补码由以下步骤生成：

计算其绝对值X的二进制原码（使用n-1位）。

对所有n位（包括符号位）执行按位取反操作。

将取反后的结果加1。

数学本质：在n位模运算系统中，负数-X的补码实质上是其模2^n下的同余值，即 $(2^n - X)$ 的二进制表示。这使得减法 A - B 可被转化为加法 $A + (2^n - B)$ 进行计算，硬件上无需独立的减法器。

3．实例分析（8位有符号整型）

+5：原码 0000 0101 -> 补码 0000 0101

-5：

+5原码：0000 0101

按位取反：1111 1010

加1：1111 1011 -> 即-5的补码 1111 1011

特殊值：

0：唯一表示为 0000 0000。

最小值 -128 $(=-2^7)$：补码为 1000 0000。此值无对应的正数原码。

最大值 +127：补码为 0111 1111。

4．溢出与环绕

当运算结果超出该数据类型可表示的范围时，发生溢出。结果将在模2^n下发生位环绕。例如，8位无符号整数中，255 (1111 1111) + 1 = 0 (0000 0000，进位被丢弃)。

二、 浮点数的存储：IEEE 754标准

浮点数存储用于近似表示实数，采用基于二进制的科学计数法。IEEE 754是通用标准。

1. 存储格式（以单精度32位为例）

浮点数由三个字段构成：

V 为：$V = (-1)^S \times M \times 2^E$

表1-3 单精度32位浮点数的构成

字段	符号位（S）	指数位（E）	尾数位/有效数字（M）
位宽	1 bit	8 bits	23 bits
作用	决定正负（0正/1负）	存储偏移后的指数	存储规格化后的小数部分表示的数值

其中 M 和 E 的实际解释取决于指数域的值。

2. 关键组成部分详解

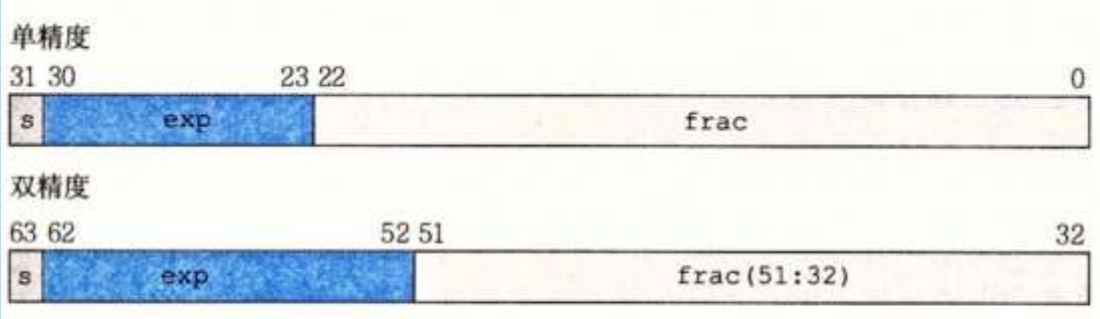

图1-5 浮点数的存储

符号位（S）：独立的符号位。

指数域（E）：采用偏移码表示。实际指数值 = E - Bias（单精度Bias = 127）。这使得指数范围从-126到127（E为1~254）可用于表示规格化数。

尾数域（M）：存储规格化二进制小数的小数部分。规格化数的二进制科学计数法形式为 1.xxxxx...，其中整数部分的“1”是隐含的（称为隐含前导1），不存储在M中，以节省1位精度。因此，实际的尾数值 M = 1.M（二进制拼接）。

3. 数值的分类与表示

根据指数域E的值，分为几种情况：

规格化数（E ∈ [1, 254]）

实际指数 = E - 127。

尾数 M = 1.M（隐含前导1）。

这是表示绝大多数非零数值的形式。

非规格化数（E = 0）

实际指数 = 1 - 127 = -126（非 -127，为了平滑过渡）。

尾数 M = 0.M（无隐含前导1）。

用于表示非常接近零的数（包括±0），填补了最小的规格化数与零之间的“下溢”间隙。

零值（E = 0, M = 0）

根据S的不同，有+0.0和-0.0两种表示，在比较运算中视为相等。

无穷大（E = 255, M = 0）

根据S的不同，有+∞和-∞。表示数值溢出或除以零的结果。

NaN（E = 255, M ≠ 0）

表示“非数值”，用于表示无效操作结果（如√-1，0/0，∞-∞）。

4．双精度浮点数

采用64位：1位S，11位E（Bias = 1023），52位M。原理与单精度完全相同，但拥有更大的范围和更高的精度。

三、 字符型数据的存储：编码映射

字符的存储本质上是存储其在特定字符编码标准中对应的整型码值。

1．ASCII编码的存储

标准ASCII使用7位（值0-127），扩展ASCII使用8位（值0-255）。

存储时，该整型码值直接以其二进制形式（通常占用1字节）存放。

例如：字符'A'的ASCII码为65，存储为 0100 0001。

2．Unicode及其编码方案的存储

Unicode定义全球字符的唯一码点（Code Point），如U+4E2D表示“中”。存储时需要编码方案将码点转换为字节序列。

UTF-32：直接存储。每个码点固定使用4字节（32位）。例如，U+4E2D存储为 0000 4E2D（大端序）或 2D 4E 00 00（小端序）。简单但空间效率低。

UTF-16：变长编码（2或4字节）。对于基本多文种平面（BMP，U+0000至U+FFFF）的字符，直接使用其16位码元。对于辅助平面的字符（如表情符号），使用一对代理对（一个高代理码元和一个低代理码元）共4字节表示。

UTF-8：变长编码（1至4字节），与字节序无关。

单字节字符以0开头，与ASCII完全兼容；多字节字符的首字节以连续几个1指示长度，后续字节均以10开头。

例如：“中”（U+4E2D）的UTF-8编码为3字节：**11100100 10111000 10101101**。其中加粗部分拼接起来即为码点4E2D的二进制位。

四、 布尔型数据的存储：最小化分配

布尔值在逻辑上只需1位，但在大多数编程语言的内存中，出于地址对齐和最小可寻址单元（字节） 的限制，通常分配一个完整的字节。

存储约定：数值0表示false，任何非零值（在C等语言中通常约定为1）表示true。

内部处理：CPU在进行布尔判断时，通常检查整个字节或字是否为零。

空间优化：在布尔数组或位字段中，编译器或程序员可使用位掩码操作，将多个布尔值压缩存储于一个字节或字中，以节省空间。

练习1.3

1. 在 64 位 Linux 系统上，`int a = -1; unsigned short b = a;` 执行后，b 的值是（ ）

A. 65535　　B. -1　　C. 0　　D. 编译错误

2. 判断

(1) 根据 GNU C++ 规范，long long 类型至少占用 8 字节，且不会少于 long 类型所占用的字节数。（ ）

(2) 在 8 位有符号整数的补码表示中，数值 -128 的编码为 1000 0000，该编码对应的原码不存在。（ ）

(3) IEEE 754 单精度浮点数的尾数域采用隐含前导 1 的规格化形式，因此其有效精度为 24 位二进制位。（ ）

(4) UTF-8 编码中，字符 ‘A’ 与 ASCII 编码完全兼容，而汉字 ‘中’ 的 UTF-8 编码需要 4 字节存储。（ ）

3.尝试解释为何现代计算机系统采用补码形式存储有符号整数，而非原码或反码。结合 8 位二进制系统，说明补码表示法如何简化硬件电路设计，并分析 0 + (-0) 运算在补码体系下的执行过程。

答案：1.A 2.TTTF 3.略

第 2 章
数据处理

本章节内容大纲

1. 数据存储
2. 数据运算

自结绳记事起，人类便在不断发明更高效的符号系统来固化转瞬即逝的经验。从甲骨文的裂纹到帕斯卡的齿轮，从打卡机的孔洞到磁盘的扇区，存储介质的进化史是人类对抗遗忘的文明史诗。C++作为一门根植于系统底层的语言，其数据处理哲学既继承了汇编的直接与犀利，又容纳了抽象的类型安全——数组是内存的线性投影，指针是地址的数学表达，而运算符则是硬件指令的符号化封装。如今，从数据库的B+树索引到神经网络的矩阵乘法，从哈希表的冲突解决到缓存行的对齐优化，数据处理的效率正悄然定义着数字文明的边界。

或许不久的将来，当存算一体芯片普及，我们仍需通过严谨的数据结构向机器描述问题的拓扑。数据处理，这门看似枯燥的内存搬运与比特运算，是程序员与冯·诺依曼架构之间的契约——你描述布局，它便还你性能。

当今社会，数据已成为继土地、能源之后的第五大生产要素。生活中，每一次推荐算法的精准投喂、每一帧光线追踪的逼真渲染、每一趟物流路径的秒级规划，背后都是数据处理逻辑在静默支撑。我们享受着存取性能带来的便利，却往往忽视其设计的精妙。

2.1

数据存储

2.1.1 常量

在程序中，不能改变它所代表的值的量就被称为常量。

字面量（直接常量）

字面量，顾名思义，就是在直接使用文字表示出的量，如`1, -114.5` 等等都属于字面量。

在C++中，字面量的使用如下：

```
#include <iostream>
using namespace std;
int main() {
    cout << sizeof(128) << endl; // int
    cout << sizeof(128U) << endl; // unsigned
    cout << sizeof(128L) << endl; // long
    cout << sizeof(128LL) << endl; // long long
    cout << 1.14 << endl; // 小数
    cout << -1.14 << endl;
    cout << 1.14E+4 << endl; // 科学计数法
    cout << -11.45E-12 << endl; // 科学计数法
    cout << sizeof(1.14) << endl; // C++的浮点数字面量默认使用double
    cout << sizeof(1.14F) << endl; // 在末尾添加F就是float了
    cout << 'A' << endl; // char
}
```

运行这个程序，应输出：

```
4
4
4
8
1.14
-1.14
11400
-1.145e-11
```

8
4
A

在程序中，sizeof表示的是这个量所占用的字节数，我们在前面的章节已经了解到，int至少2字节、long至少4字节、long long至少8字节，unsigned的字节数与int相同。而程序运行的结果也符合这个规定，还能说明输出的数字确实都是不同的数据类型。

C++中，科学计数法属于浮点数，打印的时候会自动转换成正确的形式。

在char中，除了你在键盘中可以打出来，屏幕可以显示的字符。还有一些字符无法在里面表示，如换行，单、双引号等。这时，就可以使用**转义序列**。

转义序列使用反斜杠开头表示，右边跟着的是转义代码，C++中，转义代码和对应的意义如下表：

表2-1 C++ 转义字符表

转义序列	名称	ASCII值	释义
\a	警报/响铃	7	使终端发出蜂鸣声
\b	退格	8	将光标向左移动一格
\f	换页	12	打印机中的换页操作
\n	换行	10	将光标移动到下一行开头
\r	回车	13	将光标移动到行首
\t	水平制表符	9	水平跳格（通常是8个空格）
\v	垂直制表符	11	垂直跳格
\\	反斜杠	92	表示字符 \
\'	单引号	39	表示字符 '
\"	双引号	34	表示字符 "
\?	问号	63	在某些三字母词中需要使用
\0	空字符	0	字符串结束符
\ooo	八进制数		最多3位八进制数字
\xhh	十六进制数		任意位十六进制数字
\uXXXX	Unicode字符		16位Unicode字符
\UXXXXXXXX	Unicode字符		32位Unicode字符

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

int main() {
    cout << "换行示例：第一行\n第二行" << endl;
    cout << "制表符示例：姓名\t年龄\t成绩" << endl;
    cout << "退格示例：123\b456" << endl;  // 显示"12456"
    cout << "引号示例：他说：\"你好，世界！\"" << endl;
    cout << "路径示例：C:\\Users\\Documents" << endl;
    cout << "八进制示例：\101" << endl;  // 输出 'A'
    cout << "十六进制示例：\x41" << endl;  // 输出 'A'
    cout << "Unicode示例：\u03B1" << endl;  // 输出希腊字母α

    return 0;
}
```

最后更新日期：2026-01-23

该版本为未完成版本，不代表最终品质

发布ID:43260118001