龙语言中文编程

The Loong Lang Book

作者：龙语者

版本：0.0.1

日期：2026-03-01

最新版：下载

书源码：下载

如果您有任何问题或建议，

可以通过邮箱 loong-lang@qq.com 联系我

网址：loong-lang.com

谨以此书献给龙语者诸多道友及后来者。

前言

“于千年传承之中领悟阴阳。两河是吾血脉，五岳是吾肢爪。吾之精神，刚健中正、自强不息、兼容进取。于时空之中，感悟天运、国运、人运。点滴贡献，汇聚长河，继承往者，兼收外道，融合发展，传之后世。”————龙语者

几千前年，黄河一带的华夏先祖，在生活生产实践中，逐渐发明自己的语言文字，形成自己的文明。商周朝代，人们把当时的文字记录在龟壳和骨片之上，称之为甲骨文。至今发现十余万片，曾被日本、欧美国家抢盗走几万片。甲骨文约四千单字，是迄今为止中国发现的年代最早的成熟文字系统。现在，“甲骨文”居然成了美国一家软件公司的名字。

有了文字，文明得以记录与传承，人们逐渐领悟了“阴阳”二意。阴阳是对立统一的两股力量或法则，运动转化，又相互包含，可观摩感悟“太极鱼”图案。阴阳起源，或许源自对太阳和月亮（太阴）的观察感悟，对应到白天（阳）和夜晚（阴），逐渐扩展到四季，春（少阳）夏（太阳）秋（少阴）冬（太阴），逐渐扩展到人事，男为阳，女为阴，等等。周朝时期，产生了一本书《周易》，也称《易经》，以符号代表阴阳，阳爻、阴爻。三个符号组合，可表示八种象（二的三次方），即八卦。八卦两两（共六爻）再组合，即组成六十四卦（二的六次方）。《易经》最初是为周王室的贵族阶层编写，不是写给普通人的，含义隐晦不明。内含人生大道，或帝王心术（个人观点）。孔子老而好《易》，韦编三绝，为解读它，孔子及后学作了十篇注释著作，总称为《十翼》。“阴阳”之意为什么如此重要，因为它不仅体现古代哲学思想，更是现代信息技术的基石————二进制。有句话说，“21 世纪分两种人，懂 1、0 的人和不懂它的人”，本意是，分为领悟“阴阳”与否的人。后面会再详述。

横观全球地理，中国无疑是块风水宝地。地处温带，气侯适宜，四季分明。西靠大陆，以高原、山脉作为屏障，东临大海，冷热干湿（阴阳）气流交汇之处，降水丰沛。于是有了长江、黄河，四季不干涸，粮食生产丰富，人民得以繁衍生息，哺育华夏人民几千年，称之为母亲河，毫不为过。当珍惜保护环境与水源。

几千年的传承与发展，产生了龙图腾，或许是古老部族图腾的融合演化，集各家所长，成为中国文化象征。龙的精神，深入人心。中国人乃至全球华人，自称“龙的传人”，承载传承龙的精神。于是有了龙脉的概念，长江、黄河就是中国最大的两条“水龙脉”。而众多山脉，南北或东西纵横，自然成为“山龙脉”。山脉不仅是气候的屏障，也是文

明的屏障。当天下大乱时，文明躲进大山得以保存；天下太平，文明又从大山走出。这或许是文明传承几千年不断的原因之一。当代人，还能读懂三千年前，春秋战国诸子百家的哲学经典，领悟先贤思想；又能读懂千年前的唐宋诗词文章，与古人际遇共鸣，想想是件多么了不起的事情。千年之后，后世之人或许亦能读到当代文章，那么当代之人，于历史的长河里，于文明的传承中，又该留点什么？

四百年前，明朝末年，国运式微。满清入主中原，给汉族及其文化带来巨大灾难。剃发易服，从外表到内部精神，打压毁坏篡改汉族文化。据说文盲率高达 90%，汉人文化传承几近断绝，更何谈创新。明朝编纂的百科全书《永乐大典》，中华文化思想科技集大成者，流落西方。此时欧洲正值文艺复兴，据说核心技术来自研究学习中国的《永乐大典》，真实历史大多已淹没在尘埃里。欧洲由此逐渐兴起工业革命，蒸汽机的改良、电力技术的产生与发展，大步领先东方。而此时的中国满清还在闭关锁国，大兴文字狱，打压科技创新。于是，自 1840 鸦片战争以来，西方以坚船利炮打开中国大门，满清以一种“宁予友邦，不予家奴”外来征服者的心态，签订各种不平等条约，给中国带来巨大灾难。中国逐渐走向亡国灭种的危险边缘。孙中山作为第一个有影响力的觉醒者，扛着“驱除鞑虏, 恢复中华”的大旗，带领推翻满清。满清误我华夏三百年！

一代人有一代的的命运与机遇，同时承担一代人的责任。我们上一代从饥饿与贫穷中走出来，而我们这一代人，迎来几十年的和平与发展。中国人用几十年时间，追赶上欧美国家近二百年的工业革命先发优势。当今，国际风云变幻，机遇与危险共生，中国重新崛起，文化自信回归，终将重新站于世界民族前列。

信息技术，是当代最重要的科技方向之一。20 世纪 50 年代，美国诞生第一台计算机，人类由此进入信息化时代。1970 左右，丹尼斯 • 里奇和肯 • 汤普森发明了计算机编程语言——C 语言及 Unix 操作系统，奠定现代信息产业的基石。1970 由是被称为纪元（epoch），这个时间点，从事软件开发的人应该比较熟悉。1991 年，芬兰大学生 Linus Torvalds 发布的最初的 Linux 操作系统，Linux 系统最大的特点是源码开放，以致全球千万工程师为其贡献，助其完善成熟。说来不怕笑话，本人小时候，大约 1997 年左右，家里才通了电，清楚记得第一次来电时惊喜的感觉，更别说电脑这种稀罕物。可见，中国计算机产业，曾经落后西方多年。

本人不才，出身寒微，却自负傲骨，剑走偏锋，辗转坎坷入行计算机软件开发行业，已将近二十年。凭着兴趣与坚持，尝自谓学遍天下编程语言。受中国武道思想影响，“天下武功极致，是创造一门自己的武功”，那么“计算机编程的极致，是创建一门自己的编程语言”。某日，读到计算机“编译原理”章节，突然萌生一个强烈的念头————发明一种纯中文的计算机编程语言。中文已经传承几千年，字字含义千锤百炼，表意简练精确，为什么不能用作计算机程序设计，让国人乃至全球共 15 亿华人及后裔从中受益。自 C 语言问世之后，至今 50 多年，英文编程语言层出不穷，而没有一门比较好的中文编程语言。随着汉文化的回归，中文编程必有它独特魅力与用武之地。随着人工智能与机器人的发展，或许将来的机器人会像现在的个人电脑一样普及。我希望将来某天，可

以用中文编程语言来控制各自的机器人。而中文编程，能像各自小时候写作文一样简单易用，像我们日常中文交流一样平常。我希望，精心编写的中文程序，能像唐诗宋词一样，流传千年！

我给这门中文编程语言命名————龙语言！

0.1 从阴阳到二进制

莱布尼茨，德国数学家和哲学家，17 世纪左右发明了二进制。野史传闻，某日，莱布尼茨兴冲冲向某传教士展示自己的研究成果，可以用 0 和 1 表示所有的数。传教士找来中国的八卦图，告诉他中国几千年前就发明了这个。二进制中的“0 与 1”对应着《周易》和八卦中的“阴与阳”。莱布尼茨大受震惊，于是曾向康熙皇帝写信请求加入中国籍, 他在信中表达了对中国文化的赞叹，并提出希望成为清朝子民。

我们从小熟悉的，生活中常用的数制是十进制。十进制起源是什么，或许源于人类的手指。在原始社会，那时还没有复杂文字、语言。某日，你家先祖上山打到两只猎物，别家先祖下水捕到十条鱼。于是，你家先祖拿出一只猎物，伸出十个手指头，要换十条鱼。别家先祖摆摆手，伸出 5 个手指头，表示一只猎物换 5 条鱼，成交！可见，十进制或许起源于人们的生活实践和习惯。

计算机中，最常用的数制是二进制。计算机（包括手机）数据，所有文字、图像、声音、视频，底层全是二进制进行编码。电路中的高电平、低电平，光纤中的光信号，空气中的无线电波，磁盘中的带磁与否，等等，底层都可认为是二进制编码。领悟到这一点，我们处在一个充满二进制的世界中。二进制，用中国传统术语，即为————阴阳。计算机程序，存储态是二进制，运行态也是二进制，因为它可以很方便的用与、或、非门电路表示。现代的 CPU 集成百亿甚至千亿晶体管构成的门电路，每秒可运行几十亿次 (频率)。它的运行效率，带来的生产力提升，已经超出人脑计算的维度，很难从直觉上去感受它的速度。打个比方，全球 80 亿人，全力计算某个数一年时间，而让计算机来算，打个响指，“叮咚”，它已经算出来了。而且它算得又快又正确，还不知疲倦。这正是我们依赖并发展电子计算机的原因。

8 个二进制比特 (bit) 位（阴或阳）组成字节 (Byte，简称 B)，2^{10}B=1KB，2^{10}KB=1MB，2^{10}MB=1GB。

0.2 什么是编程？

所谓编程，就是“编写程序”。什么是程序？就是做一件事情的步骤。现实中遇到一个问题，经过考虑思索之后，得出一个解决方案，按步骤一步步运算，就可以得出想要的结果。那么把解决方案的步骤，翻译成计算机能理解并执行的语言，就是编写计算机程序。程序员就是计算机程序的设计者与翻译者。

0.3 编程的难与易

编程难吗？不难，入门很容易。广义上讲，平时我们讲话，写文章，就是在编程，称之为“自然语言编程”。我们用流传几千年的，含义明确的语音及文字，按着大众约定的语法组织成语句，用来表达自己的思想，这难道不是编程吗？计算机编程语言，是由设计者定义的类似自然语言的一个更小的子集，定义更严格。

编程容易吗？也不易。当熟悉编程语言的语法之后，为一解决现实中的问题，还得熟悉领域（业务）知识，英文称之为 Domain Knowledge。比如，操作系统内核编程，得熟悉芯片指令集架构、各种数据结构与算法。再比如，网络 Web 编程，得熟悉网格协议相关知识。再比如，图形图像游戏编程，需要结合三维数学和 GPU 方面的知识。大多程序员，不过是在一个狭小的领域深耕而已。

总之，编程是一个入门容易，出成果也颇难的技能。龙语言中文编程的推出，第一目的，就是让全球更多中文用户，入此门中，去掉英文编程的神秘感，用中文指挥您的计算机按您的指令工作。

0.4 当今编程语言的不足之处

电子计算机诞生之初，打孔编程是主要的编程方式，通过在纸带上打孔来传递二进制指令，孔洞代表数字 1，空白则代表 0 。这种编程方式，容易出错，效率低下，于是人们发明了使用助记符的汇编语言，由汇编器把汇编语言翻译成机器能理解的二进制机器语言。助记符利于人们理解指令命令，比之打孔编程，是一个大的飞跃。随着程序越来越复杂，于 1970 年左右，诞生了 C 语言，使用一种更接近人类思维的方式来编程，这是第二个飞跃。在 C 语言的基础上，发明了 C++、Java、C# 等流行的编程语言，后来又有了 golang、rust 等后现代编程语言。经过 50 多年的高级编程发展，很多编程语言已经足够稳定成熟。但我认为依旧存在以下问题：

1、使用了过多的专业词汇简写，或者说行话、隐喻。只有理解并熟练掌握这些行话，才能把人类思维的自然语言程序，翻译成机器理解的程序，这对初学者来说是第一大障碍。比如，很多编程语言中常用的 int 关键字，即使去问英语当第一语言的人，int 代表什么意思，他十之八九答不上来，因为它可能是 integer(整数), 或者 interrupt(中断) 等等含义。再比如，著名的简写“GNU”，它代表的意思是“GNU’s Not Unix”，那意思中的 GNU 又代表什么意思？答案还是这个意思。如果能理解“GNU”的意思，并发觉它的值得玩味之处和领悟到它的笑点，那说明您对编程语言中的“递归”思想，已经理解。计算机软件极客黑客们，喜欢这类简写，并以此为乐，但这无疑是初学者的障碍。

2、自然语言思维转换的代价。对于使用其他自然语言，如中文的人来说，转换到英文是第二大障碍，想想自己当年初学英语所付出的时间精力有多少吧。

由此，对不懂计算机程序的人来说，虽看到它们每个字符都认识，但组合到一起表

达的含义，却完全不懂，就像看天书，魔法书一样。想到电影《黑客帝国》中的程序，满屏自上而下落下的未知的字符，这可能是不懂计算机程序的人看到计算机程序时的感觉。这其实是对它的误解，理解了计算机程序的词法、语法、语义，构建了知识模型之后，它表达的意思和自然语言是一样的。

0.5 龙语言简介

龙语言是一门使用纯中文的、面向对象的、现代编程语言，目前支持有限的范型编程。使用近乎中文自然语言的表达，来编写计算机程序。它的语义借鉴 C++，语法类似 Java。

它的设计目标是：

1、简单精确。避免各种简写带来的歧义，只要接受过九年义务教育，就能编程。利于初学者，尤其是中文用户。

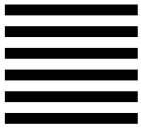
2、集成而非替代。它的存在，并非替换现有的任何一门编程语言，因为它们已经足够成熟稳定。龙语言只是提供更多一种选择，可编译成动态库、或静态库，与它们集成。

3、快速。龙语言编译器使用 C++ 实现，生成的指令运行效率与 C 语言接近，中文最终编译成机器语言，而非解释型中间代码。

4、跨平台编译、运行。基于 LLVM 编译器基础架构，实现跨平台编译与运行。编译过程依赖 C++ 语言运行时库，运行过程依赖 C 语言运行时库的支持。

如上所述，若引起您的兴趣，那么请跟我开启龙语言的学习研究之旅吧！希望龙语言成为您解决现实问题工具库中的一个，并希望有能力与条件的用户贡献它、完善它、传播它、传承它！

附上《周易》六十四卦之首的乾卦，其中真义，诸位道友自行领悟。



乾，元亨利贞
初九：潜龙勿用
九二：见龙在田，利见大人
九三：君子终日乾乾，夕惕若厉，无咎
九四：或跃在渊，无咎
九五：飞龙在天，利见大人
上九：亢龙有悔
用九：见群龙无首，吉
象曰：天行健，君子以自强不息

龙语者
2025 年 10 月 26 日
洞天井居

声明

龙语言编译器及相关文档，允许免费使用及任意分发，但请保留出处。

严禁用于破坏他人计算机系统，及窃取他人信息等非法用途，由此带来的后果，与编译器作者无关。

由编译器或程序缺陷带来的损失，编译器作者不负责。若用于生产环境，务必进行严格全面的程序测试。

致谢

- **C/C++ 作者:** 感谢 C 语言作者 Dennis Ritchie，C++ 作者 Bjarne Stroustrup，他们的工作奠定了现代软件产业的基石，启蒙了思想。
- **LLVM 作者:** 感谢 LLVM 项目发起人 Chris Lattner，集合全球顶尖编译器专家智慧，构建一套可复用的编译器开发基础架构，使发明一种新的编程语言，从未如此简单。
- **诸多开源贡献者:** 他们是现代软件产业中的无名英雄，但他们的思想及解决方案在项目的底层发光。

目录

第一章　龙语言中文编程速览

本章将对龙语言的核心语法和功能作一个快速的演示，以让用户快速形成一个大概的印象。对于有其它程序语言经验的用户，节省时间快速上手。对于无编程经验的新手用户，快速浏览一遍即可，后面章节针对每个核心功能，会深入详细解说。

1.1　Hello, 龙语者

学习程序设计的第一个程序，约定俗成的是编写“Hello World”程序，编译运行，在控制台打印一句话。龙语言的版本如下，包含中英文，用以检测正确的字符编码。

```
开始 () {
    输出 ("Hello, 龙语者! loong-lang.com")
}
/*
编译命令:
lyy hello.lyy -o ..\build\ch01\hello.exe

执行命令:
..\build\ch01\hello.exe

输出结果:
Hello, 龙语者! loong-lang.com
*/
```

源码 1: ch01/hello.lyy

是否足够简单？马上把龙语言编译环境安装起来吧！

参考：附录 A：安装指南

1.2　注意事项

龙语言语法借鉴 C/C++、Java 等，有经验的用户会很快适应，但有以下特别注意事项：

1. **编译器名称、源码扩展名均为 lyy**：龙语言 (LongYuYan) 拼音首字母简写。
2. **目前仅支持 64 位的系统架构**：Linux 64 位、Windows10 以上 64 位。Windows 控制台 cmd 使用 UTF-8 显示，输入命令“chcp 65001”，输入“chcp”可查看当前 codepage。
3. **源文件采用 UTF-8 编码**：建议初学者使用 Notepad++ 或 VSCode 等编辑器，可显示转换文件编码。
4. **标点符号采用“半角/英文”**：字符串字面量（“”）里面的除外。
5. **每条语句行末的结束符分号 (;) 可选**：写上也没错，多条语句写在同一行时，‘;’分隔。
6. **‘开始’函数是执行起点**：可执行文件有且仅能有一个‘开始’函数。
7. **自定义标识符以‘$’开头**：类似 PHP。
8. **注释语法与 C 语言相同**：单行注释使用//，多行注释使用 /* */, 被注释内容不参与编译。

1.3　变量、类型、算术表达式

龙语言是静态类型编程语言，所有数据类型在编译时已经确定，若不指定类型，会进行自动类型推断，并作出假定。声明变量以‘令’关键字开头，变量运行时可修改内容。

请看源码 2：声明了 3 个变量: $ 身高: 整数，$ 体重: 实数，$ 姓名: C 类型字符串（以‘\0’结尾）。‘输出’函数是编译器内置函数，底层调用是 C 运行时的 printf 函数，{} 是占位符，将用后面对应位置的变量值替代输出。

注意: **‘$’与变量名之间不能有空格**，示例显示有空格，是该文档语法着色器不完善的地方。建议下载源码包，用文本编辑器打开对照。

示例源码 3实现同样功能，指定了变量数据类型，‘{%s}’表示使用字符串格式输出，使用方法与 C 语言 printf 中格式化语法一致，更多输出格式，请另行查阅详细文档。

```
开始 () {
    令 $ 身高 = 180
    令 $ 体重 = 65.5
    令 $ 姓名 = "Hi 小明"
    输出 (" 精神小伙: {}, {}, {}", $ 身高, $ 体重, $ 姓名)
}
/*
编译命令:
lyy var.lyy -o ..\build\ch01\var.exe

执行命令:
..\build\ch01\var.exe

输出结果:
精神小伙: 180, 65.500000, Hi 小明
*/
```

源码 2: ch01/var.lyy

```
开始 () {
    令 $ 身高: 整数 = 180
    令 $ 体重: 实数 = 65.5
    令 $ 姓名: 字符 指针 = "Hi 小明 2"
    输出 (" 精神小伙: {}, {}, {%s}", $ 身高, $ 体重, $ 姓名)
}
/*
编译命令:
lyy var2.lyy -o ..\build\ch01\var2.exe

执行命令:
..\build\ch01\var2.exe

输出结果:
精神小伙: 180, 65.500000, Hi 小明 2
*/
```

源码 3: ch01/var2.lyy

```
开始 () {
    令 $ 身高 = 180  // 单位: 厘米
    令 $ 体重 = 60 + 5.5
    令 $ 姓名 = "Hi 小明"
    // 单位: 米, $ 身高 自动转为实数 (即浮点数 64), 减掉 3.5cm 鞋高
    令 $ 净身高 = ($ 身高 - 3.5)/100.0  //(实数)
    // BMI = 体重（kg）÷ 身高（m）²
    令 $BMI = $ 体重 / ($ 净身高 * $ 净身高)
    输出 (" 精神小伙: {}, {}, {}, BMI: {}", $ 身高, $ 体重, $ 姓名, $BMI)
}
/*
编译命令:
lyy var3.lyy -o ..\build\ch01\var3.exe

执行命令:
..\build\ch01\var3.exe

输出结果:
精神小伙: 180, 65.500000, Hi 小明, BMI: 21.025769
*/
```

源码 4: ch01/var3.lyy

示例源码 4演示了简单的算术运算，加 +、减-、乘 *、除/，语法。
个人以为，在应用级别的编程，整数、实数、字符串，这三种数据类型，可以覆盖 8 成的编程问题，非常重要。
初学者刚开始，可以把龙语言编译器当成一个高度可以自定义的计算器，解决工作生活学习中的算术问题。

1.4　循环语句

程序语句执行按路径可分为三种：顺序、循环、分支。这三种基本结构可组成任意复杂的程序。若说最有用的，非“循环语句”莫属，因为计算机最擅长做重复的事情。以下演示几种基本的循环语句语法：

```
开始 () {
    循环 (令 $i = 0; $i < 5; ++$i) {
        输出 ("i = {}", $i)
    }
}
/*
编译命令:
lyy for.lyy -o ..\build\ch01\for.exe

执行命令:
..\build\ch01\for.exe

输出结果:
i = 0
i = 1
i = 2
i = 3
i = 4
*/
```

源码 5: ch01/for.lyy

示例源码 5演示了最常见的循环语句，与 C 语言的 for 循环语法一致。

```
开始 () {
    遍历 (令 $ 元素 在 0..5) {
        输出 (" 元素 = {}", $ 元素)
    }
}
/*
编译命令:
lyy foreach.lyy -o ..\build\ch01\foreach.exe

执行命令:
..\build\ch01\foreach.exe

输出结果:
元素 = 0
元素 = 1
元素 = 2
元素 = 3
元素 = 4
*/
```

源码 6: ch01/foreach.lyy

示例源码 6演示了“遍历”序列方式的循环，与其它语言的 foreach 类似，功能与上例一样。

```
开始 () {
    令 $i = 0
    循环当 ($i < 5) {
        输出 (" 当 i = {}", $i)
        ++$i
    }
}
/*
编译命令:
lyy while.lyy -o ..\build\ch01\while.exe

执行命令:
..\build\ch01\while.exe

输出结果:
当 i = 0
当 i = 1
当 i = 2
当 i = 3
当 i = 4
*/
```

源码 7: ch01/while.lyy

示例源码 7使用“循环当”关键字，相当于 C 语言的 while 语句，易于初学者理解，功能与源码 5一样，源码 5不过是一种简便语法糖。

```
开始 () {
    令 $i = 0
    执行 {
        输出 (" 执行当 i = {}", $i)
        ++$i
    } 循环当 ($i < 5)
}
/*
编译命令:
lyy dowhile.lyy -o ..\build\ch01\dowhile.exe

执行命令:
..\build\ch01\dowhile.exe

输出结果:
执行当 i = 0
执行当 i = 1
执行当 i = 2
执行当 i = 3
执行当 i = 4
*/
```

源码 8: ch01/dowhile.lyy

示例源码 8使用“执行... 循环当”关键字，相当于 C 语言的 do while 语句，功能与前者同，但它是先执行一遍，后判断条件。

```
开始 () {
    令 $ 总数 = 0
    遍历 (令 $ 元素 在 1..=1_000_000) {
        $ 总数 += $ 元素
    }
    输出 (" 总数 = {}", $ 总数)
}
/*
编译命令:
lyy loop1m.lyy -o ..\build\ch01\loop1m.exe

执行命令:
..\build\ch01\loop1m.exe

输出结果:
总数 = 500000500000
*/
```

源码 9: ch01/loop1m.lyy

之前的循环示例或许没展现出循环的强大有用之处，示例源码 9计算从 1 加到 1 百万的总数，计算机不会投机取巧，循环语句真的令它“哼哧哼哧”算了 1 百万次，而且弹指一挥间，它就给出正确结果。想象一下，如果是千万、亿次计算，还是人力可完成的吗？初学者可从该示例，领悟计算机程序的强大与必需之处。

程序解释：

..= 小于等于，形成 1 到 1 百万的闭区间。

“$ 总数 += $ 元素”：总数与元素相加，结果再保存到总数。

1.5　分支语句

分支语句，就是根据当前判断条件，选择某一条路径执行下去。

```
开始 () {
    令 $BMI = 20.1
    若 ( $BMI < 18.5 ) {
        输出 (" 过低")
    }
    否则 若 ( $BMI >= 18.5 且 $BMI < 24.0 ) {
        输出 (" 正常")
    }
    否则 若 ( $BMI >= 24.0 且 $BMI < 28.0 ) {
        输出 (" 超重")
    }
    否则 /* 若 ( $BMI >= 28.0) */ {
        输出 (" 超重")
    }
}
/*
编译命令:
lyy if_bmi.lyy -o ..\build\ch01\if_bmi.exe

执行命令:
..\build\ch01\if_bmi.exe

输出结果:
正常
*/
```

源码 10: ch01/if_bmi.lyy

1.6　别名与宏

1.6.1　别名

别名是对通用数据类型取另一个更具意义的名称，便于理解变量用途。

```
开始 () {
    别名 $ 大小 = 整数
    令 $ 量 1: $ 大小 = 10  // 变量声明，冒号：后面是数据类型
    令 $ 量 2: $ 大小 = 20
    输出 (" 量 1: {}, 量 2: {}", $ 量 1, $ 量 2)
}
/*
编译命令：
lyy alias.lyy -o ..\build\ch01\alias.exe

执行命令：
..\build\ch01\alias.exe

输出结果：
量 1: 10, 量 2: 20
*/
```

源码 11: ch01/alias.lyy

1.6.2　宏

宏类似代码模板，宏调用的位置，实现源码替换等同的功能，而方便一改全改。宏名称由（at 符号）@ 开头，宏定义既可以是表达式，也可以是语句块。但由于宏实现方式的限制，有两点注意：

1、尽可能在同一个源文件定义与调用。

2、或尽可能在同一个源文件定义所有宏，尤其不建议分散在被包含的源文件中。

```
宏 @PI = 3.14
// 宏可以带参数，与函数类似，不产生调用指定，与源码替换一致。
宏 @相加 ($x,$y) = {
    输出 (" 开始计算和值:{}+{}", $x, $y)
    输出 (" 结果:{}", $x + $y)
}

开始 () {
    输出 ("PI 值:{}", @PI)
    @相加 (10,20)
    @相加 (100,200)
}
/*
编译命令:
lyy macro.lyy -o ..\build\ch01\macro.exe

执行命令:
..\build\ch01\macro.exe

输出结果:
PI 值:3.140000
开始计算和值:10+20
结果:30
开始计算和值:100+200
结果:300
*/
```

源码 12: ch01/macro.lyy

1.7　数组与指针

1.7.1　数组

数组是连续内存上分配的多个相同类型的数据。数据类型为“[3] 整数”，可理解为“3 个整数”，访问读取时下标从 0 开始。

```
开始 () {
    令 $ 得分: [3] 整数 = [100, 95, 98]
    输出 ("{}, {}, {}", $ 得分 [0], $ 得分 [1], $ 得分 [2])
}
/*
编译命令:
lyy arr.lyy -o ..\build\ch01\arr.exe

执行命令:
..\build\ch01\arr.exe

输出结果:
100, 95, 98
*/
```

源码 13: ch01/arr.lyy

1.7.2　指针

指针是非常基础与重要的一个概念，它的含义是某个对象指向另外一个对象。指针变量是一种特殊的变量，它的内容是另一个变量或对象的内存地址。在 64 位的 CPU 中，地址线、数据线、寄存器都是 64 比特位，指针是一个 64 位的无符号整数。指针可以有多重，即指针的指针。

```
开始 () {
    令 $v1: 整数 = 123
    令 $p1: 整数 指针 = 取址 $v1
    令 $p2: 整数 指针 指针 = 取址 $p1  // 2 重指针
    输出 ("{}, {}, {}", $v1, $p1 指向, $p2 指向 指向)
    // 修改数值，所有输出值都变了，因为访问指向同一个内存地址
    $v1 = 321
    输出 ("{}, {}, {}", $v1, $p1 指向, $p2 指向 指向)
}
/*
编译命令：
lyy ptr.lyy -o ..\build\ch01\ptr.exe

执行命令：
..\build\ch01\ptr.exe

输出结果：
123, 123, 123
321, 321, 321
*/
```

源码 14: ch01/ptr.lyy

1.8　字符数组

字符数组即 C 风格字符串（C-style strings）是 C 语言中使用的一种字符串表示方法，以空字符（'\0'）结束。龙语言默认也是使用 C 风格字符串。

```
开始 () {
    令 $s1 = "Hello123"  //注意中文字符 UTF-8 编码，长度不为 1 字节
    输出 (" 数组长度:{}, 内容:{}", 字节数 ($s1), $s1)
    // 与下 2 句功能等同，但上式自动推断长度，更灵活
    令 $s2: [9] 字符 = "Hello123"
    输出 (" 数组长度:{}, 内容:{%s}", 字节数 ($s2), 取址 $s2)
}
/*
编译命令:
lyy cstr.lyy -o ..\build\ch01\cstr.exe

执行命令:
..\build\ch01\cstr.exe

输出结果:
数组长度:9, 内容:Hello123
数组长度:9, 内容:Hello123
*/
```

源码 15: ch01/cstr.lyy

1.9　函数

函数是非常基础又重要的一个概念，为实现某个功能把一个代码块封装起来，对外提供输入参数和返回值。可以把它想象成一个黑盒机器，把东西喂进去，就出来想要的东西。程序员大部分时间在调用他人写的函数。示例如下：以关键字“函数”开头，“->”指定返回值类型。

```
1
2  函数 $ 相加($x: 整数, $y: 整数) -> 整数 {
3      输出 (" 开始计算和值:{}+{}", $x, $y)
4      令 $ 和 = $x + $y
5      输出 (" 结果:{}", $ 和)
6      返回 $ 和
7  }
8
9  开始 () {
10     输出 (" 返回值: {}", $ 相加 (10,20))
11     输出 (" 返回值: {}", $ 相加 (100,200))
12 }
13 /*
14 编译命令:
15 lyy func.lyy -o ..\build\ch01\func.exe
16
17 执行命令:
18 ..\build\ch01\func.exe
19
20 输出结果:
21 开始计算和值:10+20
22 结果:30
23 返回值: 30
24 开始计算和值:100+200
25 结果:300
26 返回值: 300
27 */
```

源码 16: ch01/func.lyy

1.10　类

类是把某一些具有相同属性的事物，抽象成一个数据结构，既包含数据成员，也可以包括函数成员，逻辑上成为一个整体。面向对象编程是被广泛接受的一个概念。

```
类 $ 某类 {
    令 $f1: 整数
    令 $f2: 整数
    构造 () {  // 默认构造函数
        本尊.$f1 = 11
        $f2 = 22
    }
    $ 某类 ($f1: 整数) {  // 主构造函数
        本尊.$f1 = $f1
    }
    析构 () {
       输出 (" 析构调用: {}, {}", $f1, $f2)
    }

    函数 虚拟 $ 输出内容 () {
        输出 (" 对象内容, 字段 1:{}, 字段 2: {}", $f1, $f2)
    }
}
开始 () {
    令 $ 对象 1 = $ 某类 (100)
    $ 对象 1.$ 输出内容 ()
}
/*
编译命令:
lyy class.lyy -o ..\build\ch01\class.exe

执行命令:
..\build\ch01\class.exe

输出结果:
对象内容, 字段 1:100, 字段 2: 22
析构调用: 100, 22
*/
```

源码 17: ch01/class.lyy

第二章　类型、运算符与表达式

变量和常量是程序处理的两种基本数据对象。声明语句说明变量的名字及类型，也可以指定变量的初值。运算符指定将要进行的操作。表达式则把变量与常量组合起来生成新的值。对象的类型决定该对象可取值的集合以及可以对该对象执行的操作。

2.1　标识符

所谓标识符（Identifier），是程序员对软件系统中抽象对象的命名，如：变量名、函数名、类名、枚举名等。在程序运行过程中，可认为标识符是**内存地址**的**别名**。命名是一个很重要的决策，一个好的命名，让人很容易理解它的用途，具备自我解释性。龙语言的标识符由‘$’符号开头，后面接一个或多个中英文、数字、下划线，但不能以数字开头。

由‘$’开头的原因：

1、利于词法、语法分析器，中文语境中极大加速编译。

2、当看到 $ 开头的标识符，就知道是程序员自造，而非编译器内置。

以下都是合法的标识符，（所有标识符中间不能有空格，文档中代码块显示空格是排版系统对中文支持不完善之故）：

$变量1

$var1

$i

$_数字a1

……

不能以数字开头，如“$123 变数”不合法。

2.2　数据类型及长度

变量数据类型，决定内存分配的字节数，及它能装载的内容范围。

表 2.1: 龙语言基本数据类型.

龙语言类型	C/C++ 类型	字节数
整数、整数 64	long long	8
自然数、自然数 64	unsigned long long	8
实数、浮点数 64	double	8
整数 32	int	4
整数 16	short	2
整数 8	char	1
自然数 32、四字节、字元	unsigned int	4
自然数 16、双字节	unsigned short	2
自然数 8、字节、字符	unsigned char	1
阴阳	bool	1
浮点数 32	float	4
浮点数 16	-	2

不要被这么多类型和别名吓到，初学者先记得最重要的整数、实数。在 CPU 的眼里，其实只有两种类型，整数和浮点数。
当声明一个未定指类型的变量，并初始为**没有小数点**的数字时，默认为**整数**类型。
当声明一个未定指类型的变量，并初始为**有小数点**的数字时，默认为**实数**类型。
浮点数规则比较复杂，有专门的浮点运算单元。而更基础的观点看，数据类型只有阴阳（比特 0 或 1）。
在分配内存变量时，阴阳类型占 1 字节。
优秀的程序员，要对自己分配的内存心里有数，精确到字节、比特。

2.3　常量

常量的值一经声明，之后不可改变。

2.3.1　字面值

字面值 (Literal)，即字面表示值。示例ch01/var.lyy中，180、65.5、”Hi 小明”，就是字面值常量。字面值直接编译进目标文件的指令中，或目标文件特定只读节区。

整数字面量, 可以带后缀：’i8’ |’i16’ |’i32’ |’i64’ |’u8’ |’u16’ |’u32’ | ’u64’。i 代表整数，u 代表自然数，数字代表比特位长。
实数字面量, 可以带后缀：’f16’ | ’f32’ | ’f64’。f 代表浮点数，数字代表比特位长。
数字字面量，中间可以带下划线，方便查看数字位数。如以下都是合法数字字面量。
123i32, 123_u8, 1_000_000_i64, 3.14f32, -6_0000.28_f64

整数不同进制的表示法:
16 进制：0X、0x 开头。如：0X10, 0x11
8 进制：0O、0o 开头。如：0O10, 0o11
2 进制：0B、0b 开头。如：0B10, 0b11

```
开始 () {
    输出 (" 十进制值: {}, {}, {}", 0x10, 0o10, 0b10)
}
/*
编译命令:
lyy literal.lyy -o ..\build\ch02\literal.exe

执行命令:
..\build\ch02\literal.exe

输出结果:
十进制值: 16, 8, 2
*/
```

源码 18: ch02/literal.lyy

2.3.2　不变 const

“不变”关键字声明常量，等同 C++ 中的 const 关键字。变量和函数参数中，不变值不可通过赋值或其它语句再次修改值。

注意：“不变”修饰的只读变量（栈或堆内存中），在 CPU 眼里，其实都是可写的，这里的只读，只不过是编译器语义分析过程中的限制。

```
函数 $add($n1 : 不变 整数, $n2 : 不变 整数) -> 整数 {
    //++ $n1  //自增 1，取消注释报错："编译错误：左值为不变值，禁止修改"
    返回 $n1 + $n2
}

开始 () {
    令 不变 $num1 : 整数 = 10
    令 不变 $num2 : 整数 = 20
    //$num1 = 1  // 取消注释报错："编译错误：左值为不变值，禁止修改"
    令 $ 结果 = $add($num1, $num2)
    输出 ("{}", $ 结果)
}
/*
编译命令：
lyy const.lyy -o ..\build\ch02\const.exe

执行命令：
..\build\ch02\const.exe

输出结果：
30
*/
```

源码 19: ch02/const.lyy

2.4　枚举

枚举是将有限个常量值一一列举出来，本质上是整型常量。使用它的意义是约束接口调用的参数，或在编译阶段尽早发现错误。

```
枚举 $ 五行 { $ 金 $ 木 $ 水 $ 火 $ 土 }  // 默认继承 整数 32
枚举 $ 星期 继承 整数
{
    $ 星期日 = -1  // 初始值
    $ 星期一 = 1
    $ 星期二  // 数值递增
    $ 星期三
}
开始 () {
    输出 (" 五行字节数:{}, 金:{}, 土:{}", 字节数 ($ 五行), $ 五行.$ 金, $ 五行.$ 土 )
    令 $i = $ 星期.$ 星期一
    输出 (" 星期字节数:{}", 字节数 ($i))
    输出 (" 一:{}, 日:{}, 三:{}", $i, $ 星期.$ 星期日, $ 星期.$ 星期三)
}
/*
编译命令:
lyy enum.lyy -o ..\build\ch02\enum.exe

执行命令:
..\build\ch02\enum.exe

输出结果:
五行字节数:4, 金:0, 土:4
星期字节数:8
一:1, 日:-1, 三:3
*/
```

源码 20: ch02/enum.lyy

2.5　声明与定义

声明（declaration）使得名字为程序所知，一个文件如果想使用别处定义的名字则必须包含对那个名字的声明。而定义（definition）负责创建与名字关联的实体。变量声明规定了变量的类型和名字，在这一点上定义与之相同。但是除此之外，定义还申请存储空间，也可能会为变量赋一个初始值。

声明可以多次，且可以存在于多个源文件中，定义只能有一次。声明的对象，甚至可以由链接器关联到其它静态库、动态中的实现。

```
// 函数的向前声明，若无，则报错
函数 $add($n1 : 不变 整数, $n2 : 不变 整数) -> 整数
开始 () {
    令 不变 $num1 : 整数 = 10
    令 不变 $num2 : 整数 = 20
    令 $ 结果 = $add($num1, $num2)
    输出 ("{}", $ 结果)
}
// 函数定义，不仅有方法签名，还有具体实现
函数 $add($n1 : 不变 整数, $n2 : 不变 整数) -> 整数 {
    返回 $n1 + $n2
}
/*
编译命令：
lyy declare.lyy -o ..\build\ch02\declare.exe

执行命令：
..\build\ch02\declare.exe

输出结果：
30
*/
```

源码 21: ch02/declare.lyy

2.6　算术运算符

算术运算是计算机发明的初衷，用以替换慢速且易出错的人工计算。想想中国人用了几千年的珠算盘，不仅慢还要求人记忆口诀，效率太低。

二元算术运算符包括：+、-、*、/、%（取模运算符）。整数除法会截断结果中的小数部分。示例ch01/var3.lyy演示了简单又重要的算术运算。**模运算就是取余数**。
如：表达式 x % y 的结果是 x 除以 y 的余数，当 x 能被 y 整除时，其值为 0。示例，如果某一年的年份能被 4 整除，且不能被 100 整除；或者能被 400 整除，那么这一年就是闰年。取模运算符% 不能应用于实数类型。在有负操作数的情况下，整数除法截取的方向以及取模运算结果的符号取决于具体机器的实现。

算术运算表达式还有简写语法，如: **“$数1 = $数1 + $数2”**可以写 **“$数1 += $数2”**。类似的操作符还有: **-=** , ***=** , **/=** , **%=**

一个表达式同时有多个运算符的情况，加、减运算优先级相同，乘、除、模运算优先级相同，前一组优先级小于后者。从左到右结合运算，可用小括号 () 决定运算优先次序，与数学表达式中相同。

```
开始 () {
    令 $ 年份 = 2025 // 依次改为 2024, 2100, 2400, 重新编译运行查看结果
    若 ($ 年份 % 4 == 0 且 $ 年份 % 100 != 0
        或 $ 年份 % 400 == 0) {
        输出 ("{} 是闰年", $ 年份)
    }
    否则 {
        输出 ("{} 是平年", $ 年份)
    }
}
/*
编译命令:
lyy mod.lyy -o ..\build\ch02\mod.exe

执行命令:
..\build\ch02\mod.exe

输出结果:
2025 是平年
*/
```

源码 22: ch02/mod.lyy

2.7　关系运算符与逻辑运算符

关系运算符比较两个值的相等性。逻辑运算符判断两个值合在一起时的真假。
关系运算符包括下列几个运算符：
> 大于, >= 大于等于, < 小于, <= 小于等于
它们具有相同的优先级。优先级仅次于它们的是相等性运算符：
== 等于, != 不等于
关系运算符的优先级比算术运算符低。

逻辑运算符包括下列几个运算符：
&& 且，|| 或，！非
注意，可以用符号，也可以用中文关键字，但注意关键字前后要有空格间隔。

```
开始 () {
    令 $ 下雨 = 真
    令 $ 气温 = 15
    令 $ 小红有空 = 真
    若 (! $ 下雨 && $ 气温 > 20 || $ 小红有空) {
        输出 (" 散步")
    }
    否则 {
        输出 (" 家里蹲")
    }
}
/*
编译命令:
lyy relat_logic.lyy -o ..\build\ch02\relat_logic.exe

执行命令:
..\build\ch02\relat_logic.exe

输出结果:
散步
*/
```

源码 23: ch02/relat_logic.lyy

以下示例与上同，更改条件和使用了中文关键字，至于选择何种风格，取决于个人偏好。

```
开始 () {
    令 $ 下雨 = 假
    令 $ 气温 = 15
    令 $ 小红有空 = 假
    若 (非 $ 下雨 且 $ 气温 大于 20 或 $ 小红有空) {
        输出 (" 散步")
    }
    否则 {
        输出 (" 家里蹲")
    }
}
/*
编译命令:
lyy relat_logic2.lyy -o ..\build\ch02\relat_logic2.exe

执行命令:
..\build\ch02\relat_logic2.exe

输出结果:
家里蹲
*/
```

源码 24: ch02/relat_logic2.lyy

2.8　类型转换

运算符执行时，要求它的操作数是相同类型，若不同，龙语言会尝试自动对部分类型进行转换。自动转换是把比特位“比较窄的”操作数转换为“比较宽的”操作数，或把整数转为浮点数，都是安全的，反之则不安全。良好的编程习惯是，不要全依赖自动转换，对数的类型要严格定义，且手工转换，注意数值符号及能表示的数字范围，避免溢出。

```
开始 () {
    令 $i1 = 1_0000_i64  // 整数（8 字节）
    令 $i2 = 0_i32 // 整数 32(4 字节）
    $i2 = (整数 32)$i1 + 1  // 转为 4 字节数

    令 $f1 = 3_000.54_f64
    令 $f2: 浮点数 32
    $f2 = (浮点数 32)$f1 + 1  // 此处必须显示转换

    令 $i3: 整数 = (整数)$f1 + 100  // 实数转为整数

    输出 ("i2: {}, f2: {%.2f}, i3: {}", $i2, $f2, $i3)
}
/*
编译命令:
lyy cast.lyy -o ..\build\ch02\cast.exe

执行命令:
..\build\ch02\cast.exe

输出结果:
i2: 10001, f2: 3001.54, i3: 3100
*/
```

源码 25: ch02/cast.lyy

2.9　自增运算符与自减运算符

自增/自减运算符对变量“加一”/“减一”，“加一”/“减一”是变量常见操作，也是关键字。

若在变量前，则先加减，后返回值。

若在变量后，则先返回值，后加减。

请看示例:

```
开始 () {
    令 $n = 10
    令 $p1 = ++ $n
    输出 ("n = {},{}", $n ,$p1)  // $n 为 11
    令 $n1 = $n ++
    输出 ("n = {},{}", $n ,$n1)

    $n = 10
    令 $p2 = -- $n
    输出 ("n = {},{}", $n ,$p2)  // $n 为 9
    令 $n2 = $n --
    输出 ("n = {},{}", $n ,$n2)
}
/*
编译命令:
lyy inc_dec.lyy -o ..\build\ch02\inc_dec.exe

执行命令:
..\build\ch02\inc_dec.exe

输出结果:
n = 11,11
n = 12,11
n = 9,9
n = 8,9
*/
```

源码 26: ch02/inc_dec.lyy

以下示例使用“加一”/“减一”关键字，功能与上同。

```
开始 () {
    令 $n = 10
    令 $p1 = 加一 $n
    输出 ("n = {},{}", $n ,$p1)  // $n 为 11
    令 $n1 = $n 加一
    输出 ("n = {},{}", $n ,$n1)

    $n = 10
    令 $p2 = 减一 $n
    输出 ("n = {},{}", $n ,$p2)  // $n 为 9
    令 $n2 = $n 减一
    输出 ("n = {},{}", $n ,$n2)
}
/*
编译命令:
lyy inc_dec2.lyy -o ..\build\ch02\inc_dec2.exe

执行命令:
..\build\ch02\inc_dec2.exe

输出结果:
n = 11,11
n = 12,11
n = 9,9
n = 8,9
*/
```

源码 27: ch02/inc_dec2.lyy

2.10　按位运算符

按位运算符在底层操作时，尤其在加解密算法中，有重要运用。只能作用于整型操作数，按对应比特位运算，支持以下操作:

& 位与: 对应比特位都为 1，结果为 1，否则为 0

| 位或: 对应比特位都为 0，结果为 0，否则为 1

^ 位异或: 相同为 0，相异为 1（口诀:“异则 1”）

« 左移: 二进制模式下，往左移

» 右移: 二进制模式下，往右移

```
开始 () {
    输出 ("{}", 0b11 & 0b10)  // 结果: 0b10
    输出 ("{}", 0b11 | 0b10)  // 结果: 0b11
    输出 ("{}", 0b11 ^ 0b10)  // 结果: 0b01
    输出 ("{}", 0b1 << 2)  // 结果: 0b100
    输出 ("{}", 0b100 >> 1)  // 结果: 0b10
}
/*
编译命令:
lyy bit_ops.lyy -o ..\build\ch02\bit_ops.exe

执行命令:
..\build\ch02\bit_ops.exe

输出结果:
2
3
1
4
2
*/
```

源码 28: ch02/bit_ops.lyy

2.11　赋值运算符与表达式

赋值表达式用‘=’号，连接左值和右值，意为将右值赋予左值。与数学中方程式中等号不是一个意思，初学者注意。
形如: “$i = 123”，意为：将值 123 保存到变量 $i。
形如: “$i = $i + 123”，可简便表达为: “$i += 123”
类似的操作符还有: -= , *= , /= , %=, **=, &&=, ||=, &=, |=, ^=, «=, »=
(注意：都是英文/半角符号)

```
开始 () {
    令 $n = 2
    $n **= 10  // 2 的 10 次方
    输出 ("{}", $n)
    $n >>= 1  // 右移 1 位
    输出 ("{}", $n)
}
/*
编译命令:
lyy assign.lyy -o ..\build\ch02\assign.exe

执行命令:
..\build\ch02\assign.exe

输出结果:
1024
512
*/
```

源码 29: ch02/assign.lyy

2.12　条件表达式

条件表达式（使用三元运算符“? :”）在表达式
expr1 ? expr2 : expr3
中，首先计算 expr1，如果其值不等于 0（为真），则计算 expr2 的值，并以该值作为条件表达式的值，否则计算 expr3 的值，并以该值作为条件表达式的值。这是唯一有三个操作数的表达式。

```
开始 () {
    令 $ 分数 = 61
    令 $ 及格 = ($ 分数 >= 60) ? 真 : 假
    //令 $ 及格 = $ 分数 >= 60  // 与上同，此例可以这样表达
    输出 (" 及格?: {}", $ 及格)  // 1 代表真，0 代表假
}
/*
编译命令:
lyy cond_expr.lyy -o ..\build\ch02\cond_expr.exe

执行命令:
..\build\ch02\cond_expr.exe

输出结果:
及格?: 1
*/
```

源码 30: ch02/cond_expr.lyy

2.13　运算符优先级与求值次序

后期稳定版再补全此节。参考 C/C++ 运算符优先级，大致相似。若不确定，使用小括号括起来，优先求值。

https://zh.cppreference.com/w/cpp/language/operator_precedence.html

第三章　控制流

程序执行时，机器指令被解释成不同电流依次飞速流经 CPU，可以把它们类比成水流。控制流，则是控制程序根据输入参数或外部环境变量，选择预定义的不同的**分支**执行或**循环**执行。

3.1　语句与程序块

简单表达式，如赋值、自增，可以理解成独立语句。一个或多个表达式，可组成复杂语句，如“若-否则”、“循环”语句。多个语句，可组成程序块，由花括号“{”与“}”包围。**每条语句以换行符或“;”结束。**

程序块拥有独立的变量声明环境，允许与上层环境出现同名的但独立的变量，这是与其它很多语言不同的地方，更灵活，但同时需要更注意。

```
开始 () {
    令 $a = 10
    令 $b = 20
    输出 ("a={}, b={}", $a, $b)
    {
        令 $a = 100  // 同名但独立的变量
        $b = 200     // 可使用上层环境变量
        输出 (" 语句块中: a={}, b={}", $a, $b)
    }
    输出 ("a={}, b={}", $a, $b)
}
/*
编译命令:
lyy block.lyy -o ..\build\ch03\block.exe

执行命令:
..\build\ch03\block.exe

输出结果:
a=10, b=20
语句块中: a=100, b=200
a=10, b=200
*/
```

源码 31: ch03/block.lyy

3.2　“若-否则”语句

该语句根据条件判断结果的真假，选择不同的分支往下执行。

请参看第一章ch01/if_bmi.lyy示例。

3.3　“匹配”语句

匹配语句是一种多路判定语句，它测试表达式是否与一些**常量整数值**中的某一个值匹配，并执行相应的分支动作。相当于其它语言的 switch 语句，且支持返回值。最后下划线“_”代表上面的匹配值之外的所有情况。

```
开始 () {
    令 $i = 30  // 修改成其他值，重新编译执行查看结果
    令 $ 结果 = 匹配 ($i) {
        当 1 => 11   // => 后面是返回值
        当 2 或 20 => 22  //  多个匹配值
        当 _ => 输出 (" 其他情况"); 33  // 多条语句写在同一行，最后是返回值
    }
    输出 ("{}", $ 结果)
}
/*
编译命令：
lyy match.lyy -o ..\build\ch03\match.exe

执行命令：
..\build\ch03\match.exe

输出结果：
其他情况
33
*/
```

源码 32: ch03/match.lyy

3.4　“循环”语句

循环语句第一章已有示例，这里详细解释语法。

3.4.1　循环

```
开始 () {
    循环 (令 $i = 0; $i < 5; ++$i) {
        输出 ("i = {}", $i)
    }
}
/*
编译命令:
lyy for.lyy -o ..\build\ch01\for.exe

执行命令:
..\build\ch01\for.exe

输出结果:
i = 0
i = 1
i = 2
i = 3
i = 4
*/
```

源码 33: ch01/for.lyy

语法结构含义如下:

循环 (初始动作; 判断条件; 迭代动作)

{

循环体语句块

}

先执行初始动作，然后执行判断条件，若条件成立，则执行循环体，最后再执行迭代动作。然后再执行判断条件……，真到判断条件不成立，则退出循环。

3.4.2　循环当

参考第一章示例ch01/while.lyy，这个含义是最易理解的。

3.4.3　执行…循环当

参考第一章示例ch01/dowhile.lyy。

3.4.4　遍历

参考第一章示例ch01/foreach.lyy。这里再演示其它用法。
此示例演降序循环，“:-1”代表步长为-1，迭代每次减 1。

```
开始 () {
    遍历 (令 $ 元素 在 5..0:-1) {
        输出 (" 元素 = {}", $ 元素)
    }
}
/*
编译命令:
lyy foreach_dec.lyy -o ..\build\ch01\foreach_dec.exe

执行命令:
..\build\ch01\foreach_dec.exe

输出结果:
元素 = 5
元素 = 4
元素 = 3
元素 = 2
元素 = 1
*/
```

源码 34: ch03/foreach_dec.lyy

也支持实数及变量作为循环边界，以下示例步长为 0.2，“遍历”语法**要求步长为常数**，因为要在编译时确定循环迭代方向。

初学者要特别注意，使用实数时，判断相等性不能直接使用“等于”，由于误差，内在编码未必相等。一般作法是使用“**绝对值 (实数1 - 实数2) < 误差值**”这个方式。此例中，右边界 r2 - 0.001，一般偏移步长小 2 个数量级以上的误差值，即安全。

```
开始 () {
    令 $r1 = 9.0
    令 $r2 = 10.0 - 0.001
    遍历 (令 $r 在 $r1 .. $r2 : 0.2) {
        输出 ("r = {%.1f}", $r)
    }
}
/*
编译命令:
lyy foreach_real.lyy -o ..\build\ch01\foreach_real.exe

执行命令:
..\build\ch01\foreach_real.exe

输出结果:
r = 9.0
r = 9.2
r = 9.4
r = 9.6
r = 9.8
*/
```

源码 35: ch03/foreach_real.lyy

3.5　中断继续

在循环过程中，“中断”可以跳出当前循环，“继续”可以跳过循环体，转到下一次迭代。

```
开始 () {
    遍历 (令 $i 在 1 ..= 10 : 1) {
        若 ($i == 6) { 中断; }  // 花括号 {} 和 ; 不能少
        若 ($i == 4) {
            继续  // 或者换行
        }
        输出 ("{}", $i)
    }
}
/*
编译命令:
lyy break_continue.lyy -o ..\build\ch03\break_continue.exe

执行命令:
..\build\ch03\break_continue.exe

输出结果:
1
2
3
5
*/
```

源码 36: ch03/break_continue.lyy

3.6　跳转标签

在函数内，可跳转到自定义标签位置。谨慎使用，可能造成逻辑混乱或内存泄露。

```
开始 () {
    令 $Y = 真
    标签 $L1
    输出 (" 步骤 1")
    若 ( $Y ) {
        $Y = 假
        跳转 $L2
    }
    否则 {
        跳转 $L3
    }
    输出 (" 步骤 2")
    标签 $L2
    输出 (" 步骤 3")
    跳转 $L1
    标签 $L3
    输出 (" 退出")
}
/*
编译命令:
lyy goto.lyy -o ..\build\ch03\goto.exe

执行命令:
..\build\ch03\goto.exe

输出结果:
步骤 1
步骤 3
步骤 1
退出
*/
```

源码 37: ch03/goto.lyy

第四章　函数

函数可以把大的计算任务分解成若干个较小的任务，程序设计人员可以基于函数进一步构造程序，而不需要重新编写一些代码。一个设计得当的函数可以把程序中不需要了解的具体操作细节隐藏起来，从而使整个程序结构更加清晰，并降低修改程序的难度。随着时间推移需求的变化，每个函数在保证接口不变的情况下，可以单独修改或替换函数实现，而不影响其它。

总之，函数带来的优势有三：**封装与信息隐藏**、**复用**、**独立变化**。基于函数调用机制的编程范式，称之为**过程式程序设计**。

4.1　函数的基本知识

龙语言**不支持**函数（方法）的**重载**，即不支持相同函数名，不同参数。

函数的定义形式如下：

```
函数    $函数名 (参数声明列表) -> 返回值类型
{
    声明和语句
}
```

参数声明列表形式如下：

```
$参数名: 参数类型, ...
```

当函数没有返回值时，“-> 返回值类型”可省略，或可写“-> 空”。

请参看第一章ch01/func.lyy示例。

4.2　外部变量与函数

函数体内的变量称为**局部变量**，与全局函数同一级的顶层变量，称为**全局变量**。全局变量可以是外部变量，它引用另一个编译单元（源文件）的全局变量。外部函数，引用来自另一个编译单元的函数。

```
令 $GI: 整数 = 123

函数 $ 实数和 ($r1: 实数, $r2: 实数) -> 实数
{
    返回 $r1 + $r2
}
```

源码 38: ch04/extern_decl.lyy

```
// 外部变量声明
外部 令 $GI: 整数
// 外部函数声明
外部 函数 $ 实数和 ($r1: 实数, $r2: 实数) -> 实数

开始 () {
    输出 ("{}", $GI)
    输出 ("{%.1f}", $ 实数和 (1.1, 2.2))
}
/*
编译命令:
lyy extern_decl.lyy extern_main.lyy -o ..\build\ch04\extern_main.exe

执行命令:
..\build\ch04\extern_main.exe

输出结果:
123
3.3
*/
```

源码 39: ch04/extern_main.lyy

4.3　递归

递归调用，即函数可以**直接或间接调用自身**。这是一种很重要，对初学者又比较难理解的解决问题思路。把一个大问题，分解成若干小问题，在小问题上求解，再合并解集，得出最终问题的结果。所谓“递归”，即“**层次递进，依次回归**”。函数调用自身，每次调用，即生成一个栈帧，包含函数参数、局部变量、返回地址等信息。递进达到小问

题，可以明显得到答案后，依次返回，弹出栈帧，返回结果。递归一定要有合理的返回条件，否则无穷递归会耗尽栈内存，导致栈溢出 (StackOverFlow)。初学者可能需要花点时间领悟，理解递归之后，还需要再理解“尾递归”，即不产生栈帧，减少内存消耗和栈溢出风险，这里不再举例。斐波那契数列（Fibonacci sequence），前 2 个数为 1，后面每个数等于前 2 个数之和，可作为递归一个简单示例。

```
函数 $fib($n: 整数) -> 整数
{
    若 ($n <= 2) {
        返回 1
    }
    否则 {
        返回 $fib($n-1) + $fib($n-2)
    }
}
开始 () {
    令 $ 位数 = 6
    输出 ("Fib 数列第 {} 个数是 {}", $ 位数, $fib($ 位数))
}
/*
编译命令:
lyy fib.lyy -o ..\build\ch04\fib.exe

执行命令:
..\build\ch04\fib.exe

输出结果:
Fib 数列第 6 个数是 8
*/
```

源码 40: ch04/fib.lyy

4.4　开始函数

“开始”函数，是可执行程序的起点，相当于 C/C++ 里的 main 函数，只允许出现一个，否则会出现链接错误。在操作系统里运行程序时，会传递参数给开始函数，执行结束后，会返回一个整数 32 的值给操作系统，表明客户程序是否正常结束，0 代表正常，其它值代表异常结束。龙语言最简化的开始函数，“开始 ( ) { }”，默认返回 0，其

实背后已经作了默认处理。

C/C++ 的开始函数叫 **main 函数**，全签名形式如下，功能最全面。有了这个函数，就不必有“开始”函数，它只是一个简化版 main 函数。main 函数可以接收来自操作系统的参数，并返回自定义状态码（整数 32）给操作系统。

```
函数 $main($argc: 整数 32, $argv: 字符 指针 指针) -> 整数 32
{
    输出 (" 参数个数:{}", argc)
    输出 (" 第一个参数: {%s}", $argv 指向)
    输出 (" 第二个参数: {%s}", $argv 偏移 1 指向)
    返回 0
}
/*
编译命令:
lyy main.lyy -o ..\build\ch04\main.exe

执行命令:
..\build\ch04\main.exe loong-lang.com

输出结果:
参数个数:2
第一个参数: ..\build\ch04\main.exe
第二个参数: loong-lang.com
*/
```

源码 41: ch04/main.lyy

第五章　指针与数组

指针是一种保存**变量地址**的变量。现代流行的操作系统均是使用 64 位平坦内存模式，变量逻辑地址是 64 位无符号整数，相当于龙语言的**自然数**类型。龙语言有个“空”类型，对应 C/C++ 的 void 类型，而“空指针”的含义是该指针指向的对象类型为空（即任意类型），“空指针”是通用指针类型。

在应用层的程序，都假定它拥有独立平坦地址空间，所使用的地址均是逻辑地址。至于同时运行多个程序，保证地址不冲突，是操作系统内核解决的问题，操作系统会将逻辑地址映射到真正的物理内存地址，应用层程序员无需操心这些细节。

5.1　指针与地址

想象计算机内存是一块方方正正的人类小区，其中一座是小明家的房子，作为快递小哥的我要送一个件到小明家，但没有他家的直接地址，但我知小红家知道小明家地址，于是先从小红家取到小明家的地址，再往小明家送达快递成功。

```
开始 () {
    令 $ 小明家 = 0
    //令 $ 小明家: 整数 = 0  // 自动推断, 与上一行同
    令 $ 小红家 = 取址 $ 小明家
    //令 $ 小红家: 整数 指针 = 取址 $ 小明家  //与上一行同
    $ 小红家 指向 = 1944  // 送去编号为 1944 快递
    输出 ("{}, {}", $ 小明家, $ 小红家 指向)  // 查看快递号, 两种方式访问一样
}
/*
编译命令:
lyy addr.lyy -o ..\build\ch05\addr.exe

执行命令:
..\build\ch05\addr.exe

输出结果:
1944, 1944
*/
```

源码 42: ch05/addr.lyy

5.2　指针与函数参数

龙语言是以**传值的方式将参数值传递给被调用函数**。因此，被调用函数不能直接修改主调函数中变最的值。示例，“交换”函数来交换变量的值。

```
// 错误的示例
函数 $ 交换 1($f1: 实数, $f2: 实数) -> 空
{
    令 $t = $f1
    $f1 = $f2
    $f2 = $t
}
// 正确的示例
函数 $ 交换 2($p1: 实数 指针, $p2: 实数 指针) // -> 空
{
    令 $t = $p1 指向
    $p1 指向 = $p2 指向
    $p2 指向 = $t
}
开始 () {
    令 $x = 1.1
    令 $y = 2.2
    $ 交换 1($x, $y)
    输出 (" 交换 1 后: {%.1f},{%.1f}", $x, $y)  // 值未变
    $ 交换 2(取址 $x, 取址 $y)  // 传递指针值
    输出 (" 交换 2 后: {%.1f},{%.1f}", $x, $y)  // 值已交换
}
/*
编译命令:
lyy swap.lyy -o ..\build\ch05\swap.exe

执行命令:
..\build\ch05\swap.exe

输出结果:
交换 1 后: 1.1,2.2
交换 2 后: 2.2,1.1
*/
```

源码 43: ch05/swap.lyy

5.3　指针与数组

数组是一组类型相同的数值，保存在相邻的一块内存中，下标从 0 开始。由于相邻的特性，特别适合用指针来操作，只需要定位到某一个元素（通常是开头），就很方便通过“偏移”操作定位到其它任意元素，但注意不能超出数组长度限制。

```
开始 () {
    令 $a = [ 10, 20, 30, 40 ]  // 一维 [4] 整数 数组
    输出 (" 首尾值: {},{}", $a[0], $a[3])  // 下标最小为 0, 最大为 3
    令 $pa = 取址 $a  // 指针指向数组开始
    令 $pe = $pa 偏移 3  // 尾元素指针
    输出 (" 首尾值: {},{}", $pa 指向, $pe 指向)  // 使用 指针 偏移 操作定位到最后元素
    输出 (" 倒数第二元素: {}", $pe 偏移 -1 指向)  // 偏移可以是负整数, 往前定位
}
/*
编译命令:
lyy array.lyy -o ..\build\ch05\array.exe

执行命令:
..\build\ch05\array.exe

输出结果:
首尾值: 10,40
首尾值: 10,40
倒数第二元素: 30
*/
```

源码 44: ch05/array.lyy

5.4　地址算术运算

之前说过地址值不过是个 64 比特位 (8 字节) 的自然数，当然同样也支持算术运算。如果 $p 是一个指向数组中某个元素的指针，那么 $p ++ 将对 p 进行自增运算并指向下一个元素，而 $p += $i 将对 $p 进行加 $i 的增量运算，使其指向指针 $p 当前所指向的元素之后的第 $i 个元素。这类运算是指针或地址算术运算中最简单的形式。与指针“偏移”操作类型，但指针算术操作修改指针变量值。

```
开始 () {
    令 $a = [ 10, 20, 30, 40 ]  // 一维 [4] 整数 数组
    令 $pa: 整数 指针 = 取址 $a  // 指针指向数组开始
    输出 (" 首元素: {}", $pa 指向)
    ++ $pa  // 移到下一个
    输出 (" 二元素: {}", $pa 指向)
    $pa --  // 回到上一个
    输出 (" 首元素: {}", $pa 指向)
    $pa += 3  // 后移 3 个元素
    输出 (" 尾元素: {}", $pa 指向)
}
/*
编译命令:
lyy ptr_arith.lyy -o ..\build\ch05\ptr_arith.exe

执行命令:
..\build\ch05\ptr_arith.exe

输出结果:
首元素: 10
二元素: 20
首元素: 10
尾元素: 40
*/
```

源码 45: ch05/ptr_arith.lyy

5.5　字符指针与函数

字符串字面量是一个字符数组，例如：

”loong-lang.com”

在字符串的内部表示中，字符数组以空字符’\0’ 结尾，所以，程序可以通过检查空字符找到字符数组的结尾。字符串常量占据的存储单元数也因此比双引号内的字符数大 1。

```
开始 () {
    令 $msg = "loong-lang.com"  // 最简式
    //$msg[0] = 'L'  //可修改单个字符
    输出 (" 字符数组大小:{}, 内容:{}", 字节数 ($msg), $msg)
    令 $msg2: [15] 字符 = "loong-lang.com"  // 字符数组
    $msg2[0] = 'L'  //可修改单个字符
    输出 (" 字符数组大小:{}, 内容:{%s}", 字节数 ($msg2), 取址 $msg2)
    令 $pmsg: 字符 指针 = "loong-lang.com"  // 指针直接指向字面量串
    // %s 把参数当字符串输出
    输出 (" 字符指针大小:{}, 内容:{%s}", 字节数 ($pmsg), $pmsg)
}
/*
编译命令:
lyy char_ptr.lyy -o ..\build\ch05\char_ptr.exe

执行命令:
..\build\ch05\char_ptr.exe

输出结果:
字符数组大小:15, 内容:loong-lang.com
字符数组大小:15, 内容:Loong-lang.com
字符指针大小:8, 内容:loong-lang.com
*/
```

源码 46: ch05/char_ptr.lyy

以下示例通过字符指针，复制一个字符串到目的串。

```
函数 $ 串复制($dst: 字符 指针, $src: 字符 指针) {
    循环当 ($dst 指向 = $src 指向) {
    //循环当 (($dst 指向 = $src 指向) != '\0') {
        ++ $dst
        ++ $src
    }
}
开始 () {
    令 $ 来源 = "loong-lang.com"
    令 $ 目的: [20] 字符  // 目的字符数组, 足够容纳源串
    $ 串复制 (取址 $ 目的, 取址 $ 来源)
    输出 (" 字符数组大小:{}, 内容:{}", 字节数 ($ 来源), $ 来源)
    输出 (" 字符数组大小:{}, 内容:{%s}", 字节数 ($ 目的), 取址 $ 目的)
}
/*
编译命令:
lyy str_cpy.lyy -o ..\build\ch05\str_cpy.exe

执行命令:
..\build\ch05\str_cpy.exe

输出结果:
字符数组大小:15, 内容:loong-lang.com
字符数组大小:20, 内容:loong-lang.com
*/
```

源码 47: ch05/str_cpy.lyy

5.6　指针数组以及指向指针的指针

由于指针本身也是变量，所以它们也可以像其它变量一样存储在数组中。指向指针的指针示例，参考第一章：ch01/ptr.lyy。

```
开始 () {
    令 $pmsg: [] 字符 指针 = [ "loong-lang.com", " 龙语者" ]  // 指针直接指向字面量串
    输出 ("{%s}", $pmsg[0] 指向)
    输出 ("{%s}", $pmsg[1] 指向)
}
/*
编译命令:
lyy ptr_arr.lyy -o ..\build\ch05\ptr_arr.exe

执行命令:
..\build\ch05\ptr_arr.exe

输出结果:
loong-lang.com
龙语者
*/
```

源码 48: ch05/ptr_arr.lyy

5.7　多维数组

龙语言提供了类似于矩阵的多维数组。一维数组相当于一行，二维数组相当于多行多列的平面，三维数组相当二维之上再有深度，更多维数组很少使用。多维数组各元素在内存中都是连续的。以二维数组为例：

```
开始 () {
    令 $a: [2][3] 整数 = [  // 二维 2 行 3 列 整数 数组
    //令 $a: [][] 整数 = [  // 也可以不写行列数，自动推断
        [ 11, 12, 13 ],
        [ 21, 22, 23 ],
    ]
    // 4 个角上的数值，\n 代表换行
    输出 ("{}, {}\n{}, {}", $a[0][0], $a[0][2], $a[1][0], $a[1][2])
}
/*
编译命令：
lyy array2d.lyy -o ..\build\ch05\array2d.exe

执行命令：
..\build\ch05\array2d.exe

输出结果：
11, 13
21, 23
*/
```

源码 49: ch05/array2d.lyy

5.8　命令行参数

在支持 C 语言的环境中，可以在程序开始执行时将命令行参数传递给程序。调用主函数 main 时，它带有两个参数。第一个参数（习惯上称为 argc，用于参数计数）的值表示运行程序时命令行中参数的数目；第二个参数（称为 argv，用于参数向量）是一个指向字符串数组的指针，其中每个字符串对应一个参数。我们通常用多级指针处理这些字符串。最简单的例子是程序 echo，它将命令行参数回显在屏幕上的一行中，其中命令行中各参数之间用空格隔开。也就是说，命令

echo hello, world

将打印下列输出：

hello, world

按照 C 语言的约定，argv[0] 的值是启动该程序的程序名，因此 argc 的值至少为 1。如果 argc 的值为 1，则说明程序名后面没有命令行参数。在上面的例子中，argc 的值为 3，argv[0]、argv[1] 和 argv[2] 的值分别为“echo”、“hello,”，以及“world”。第一个可选参数为 argv[1]，而最后一个可选参数为 argv[argc-1]。另外，ANSI 标准要求 argv[argc] 的值必须为一空指针 (参见下图)。

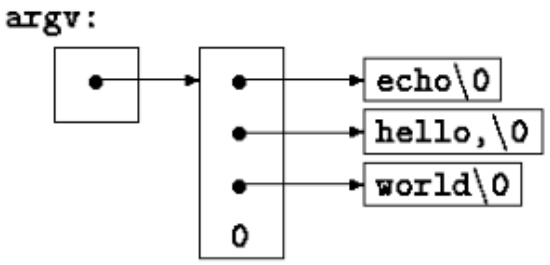


图 5.1: argv

龙语言遵循 C 语言的约定，不过 argv 使用“$argv: 字符　指针　指针”类型。以下示例与第四章main 函数示例类似。

```
函数 $main($argc: 整数 32, $argv: 字符 指针 指针) -> 整数 32
{
    循环 (令 $i=0; $i < $argc; ++ $i){
        输出 (" 第 {} 个参数值: {%s}", $i, $argv 偏移 $i 指向)
    }
    返回 0
}
/*
编译命令:
lyy argv.lyy -o ..\build\ch05\argv.exe

执行命令:
..\build\ch05\argv.exe loong-lang.com so good 123

输出结果:
第 0 个参数值: ..\build\ch05\argv.exe
第 1 个参数值: loong-lang.com
第 2 个参数值: so
第 3 个参数值: good
第 4 个参数值: 123
*/
```

源码 50: ch05/argv.lyy

5.9　指向函数的指针

如前所述，指针变量值是一个自然数 (64 比特位) 表示的地址，而程序运行时，函数也有一个内存地址，所以指针同样可以指向函数。

```
函数 $ 和($n1: 整数, $n2: 整数) -> 整数 {
    返回 $n1 + $n2
}
函数 $ 差($n1: 整数, $n2: 整数) -> 整数 {
    返回 $n1 - $n2
}
// 定义函数指针别名, "$_ 名" 可以任意, 占位作用
别名 $ 运算式 = (函数 $_ 名 ($n1: 整数, $n2: 整数) -> 整数) 指针
开始 () {
    令 $pfn: $ 运算式 // 函数指针
    $pfn = $ 和  // 指向 “和” 函数
    输出 ("{}", $pfn(10, 20))  // 调用函数指针
    $pfn = $ 差
    输出 ("{}", $pfn(10, 20))  // 调用函数指针
}
/*
编译命令:
lyy ptr_fn.lyy -o ..\build\ch05\ptr_fn.exe

执行命令:
..\build\ch05\ptr_fn.exe

输出结果:
30
-10
*/
```

源码 51: ch05/ptr_fn.lyy

第六章　类

面向对象 (Object Oriented) 是软件开发方法，一种编程范式。它允许程序员自定义数据类型，组织感兴趣的复杂的数据属性、方法为一个整体。面向对象有三大特点（封装，继承，多态）。封装为了信息隐藏，调用者不需要关注类内部现实细节，只需要关注提供的接口功能。继承方便代码重用，不仅可以继承父类数据成员，也可继承方法（函数）成员。多态表明在运行时，同样类型的对象指针，却有不同的功能表现，通过虚拟方法实现。

龙语言的面向对象功能还处于实验阶段，后期可能更改语义实现。

6.1　继承

龙语言支持单独继承，不支持如 C++ 的多继承。继承示例如下，“本尊”引用本身指针，相当于 C++ 语言中的 this；“父尊”引用父类指针，相当于 Java 语言中的 super。指针和对象访问成员，均是操作符“.”（点号），指针会自动解引用。

```
类 $ 人类 {
    令 $ 年龄: 整数  // 数据成员
    令 $ 体重: 实数
    函数 $ 打印 () {  // 方法成员
        输出 (" 某人年龄: {}, 体重:{%.1f}", $ 年龄, $ 体重)
    }
}
类 $ 龙语者
    继承 $ 人类
{
    令 $ 学号: 整数
    函数 $ 打印 () {
        父尊.$ 打印 ()  // 调用父类方法
        输出 (" 龙语者学号: {}", 本尊.$ 学号)
    }
}
开始 () {
    令 $ 路人甲: $ 人类
    $ 路人甲.$ 年龄 = 18
    $ 路人甲.$ 体重 = 60.6
    $ 路人甲.$ 打印 ()

    令 $ 小明: $ 龙语者
    $ 小明.$ 年龄 = 30  //继承的父类数据成员
    $ 小明.$ 体重 = 70.7
    $ 小明.$ 学号 = 2025
    $ 小明.$ 打印 ()
}
/*
编译命令:
lyy inherit.lyy -o ..\build\ch06\inherit.exe

执行命令:
..\build\ch06\inherit.exe

输出结果:
某人年龄: 18, 体重:60.6
某人年龄: 30, 体重:70.7
龙语者学号: 2025
*/
```

源码 52: ch06/inherit.lyy

6.2　多态

多态要求两个条件：一、使用对象指针；二、调用虚拟方法。

```
类 $ 人类 {
    函数 虚拟 $ 工作 () {  // 虚拟关键字指定“虚拟方法”
        输出 (" 散步思考")
    }
}
类 $ 龙语者
    继承 $ 人类
{
    函数 虚拟 $ 工作 () {
        输出 (" 写下 Hello, Loong Lang")
    }
}
// 全局函数，传进不同的对象指针，调用不同的函数
函数 $ 上班 ($ 某人: $ 人类 指针) {
    $ 某人.$ 工作 ()
}
开始 () {
    令 $ 路人甲: $ 人类
    令 $ 小明: $ 龙语者
    $ 上班 (取址 $ 路人甲)
    $ 上班 (取址 $ 小明)  // 取地址传入函数，自动转为父类指针
}
/*
编译命令:
lyy dynamic.lyy -o ..\build\ch06\dynamic.exe

执行命令:
..\build\ch06\dynamic.exe

输出结果:
散步思考
写下 Hello, Loong Lang
*/
```

源码 53: ch06/dynamic.lyy

6.3　构造与析构

构造函数、析构函数分别是创建对象后、销毁对象前自动调用的函数，用来初始或释放占用的系统资源。若类没有定义这两个函数，也会自动生成这两个函数，包含空实现。子类构造会先调用父构造，子类析构后再调用父析构。

```
类 $ 人类 {
    构造 () { 输出 (" 构造 人类"); }  // 写到同一行、节省空间。语句后面加 ;
    析构 () { 输出 (" 析构 人类"); }
}
类 $ 龙语者 继承 $ 人类
{
    构造 () {
        输出 (" 构造 龙语者")
    }
    析构 () {
        输出 (" 析构 龙语者")
    }
}
开始 () {
    令 $ 小明: $ 龙语者
    输出 ("---")
}
/*
编译命令:
lyy ctor_dtor.lyy -o ..\build\ch06\ctor_dtor.exe

执行命令:
..\build\ch06\ctor_dtor.exe

输出结果:
构造 人类
构造 龙语者
---
析构 龙语者
析构 人类
*/
```

源码 54: ch06/ctor_dtor.lyy

6.4　注解“# 数据”

“注解”用于附加到函数或类上面，提供额外控制信息。有时我们想要得到一个纯粹的数据类，不添加任何默认函数（构造、析构），可以给类添加“# 数据”注解，得到类似 C 语言的结构体。注解以‘#’开头，后文还会介绍其它注解。示例如下，表示平面上的点 (x,y)：

```
类 $ 点
    # 数据  // 注解，去掉与否，比较编译运行结果
{
    令 $x: 整数
    令 $y: 整数
    // 以下不再需要，仅供演示，不再默认自动生成，也不会自动调用，相当于普通函数
    构造 () { 输出 (" 构造 点"); }
    析构 () { 输出 (" 析构 点"); }
}
开始 () {
    令 $p: $ 点
    $p.$x = 19
    $p.$y = 44
    输出 ("({},{})", $p.$x, $p.$y)
}
/*
编译命令:
lyy point.lyy -o ..\build\ch06\point.exe

执行命令:
..\build\ch06\point.exe

输出结果:
(19,44)
*/
```

源码 55: ch06/point.lyy

6.5　主构造函数

主构造函数是与类名相同的函数，方便初始对象并提供参数（默认构造无参数）。由于龙语言不支持函数重载，所以只能有一个主构造函数，故定义参数需要慎重。

```
类 $ 人类 {
    令 $ 年龄: 整数  // 数据成员
    令 $ 体重: 实数
    $ 人类 ($_ 年龄: 整数, $_ 体重: 实数) {  //主构造函数
        $ 年龄 = $_ 年龄
        $ 体重 = $_ 体重
    }
    函数 $ 打印 () {  // 方法成员
        输出 (" 某人年龄: {}, 体重:{%.1f}", $ 年龄, $ 体重)
    }
}
类 $ 龙语者
    继承 $ 人类
{
    令 $ 学号: 整数
    $ 龙语者 ($_ 年龄: 整数, $_ 体重: 实数, $_ 学号: 整数) { //主构造函数
        父尊 ($_ 年龄, $_ 体重)  // 父类主构造函数
        $ 学号 = $_ 学号
    }
    函数 $ 打印 () {
        父尊.$ 打印 ()  // 调用父类方法
        输出 (" 龙语者学号: {}", 本尊.$ 学号)
    }
}
开始 () {
    令 $ 路人甲 = $ 人类 (18, 60.6)
    $ 路人甲.$ 打印 ()

    令 $ 小明 = $ 龙语者 (30, 70.7, 2025)
    $ 小明.$ 打印 ()
}
/*
编译命令:
lyy main_ctor.lyy -o ..\build\ch06\main_ctor.exe

执行命令:
..\build\ch06\main_ctor.exe

输出结果:
某人年龄: 18, 体重:60.6
某人年龄: 30, 体重:70.7
龙语者学号: 2025
*/
```

源码 56: ch06/main_ctor.lyy

6.6　类操作符重载

类操作符重载，允许自定义类像内置数据类型一样，用操作符去运算，C++ 编程语言有同样特性。以下示例，实现“操作符 ()”，使对象可以像函数一样调用。

支持重载的操作符如下：

`[]`, `()`, `!`, `+`, `-`, `**`, `*`, `/`, `%`, `<<`, `>>`, `>`, `<`, `<=`, `>=`, `==`, `!=`, `&`, `^`, `|`

由于个人时间精力有限，以上操作符未完整测试，若使用过程发现 bug 或需要更多操作符，请来邮件。

```
类 $ 点
{
    令 $x: 整数
    令 $y: 整数
    函数 操作符 () ($ 偏移: 整数) {  // 也可无参数
        输出 ("x: {}, y: {}", $x + $ 偏移, $y + $ 偏移)
    }
}
开始 () {
    令 $p: $ 点
    $p.$x = 10
    $p.$y = 20
    $p(5)  // 对象像函数一样调用
}
/*
编译命令:
lyy op_parens.lyy -o ..\build\ch06\op_parens.exe

执行命令:
..\build\ch06\op_parens.exe

输出结果:
x: 15, y: 25
*/
```

源码 57: ch06/op_parens.lyy

类操作符函数，与类的普通成员函数编译后生成的代码是类似的，只不过是调用方式的差别。如表达式

“$对象1 操作符x $对象2”

相当于成员方法调用

“$对象1.操作符x($对象2)”

以下示例演示加法操作符：

```
类 $ 点
{
    令 $x: 整数
    令 $y: 整数
    $ 点 ($_x: 整数, $_y: 整数) {
        $x = $_x
        $y = $_y
    }
    函数 操作符 + ($ 其它: $ 点) -> $ 点 {
        令 $p: $ 点
        $p.$x = 本尊.$x + $ 其它.$x
        $p.$y = 本尊.$y + $ 其它.$y
        返回 $p
    }
}
开始 () {
    令 $p1 =  $ 点 (10, 20)
    令 $p2 =  $ 点 (100, 200)
    令 $p3 = $p1 + $p2
    输出 ("x: {}, y: {}", $p3.$x , $p3.$y)
}
/*
编译命令:
lyy op_add.lyy -o ..\build\ch06\op_add.exe

执行命令:
..\build\ch06\op_add.exe

输出结果:
x: 110, y: 220
*/
```

源码 58: ch06/op_add.lyy

6.7　栈对象与堆对象

前文说函数调用时，产生栈帧，函数内变量（对象）位于栈内存，同样内存地址空间有一段内存称为堆内存，也可在其中分配对象。栈内存在栈帧弹出时，自动释放对象所占内存，但栈内存大小有限，适合小对象自动分配。堆内存需要手动分配对象，及释放，但适合大对象。龙语言堆内存对象，使用 C 语言的 malloc 函数实现，使用关键字“新建”、“删除”。

```
类 $ 龙语者 {
    构造 () { 输出 (" 构造 龙语者"); }
    析构 () { 输出 (" 析构 龙语者"); }
}
开始 () {
    令 $pi = 新建 整数
    //令 $pi: 整数 指针 = 新建 整数  //与上同
    $pi 指向 = 123
    输出 ("{}", $pi 指向)
    删除 $pi
    // 堆内存分配对象，栈内存保存了对象指针
    令 $ 小明 = 新建 $ 龙语者
    //令 $ 小明: $ 龙语者 指针  = 新建 $ 龙语者  //与上同
    删除 $ 小明
}
/*
编译命令:
lyy heap.lyy -o ..\build\ch06\heap.exe

执行命令:
..\build\ch06\heap.exe

输出结果:
123
构造 龙语者
析构 龙语者
*/
```

源码 59: ch06/heap.lyy

6.8　包名与包含

当软件工程较大时，可能由多人合作分工完成，或使用别人已经做好的组件，这样可能导致命名冲突。比如，出现两个同名函数，但实现细节不同。“包名”可解决这个问题，某些编程语言叫命名空间。包名下定义的类、函数、全局变量，均带有包名前缀，避免冲突，包名一般与目录对应，但不是必需。“包名”命名规则与标识符类似，但由字符“‘”两端包围。‘反单引号（backquote），又称反引号。位置在键盘中数字键“1”的左边，其上档符号是“~”。

如何引入另一个源码文件，一种方式就是“包含”，包含是源码级的处理，属于同一个编译单元。

以下示例有 2 个源文件，但编译时只需要一个源文件即可。

```
// 2 层包名声明
包名 `通用`.`工具类`

// 全局 声明，在包含的地方可调用
函数 全局 $ 工作 () {
    输出 (" 通用工作")
}
```

源码 60: ch06/inc.lyy

```
包名 `我的`

包含 "inc.lyy"

函数 $ 工作 () {
    输出 (" 我的工作")
}
开始 () {
    `通用`.`工具类`.$ 工作 ()  // 调用被包含文件中的函数，含包名
    `我的`.$ 工作 ()
    $ 工作 ()  // 与上同，同一源文件直接调用
}
/*
编译命令：
lyy inc_main.lyy -o ..\build\ch06\inc_main.exe

执行命令：
..\build\ch06\inc_main.exe

输出结果：
通用工作
我的工作
我的工作
*/
```

源码 61: ch06/inc_main.lyy

6.9　对象数组与指针

自定义抽象数据类型——类，也可定义数组与指针，与内置基本数据类型如“整数”等类似。以下示例，演示了对象数组与指针的用法。

```
类 $ 龙语者 {
    令 $ 级别: 整数
    构造 () {
        $ 级别 = 10
        输出 (" 构造 龙语者")
        //输出 (" 内存地址: {}", 本尊)
    }
    析构 () { 输出 (" 析构 龙语者"); }
    函数 $ 信息 () { 输出 (" 级别: {}", 本尊.$ 级别); }
}
//令 $ 人们: [2] $ 龙语者  // 第二步: 也可声明为顶层变量
开始 () {
    令 $ 人们: [2] $ 龙语者  // 第一步:
    $ 人们 [0].$ 信息 ()
    令 $ 人指针 = 取址 $ 人们 [1]
    //令 $ 人指针 : $ 龙语者 指针  = 取址 $ 人们 [1]  // 与上同
    $ 人指针.$ 信息 ()
}
/*
编译命令:
lyy obj_arr.lyy -o ..\build\ch01\obj_arr.exe

执行命令:
..\build\ch01\obj_arr.exe

输出结果:
构造 龙语者
构造 龙语者
级别: 10
级别: 10
析构 龙语者
析构 龙语者
*/
```

源码 62: ch06/obj_arr.lyy

第七章　模板与泛型编程

现代编程语言大都支持模板与泛型编程，它有非常重要的应用。在软件工程实践中，人们常常发现某些函数或类，实现算法或业务处理逻辑一样，但需要支持不同的数据类型。如同时支持传入整数、实数、自定义类等。于是发明了模板编程。

什么是“模板？它是生成其它源代码的样板。它的编译处理过程如下：

```
模板 -> 实例化（传入模板参数） -> 特定函数或类 -> 目标代码
```

由于它不针对特定数据类型，故又称“通用编程”或“泛型编程”，“泛”的意思，即“一般的，非特化的”。C++ 编程语言标准库大量使用模板，它的名称称为“std”，意为“Standard Template Library”，即“标准模板库”。

由于模板的实现比较复杂，龙语言目前仅支持简单的函数模板，后期有待提升。但需要理解泛型编程思想，很重要。

7.1　函数模板

语法使用尖括号“<>”包括标识符，代表传入的类型，在调用时才实例化产生类型特化的函数。

```
1
2  函数 $ 和<$ 吾型>($i: $ 吾型, $j: $ 吾型)-> $ 吾型
3  {
4      返回 $i + $j
5  }
6
7  开始 () {
8      令 $i = $ 和<整数>(10,20)
9      输出 ("{}", $i)
10     令 $f = $ 和<实数>(10.1,20.2)
11     输出 ("{%.1f}", $f)
12 }
13 /*
14 编译命令:
15 lyy fn_tpl.lyy -o ..\build\ch07\fn_tpl.exe
16
17 执行命令:
18 ..\build\ch07\fn_tpl.exe
19
20 输出结果:
21 30
22 30.3
23 */
```

源码 63: ch07/fn_tpl.lyy

第八章　输入与输出

输入/输出功能是一种编程语言的必备功能。一个程序没有输入，也没有输出，那么它是无意义的。之前我们编写的程序，都在一个黑窗口里面编译、运行，这个黑窗口，我们称之为“控制台 (Console)”，我们的程序也称为控制台程序。控制台的历史可追溯到大型机，那时计算机还是庞大且昂贵的东西，于是人们用显示器通过控制线程连接到大型机上操作，这个黑显示器就称为控制台，当前的计算机都是图形化界面，所以里面的控制台又称为“虚拟控制台”。

控制台里的键盘输入，又称为“标准输入”，输出文本到控制台，又称“标准输出”。C/C++ 定义三种流：stdin 标准输入流，stdout 标准输出流，stderr 标准错误流。标准错误流与标准输出流类类似，但是无缓存。

C/C++ 输入输出标准库可参考：https://en.cppreference.com/w/c/io.html

8.1　格式化输出 printf 函数

我们之前用到的“输出”编译器内置函数，底层就是转为 C 语言的 printf 函数。输出函数为了初学者方便使用，而 printf 用于熟练者更精细控制。C 语言标准定义了许多库函数，在龙语言里面可以直接引用，因为链接器会自动链接 C 语言运行时的导入库。

C 语言 printf 函数声明如下：

```
int printf(const char* format, ... );
```

龙语言 printf 函数对应声明如下：

```
外部 函数 $printf($format: 字节 指针, ...) -> 整数32 #原名
```

format 是格式化字符串，“...”是可变参数，意为参数个数不定，但要与 format 里面的格式对应匹配。

龙语言这里引入一个新的函数注解“# 原名”，意为目标文件中保持”printf” 名，不受包名影响。

详细文档参考：https://en.cppreference.com/w/c/io/fprintf.html

需要熟悉记忆的格式，如：

%lld 整数
%llu 自然数
%d 整数32
%u 自然数32
%s 字符串
%f 实数
%c 字符

示例如下：

```
// C 语言标准库，运行时中的函数 printf
外部 函数 $printf($format: 字节 指针, ...) -> 整数 32 #原名

开始 () {
    令 $ 身高 = 180
    令 $ 体重 = 65.5
    令 $ 姓名 = "Hi 小明"
    // \n 意为转义字符“换行”
    $printf(" 精神小伙：%lld，%.1f，%s\n", $ 身高, $ 体重, $ 姓名)
}
/*
编译命令：
lyy printf.lyy -o ..\build\ch08\printf.exe

执行命令：
..\build\ch08\printf.exe

输出结果：
精神小伙：180，65.5，Hi 小明
*/
```

源码 64: ch08/printf.lyy

8.2　变长参数表

变长参数表，指函数传入的参数个数可变，最常见的函数是 C 语言的 printf 函数。龙语言遵循 C 语言调用约定，同样支持变长参数表，用“...”代表传入的任意参数。

表 8.1: 变长参数表关键字.

龙语言	C/C++	含义
变参列表	va_list	声明变参指针, 形如: 变参列表 ($ap) 变量名 $ap 可任意取其它名
变参开始	va_start	初始变参指针
变参值	va_arg	从变参指针获取指定类型的参数值， 并让指针指向下一个参数位置
变参结束	va_end	清理变参指针

```
函数 $ 打印($cnt: 整数, ...) {
    变参列表 ($ap)
    变参开始 ($ap)
    令 $i = 变参值 ($ap, 整数)  // 取参数，并移动 $ap 指针到下一个参数
    输出 (" 个数:{}, 一:{}, 二:{%.2f}", $cnt, $i, 变参值 ($ap, 实数))  // 直接取参数并打印
    变参结束 ($ap)
}
开始 () {
    $ 打印 (2, 123, 3.14)
}
/*
编译命令:
lyy var_arg.lyy -o ..\build\ch08\var_arg.exe

执行命令:
..\build\ch08\var_arg.exe

输出结果:
个数:2, 一:123, 二:3.14
*/
```

源码 65: ch08/var_arg.lyy

上面示例演示常见用法，而下例是更灵活的例子，总之传入的变参与函数内的处理参数逻辑要一一对应。

```
函数 $ 打印和( ... ) {
    变参列表 ($ap)
    变参开始 ($ap)
    令 $ 个数 = 变参值 ($ap, 整数)  // 取第一个参数，表示后面还有的个数
    令 $ 总和 = 0
    遍历 (令 _ 在 1..=$ 个数) {
        $ 总和 += 变参值 ($ap, 整数)  // 类型都是整数
    }
    变参结束 ($ap)
    输出 (" 个数:{}, 和:{}", $ 个数, $ 总和)
}
开始 () {
    $ 打印和 (4, 11, 22, 33, 44)
}
/*
编译命令:
lyy var_arg2.lyy -o ..\build\ch08\var_arg2.exe

执行命令:
..\build\ch08\var_arg2.exe

输出结果:
个数:4, 和:110
*/
```

源码 66: ch08/var_arg2.lyy

8.3　格式化输入 scanf 函数

龙语言“输入”函数从键盘读入，并自动转化为需要的数值类型。示例如下：

```
// C 语言标准库，运行时中的函数
外部 函数 $printf($format: 字节 指针, ...) -> 整数 32 #原名
开始 () {
    令 $ 身高: 整数 = 0
    令 $ 体重: 实数 = 0.0
    令 $ 姓名: [32] 字符
    输入 ($ 身高, $ 体重, $ 姓名)
    // \n 意为转义字符“换行”
    $printf(" 精神小伙: %lld, %.1f, %s\n", $ 身高, $ 体重, 取址 $ 姓名)
}
/*
编译命令:
lyy input.lyy -o ..\build\ch08\input.exe

执行命令 [IGNORE]:
..\build\ch08\input.exe
[键盘输入] 180 60.6 loong-lang.com

输出结果:
精神小伙: 180, 60.6, loong-lang.com
*/
```

源码 67: ch08/input.lyy

龙语言“输入”内置函数，方便初学者，但功能及灵活性不如 scanf。C 语言 scanf 函数声明如下：

```
int scanf(const char* format, ... );
```

龙语言 scanf 函数对应声明如下：

```
外部 函数 $scanf($format: 字节 指针, ...) -> 整数32 #原名
```

输入格式化与 printf 的格式化类似，参考文档：
https://en.cppreference.com/w/c/io/fscanf.html
示例如下：

```
// C 语言标准库，运行时中的函数
外部 函数 $printf($format: 字节 指针, ...) -> 整数 32 #原名
外部 函数 $scanf($format: 字节 指针, ...) -> 整数 32 #原名

开始 () {
    令 $ 身高: 整数 = 0
    令 $ 体重: 实数 = 0.0
    令 $ 姓名: [32] 字符
    $scanf("%lld%lf%s", 取址 $ 身高, 取址 $ 体重, 取址 $ 姓名)
    // \n 意为转义字符“换行”
    $printf(" 精神小伙: %lld, %.1f, %s\n", $ 身高, $ 体重, 取址 $ 姓名)
}
/*
编译命令:
lyy scanf.lyy -o ..\build\ch08\scanf.exe

执行命令 [IGNORE]:
..\build\ch08\scanf.exe
[键盘输入] 180 60.6 loong-lang.com

输出结果:
精神小伙: 180, 60.6, loong-lang.com
*/
```

源码 68: ch08/scanf.lyy

第九章　标准库与“标准 C”

目前标准库还在建设中，目标是达成 C++ 标准库那样的功能。若您有一个好用的函数或类，欢迎提交补丁。相信在全球华人及后裔的努力下，哪怕千人或万人之中有一人提交一行代码，假以时日，标准库终将趋于完善与易用。

9.1　标准 C

前文提到，龙语言编译时会链接 C 语言运行时，所以，只需要声明“标准 C”函数函数原型，即可调用 C 函数。声明文件位于

`{LYY_HOME}/std/标准C/}`

目录下，LYY_HOME 是龙语言安装主目录，std 是 standard 标准的意思。以下示例演示了调用 C 语言的 qsort 快速排序，对无序数组进行排序处理。

```
包含 " 标准 C/stdio.lyy"  // 标准 C 函数声明
包含 " 标准 C/stdlib.lyy"

函数 $ 排序方法 ($a:整数 指针, $b:整数 指针) -> 整数 32
{
    返回 (整数 32)(($a 指向) - ($b 指向))  // a-b 升序, b-a 降序
}
开始 () {
    令 $a : [] 整数 = [ 10, -2, 33, 8]
    令 $ 元素个数 = 字节数 ($a) / 字节数 (整数)
    令 $ 元素大小 = 字节数 (整数)
    //输出 ("{},{}", 字节数 ($a), 字节数 (整数))
    遍历 (令 $i 在 0..$ 元素个数) {
        `标准 C`.$printf("%lld, ", $a[$i])  // 输出
    }
    `标准 C`.$printf("\n")  // 换行

    `标准 C`.$qsort($a, $ 元素个数, $ 元素大小, $ 排序方法) //C 快速排序, 关键的一行!

    遍历 (令 $i 在 0..$ 元素个数) {
        `标准 C`.$printf("%lld, ", $a[$i])  // 输出
    }
    `标准 C`.$printf("\n")
}
/*
编译命令:
lyy qsort.lyy -o ..\build\ch09\qsort.exe

执行命令:
..\build\ch09\qsort.exe

输出结果:
10, -2, 33, 8,
-2, 8, 10, 33,
*/
```

源码 69: ch09/qsort.lyy

9.2　标准库

龙语言标准库位于目录：

`{LYY_HOME}/std/标准库/}`

标准库底层可使用其它已经成熟 C 语言组件，使用 C 编译器编译成动/静态库，通过包含龙语言函数声明，链接器链接，直接使用。

再次强调，龙语言使用 **C 函数调用约定**。

比如，最常用的字符串，底层使用 sds：

https://github.com/antirez/sds

sds - Simple Dynamic Strings library for C

UTF-8 字符处理使用 utf8proc:

https://github.com/JuliaStrings/utf8proc

utf8proc - a clean C library for processing UTF-8 Unicode data

其它组件也可使用此类方法，避免从头开始实现，不必重新造轮子，有好用的拿来即可。

字符串简单示例如下，更多功能及用法请查看各依赖库源码：

```
包含 " 标准库/字符串.lyy"

开始 () {
    令 $s = `标准库`.$ 字符串 ("Hi, 龙语者, loong-lang.com")
    令 $s1: `标准库`.$ 字符串
    $s1.$ 追加 ("Hi, 龙语者 1, loong-lang.com")
    令 $c: $sds = $s1.$ 复制内容 ()
    $c = `依赖库`.$sdscat($c, ",666")  // 串连接
    输出 ("{%s}", $s)
    输出 ("{%s}", $s1)
    输出 ("{%s}", $c)
    `依赖库`.$sdsfree($c)
}
/*
编译命令:
lyy str.lyy -o ..\build\ch09\str.exe

执行命令:
..\build\ch09\str.exe

输出结果:
Hi, 龙语者, loong-lang.com
Hi, 龙语者 1, loong-lang.com
Hi, 龙语者 1, loong-lang.com,666
*/
```

源码 70: ch09/str.lyy

9.3　动态库

将组件编译成动态库（又名“共享库”），是编程语言工具包的常见需求。构建成动态库后，可以被其它所有支持的编程语言调用。

9.3.1　Windows MSVC 调用

在 Windows MSVC 编译器命令行中，需要龙语言源文件及导出文件，如下：

```
EXPORTS
    iadd
    fadd
```

源码 71: ch09/add_export.def

```
函数 $iadd($a:整数, $b:整数) -> 整数
{
    返回 $a + $b
}
函数 $ 实数和 ($a:实数, $b:实数) -> 实数
    # 导出名 ("fadd")  // 函数注解，设置目标文件中符号名
{
    返回 $a + $b
}

/*
编译命令 [MSVC]:
lyy add.lyy -o ..\build\ch09\libadd.dll -shared -Wl "/DEF:add_export.def"
编译命令 [LINUX]:
lyy -shared add.lyy -o ../build/ch09/libadd.so

查看导出符号表 [MSVC]:
dumpbin /EXPORTS ..\build\ch09\libadd.dll
查看导出符号表 [LINUX]:
nm -D ../build/ch09/libadd.so
*/
```

源码 72: ch09/add.lyy

C 语言调用上面动态库示例：

```c
#include <stdio.h>
extern long long iadd(long long, long long);
extern double fadd(double,double);
int main()
{
    printf("%lld\n", iadd(10,20));
    printf("%.2f\n", fadd(10.11,20.22));
}
/*
编译命令 [MSVC]:
cl add_use.c ..\build\ch09\libadd.lib /link /out:..\build\ch09\add_use.exe
执行命令 [MSVC]:
..\build\ch09\add_use.exe

编译命令 [LINUX]:
clang add_use.c ../build/ch09/libadd.so -o ../build/ch09/add_use.out
执行命令 [LINUX]:
../build/ch09/add_use.out

查看依赖 [MSVC]:
dumpbin /DEPENDENTS ..\build\ch09\add_user.exe
查看依赖 [LINUX]:
ldd ../build/ch09/add_use.out
*/
```

源码 73: ch09/add_use.c

9.3.2　Python3 调用

Python3 调用共享库有 2 种方法：ctypes 和 CFFI。

ctypes

ctypes 是 **Python 标准库**中的外部函数接口模块，用于调用 C 语言编写的动态链接库（DLL/共享库）。

```python
import ctypes

lyy_dll = ctypes.CDLL('../build/ch09/libadd.dll')
# 定义函数返回类型
lyy_dll.iadd.restype = ctypes.c_longlong
# 定义函数参数类型
lyy_dll.iadd.argtypes = [ctypes.c_longlong, ctypes.c_longlong]

# 设置完返回类型和参数类型后，就可以调用 DLL 中的函数了。
result = lyy_dll.iadd(100, 200)
print("python3 调用 lyy 动态库结果:", result)
# 运行命令: python add_ctypes.py
# 运行结果: python3 调用 lyy 动态库结果: 300
```

源码 74: ch09/add_ctypes.py

CFFI

CFFI（C Foreign Function Interface）是 Python 的一个外部函数接口库，用于直接调用 C 代码。python pip 安装命令 (使用清华镜像):

```
pip install -i https://mirrors.tuna.tsinghua.edu.cn/pypi/web/simple cffi
```

```python
import os
from cffi import FFI

ffi = FFI()

# 定义 C 函数接口
ffi.cdef("""
    double fadd(double, double);
""")

# 加载 DLL 文件，相对路径转绝对路径
absolute_path = os.path.abspath(r'../build/ch09/libadd.dll')
lyy_dll = ffi.dlopen(absolute_path)
# 定义接口后，就可以像使用普通 Python 函数一样调用 DLL 中的函数。
result = lyy_dll.fadd(100.1, 200.2)
print("python cffi 调用 lyy 共享库结果:{:.1f}".format(result))
# 运行命令: python add_cffi.py
# 运行结果: python cffi 调用 lyy 共享库结果:300.3
```

源码 75: ch09/add_cffi.py

9.4　静态库

静态库是将多个目标文件链接成为一个目标文件，运行时无需其它依赖，分发更方便，但会导致目标文件较大。

9.4.1　Windows MSVC

```
函数 $iadd($a:整数, $b:整数) -> 整数
{
    返回 $a + $b
}
函数 $ 实数和 ($a:实数, $b:实数) -> 实数
    # 导出名 ("fadd")  // 函数注解，设置目标文件中符号名
{
    返回 $a + $b
}

/*
编译命令 [MSVC]:
lyy -static add_static.lyy -o ..\build\ch09\libadd_static.lib
编译命令 [LINUX]:
lyy -static add_static.lyy -o ../build/ch09/libadd_static.lib
*/
```

源码 76: ch09/add_static.lyy

C 语言调用上面静态库示例：

```c
#include <stdio.h>
extern long long iadd(long long, long long);
extern double fadd(double,double);
int main()
{
    printf("%lld\n", iadd(100,200));
    printf("%.2f\n", fadd(100.11,200.22));
}
/*
编译命令 [MSVC]:
cl add_use_static.c ..\build\ch09\libadd_static.lib /link /out:..\build\ch09\add_user_static.exe
执行命令 [MSVC]:
..\build\ch09\add_user_static.exe

编译命令 [LINUX]:
gcc add_use_static.c ../build/ch09/libadd_static.lib -o ../build/ch09/add_user_static.out
执行命令 [LINUX]:
../build/ch09/add_user_static.out

输出结果:
300
300.33

查看依赖 [MSVC]:
dumpbin /DEPENDENTS ..\build\ch09\add_user_static.exe
查看依赖 [LINUX]:
ldd ../build/ch09/add_user_static.out
*/
```

源码 77: ch09/add_use_static.c

第十章　终章：既济，未济

对于有其它编程语言经验的读者，此书是很容易的。若是此前无任何编程经验，读到此处能理解 8 成以上，不用怀疑，你就是传说中的天选之子:-)。暂时不理解也没关系，只是暂时机缘未到，假以时日，点滴知识串连起来融汇贯通，也不是难事。但不要以为理解本书就熟悉编程了，建议以本书学会的编程思想为基础，再去学一门或几门编程语言。若是计算机专业的，建议再学 C/C++。C++ 编程语言较复杂，但本人记得一句话，已忘记出处，大意是，“不需要懂每个 C++ 的细节，再去写有用的 C++ 程序”。言外之意，不需要被复杂的语言细节吓到，真正有用的软件，大多用到的只不过一些很简单朴素的语言特性。

《周易》64 卦，最后两卦是“既济，未济”。济，渡河也。既济是已渡河，为何既济之后又是未济（未渡河）? 古人教导我们，渡了这条河，前面还有一条更大的河。河外有河，山外有山，天外有天，人外有人。“路漫漫其修远兮，吾将上下而求索”。

下面以解决一个古典的问题，来演示如何用龙语言编程解决现实中的问题。老祖宗应该怎么也想不到，千年之后有人用一个叫电脑和中文编程的东西，解决当时的计算问题。选择这个示例的原因：一是体现古人朴素的世界观，弘扬古典文化。二是个人觉得包含人生哲理，有值得思索把玩之处。此示例规则有一定难度，看不懂也没关系，可以跳过。

10.1　天干阴阳五行示例

表 10.1: 十天干及阴阳五行.

天干	甲	乙	丙	丁	戊	己	庚	辛	壬	癸
阴阳	阳	阴	阳	阴	阳	阴	阳	阴	阳	阴
五行	木		火		土		金		水	

十天干可初浅认为是古人用的序数，分别有阴阳、五行属性如上表格。

五行相生的顺序为：木生火，火生土，土生金，金生水，水生木；五行相克的顺序为：木克土、土克水、水克火、火克金、金克木。

从十天干任取 2 位分别当作“主体”、“客体”。当主体遇到客体时，由主、客体各自己阴阳五行属性，判断两者**“关系”**，遵循以下原则：

我克者为“正财”、“偏财”
克我者为“正官”、“偏官”
我生者为“伤官”、“食神”
生我者为“正印”、“偏印”
同我者为“劫财”、“比肩”
阳遇阴、阴遇阳为正（即前者）；阳遇阳、阴遇阴为偏（后者）。

举例：主体“甲（阳、木）”遇到客体“丙（阳、火）”，由于“木生火，阳遇阳”，则关系为“食神”；类似，甲遇丁，则关系为“伤官”。

问题：任意给定 2 位天干，编程判断他们的关系。

源码 **ch10/wuxing.lyy**：

```
枚举 $ 天干 { $ 甲 $ 乙 $ 丙 $ 丁 $ 戊 $ 己 $ 庚 $ 辛 $ 壬 $ 癸 }
枚举 $ 阴阳 { $ 阴 $ 阳}  // 枚举与阴阳类型区分
枚举 $ 五行 { $ 金 $ 木 $ 水 $ 火 $ 土 }
枚举 $ 生克 { $ 我克 $ 克我 $ 我生 $ 生我 $ 同我 }

函数 $ 关系 ($ 主体: $ 天干, $ 客体: $ 天干) {
    令 $ 天干属性: [10][2] 整数 32 = [
        [$ 阴阳.$ 阳, $ 五行.$ 木],  // 甲: 下标 0 即是天干值
        [$ 阴阳.$ 阴, $ 五行.$ 木],  // 乙
        [$ 阴阳.$ 阳, $ 五行.$ 火],  // 丙
        [$ 阴阳.$ 阴, $ 五行.$ 火],  // 丁
        [$ 阴阳.$ 阳, $ 五行.$ 土],  // 戊
        [$ 阴阳.$ 阴, $ 五行.$ 土],  // 己
        [$ 阴阳.$ 阳, $ 五行.$ 金],  // 庚
        [$ 阴阳.$ 阴, $ 五行.$ 金],  // 辛
        [$ 阴阳.$ 阳, $ 五行.$ 水],  // 壬
        [$ 阴阳.$ 阴, $ 五行.$ 水],  // 癸
    ]
    令 $ 五行关系: [25][3] 整数 32 = [
        [$ 五行.$ 木, $ 五行.$ 木, $ 生克.$ 同我],  // 主体: 木
        [$ 五行.$ 木, $ 五行.$ 火, $ 生克.$ 我生],
        [$ 五行.$ 木, $ 五行.$ 土, $ 生克.$ 我克],
        [$ 五行.$ 木, $ 五行.$ 金, $ 生克.$ 克我],
        [$ 五行.$ 木, $ 五行.$ 水, $ 生克.$ 生我],
        [$ 五行.$ 火, $ 五行.$ 木, $ 生克.$ 生我],  // 主体: 火
```

```
        [$ 五行.$ 火, $ 五行.$ 火, $ 生克.$ 同我],
        [$ 五行.$ 火, $ 五行.$ 土, $ 生克.$ 我生],
        [$ 五行.$ 火, $ 五行.$ 金, $ 生克.$ 我克],
        [$ 五行.$ 火, $ 五行.$ 水, $ 生克.$ 克我],
        [$ 五行.$ 土, $ 五行.$ 木, $ 生克.$ 克我],  // 主体：土
        [$ 五行.$ 土, $ 五行.$ 火, $ 生克.$ 生我],
        [$ 五行.$ 土, $ 五行.$ 土, $ 生克.$ 同我],
        [$ 五行.$ 土, $ 五行.$ 金, $ 生克.$ 我生],
        [$ 五行.$ 土, $ 五行.$ 水, $ 生克.$ 我克],
        [$ 五行.$ 金, $ 五行.$ 木, $ 生克.$ 我克],  // 主体：金
        [$ 五行.$ 金, $ 五行.$ 火, $ 生克.$ 克我],
        [$ 五行.$ 金, $ 五行.$ 土, $ 生克.$ 生我],
        [$ 五行.$ 金, $ 五行.$ 金, $ 生克.$ 同我],
        [$ 五行.$ 金, $ 五行.$ 水, $ 生克.$ 我生],
        [$ 五行.$ 水, $ 五行.$ 木, $ 生克.$ 我生],  // 主体：水
        [$ 五行.$ 水, $ 五行.$ 火, $ 生克.$ 我克],
        [$ 五行.$ 水, $ 五行.$ 土, $ 生克.$ 克我],
        [$ 五行.$ 水, $ 五行.$ 金, $ 生克.$ 生我],
        [$ 五行.$ 水, $ 五行.$ 水, $ 生克.$ 同我],
    ]
    //输出 (" 主体:{}, 客体:{}", $ 主体, $ 客体)
    令 $ 阴阳相异:整数 32 = $ 天干属性 [$ 主体][0] 位异或 $ 天干属性 [$ 客体][0]
    //输出 (" 阴阳相异:{}", $ 阴阳相异)
    令 $ 主体五行:整数 32 = $ 天干属性 [$ 主体][1]
    令 $ 客体五行:整数 32 = $ 天干属性 [$ 客体][1]
    //输出 (" 主体五行:{}, 客体五行:{}", $ 主体五行, $ 客体五行)
    令 $ 生克值: 整数 32 = -1
    遍历 (令 $i 在 0..25){
        若 ($ 五行关系 [$i][0] == $ 主体五行 且 $ 五行关系 [$i][1] == $ 客体五行){
            $ 生克值 = $ 五行关系 [$i][2]
            中断
        }
    }
    若 ($ 生克值 == -1){
        输出 (" 主体或客体值错误")
        返回
    }
    //输出 (" 生克值:{}", $ 生克值)
    若 ($ 生克值 == $ 生克.$ 我克){
        若 ($ 阴阳相异 == 1){
            输出 (" 正财")
        }
```

```
        否则 若 ($ 阴阳相异 == 0){
            输出 (" 偏财")
        }
    }
    否则 若 ($ 生克值 == $ 生克.$ 克我){
        若 ($ 阴阳相异 == 1){
            输出 (" 正官")
        }
        否则 若 ($ 阴阳相异 == 0){
            输出 (" 偏官")
        }
    }
    否则 若 ($ 生克值 == $ 生克.$ 我生){
        若 ($ 阴阳相异 == 1){
            输出 (" 伤官")
        }
        否则 若 ($ 阴阳相异 == 0){
            输出 (" 食神")
        }
    }
    否则 若 ($ 生克值 == $ 生克.$ 生我){
        若 ($ 阴阳相异 == 1){
            输出 (" 正印")
        }
        否则 若 ($ 阴阳相异 == 0){
            输出 (" 偏印")
        }
    }
    否则 若 ($ 生克值 == $ 生克.$ 同我){
        若 ($ 阴阳相异 == 1){
            输出 (" 劫财")
        }
        否则 若 ($ 阴阳相异 == 0){
            输出 (" 比肩")
        }
    }
}
开始 () {
    $ 关系 ($ 天干.$ 甲, $ 天干.$ 甲)
    $ 关系 ($ 天干.$ 甲, $ 天干.$ 乙)
    $ 关系 ($ 天干.$ 甲, $ 天干.$ 丙)
    $ 关系 ($ 天干.$ 甲, $ 天干.$ 丁)
```

```
    $ 关系 ($ 天干.$ 甲, $ 天干.$ 戊)
    $ 关系 ($ 天干.$ 甲, $ 天干.$ 己)
    $ 关系 ($ 天干.$ 甲, $ 天干.$ 庚)
    $ 关系 ($ 天干.$ 甲, $ 天干.$ 辛)
    $ 关系 ($ 天干.$ 甲, $ 天干.$ 壬)
    $ 关系 ($ 天干.$ 甲, $ 天干.$ 癸)
    /*
    // 取消注释测试以下
    $ 关系 ($ 天干.$ 癸, $ 天干.$ 甲)
    $ 关系 ($ 天干.$ 癸, $ 天干.$ 乙)
    $ 关系 ($ 天干.$ 癸, $ 天干.$ 丙)
    $ 关系 ($ 天干.$ 癸, $ 天干.$ 丁)
    $ 关系 ($ 天干.$ 癸, $ 天干.$ 戊)
    $ 关系 ($ 天干.$ 癸, $ 天干.$ 己)
    $ 关系 ($ 天干.$ 癸, $ 天干.$ 庚)
    $ 关系 ($ 天干.$ 癸, $ 天干.$ 辛)
    $ 关系 ($ 天干.$ 癸, $ 天干.$ 壬)
    $ 关系 ($ 天干.$ 癸, $ 天干.$ 癸)
    */
}
/*
编译命令:
lyy wuxing.lyy -o ..\build\ch10\wuxing.exe -O3

执行命令:
..\build\ch10\wuxing.exe

输出结果:
比肩
劫财
食神
伤官
偏财
正财
偏官
正官
偏印
正印
*/
```

若是有哈希表之类的数据结构，该问题还有不同的解法。也就是说，同一个问题，不同的编程语言，不同的人员，会有不同的解法。编程之道，并非仅有固定一条路。

10.2　待完成

由于个人时间精力和技术广度的限制，以下方面未完成，计划将来要做的事。若您有任一方面的特长，请通过邮件或网站与本人联系，为此项目做一点贡献。哪怕每人仅贡献一行代码，众人捡柴火焰高，您做的些许贡献或将影响千万华人及后裔。

1. 标准库
2. 语言特性增加 RTTI 及异常，模板功能加强。
3. 词法、语法解析器的加强
4. 调试器，参考或兼容 LLDB
5. 集成开发环境 IDE，初步计划基于 Code::Blocks

附录 A　安装指南

龙语言编译器基于 LLVM，只要操作系统支持编译 LLVM 及 clang++，即可支持龙语言。

龙语言编译器下载主页为：https://loong-lang.com/download/
选择最适合自己的某一平台版本下载即可。

A.1　Windows

Windows 操作系统下，有 2 个版本选择，一是MSVC（MicroSoft Visual C++），二是MSYS2。初学者如何选择？若是仅在 Windows 平台，且进行产品级别的开发，不考虑迁移 Linux，建议选择 MSVC 版本。若是今后考虑跨平台，转到 Linux，建议使用 MSYS2 版本。

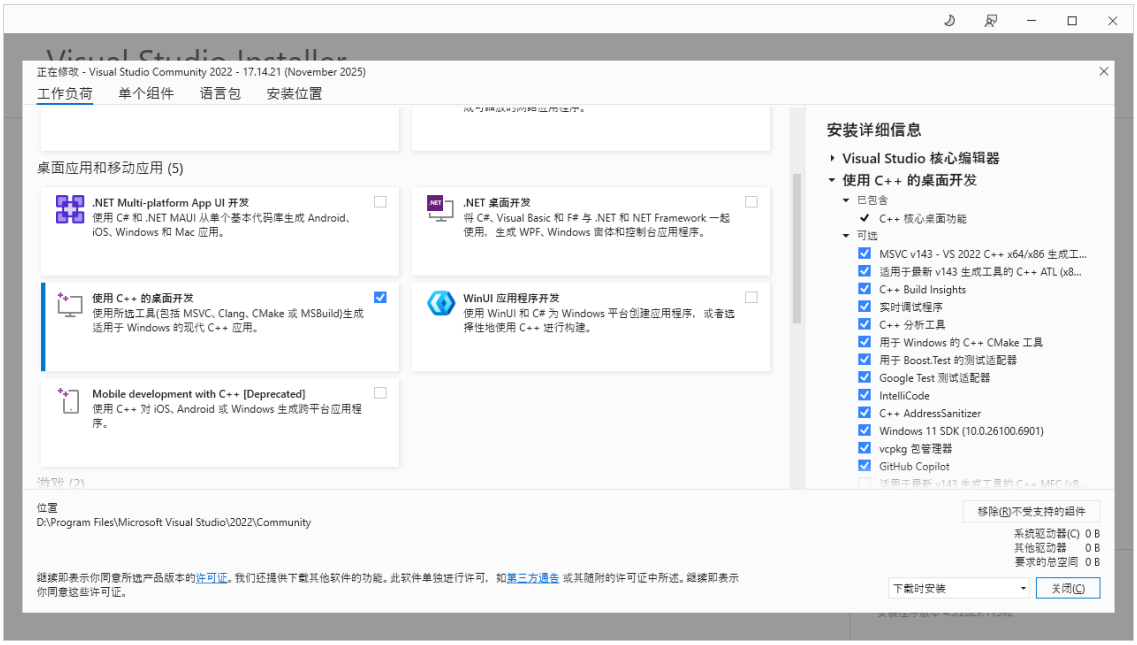
A.1.1　MSVC

MSVC 2022 与 2026 任选一种即可。

安装 MSVC 2022

1. 下载 MSVC 2022。这里选择“**Microsoft Visual Studio Community 2022 (64 位)**”，直接下载安装器: https://aka.ms/vs/17/release/vs_community.exe

2. 双击安装。选中“使用 C++ 的桌面开发”，然后根据向导下一步即可。



3. (可选) 从菜单栏找到“x64 Native Tools Command Prompt for VS 2022”，保存快捷方式到“开始”屏幕或桌面，方便以后访问。龙语言的编译过程，也在此命令窗口中。

4. (可选) 在命令行窗口运行“cl”（MSVC 编译器）及”link”（MSVC 链接器）。输出如下，表明 MSVC 安装成功：

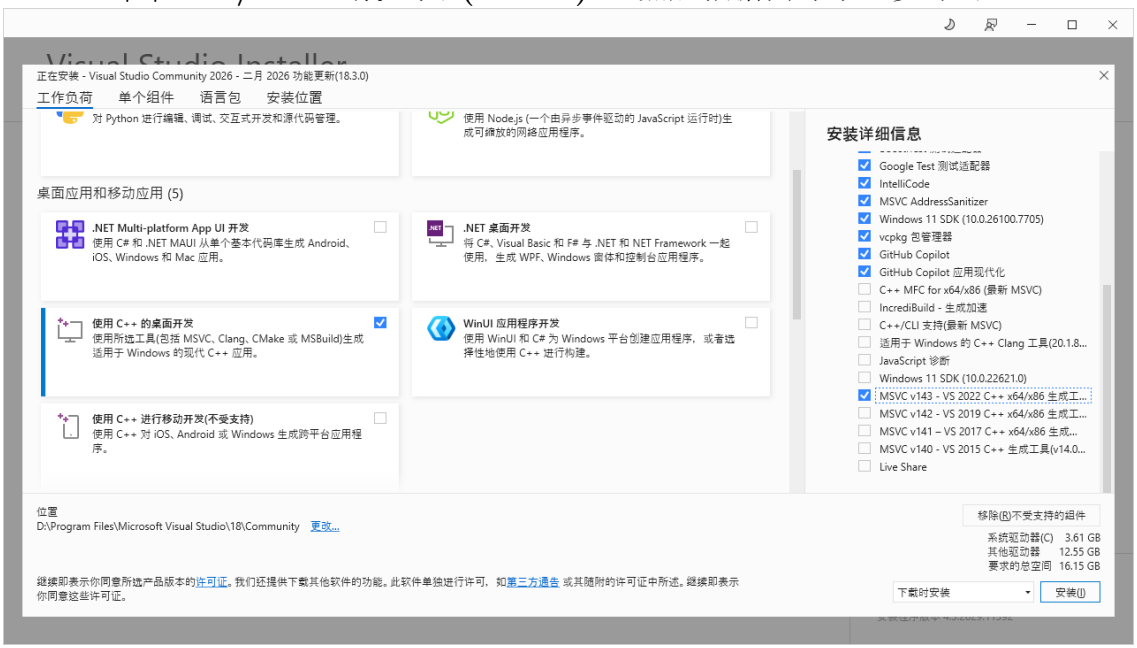
```
> cl
用于 x64 的 Microsoft (R) C/C++ 优化编译器 19.44.35222 版
版权所有(C) Microsoft Corporation。保留所有权利。

用法: cl [ 选项... ] 文件名... [ /link 链接选项... ]

> link
Microsoft (R) Incremental Linker Version 14.44.35222.0
Copyright (C) Microsoft Corporation.  All rights reserved.

用法: LINK [选项] [文件] [@commandfile]
...
```

安装 MSVC 2026

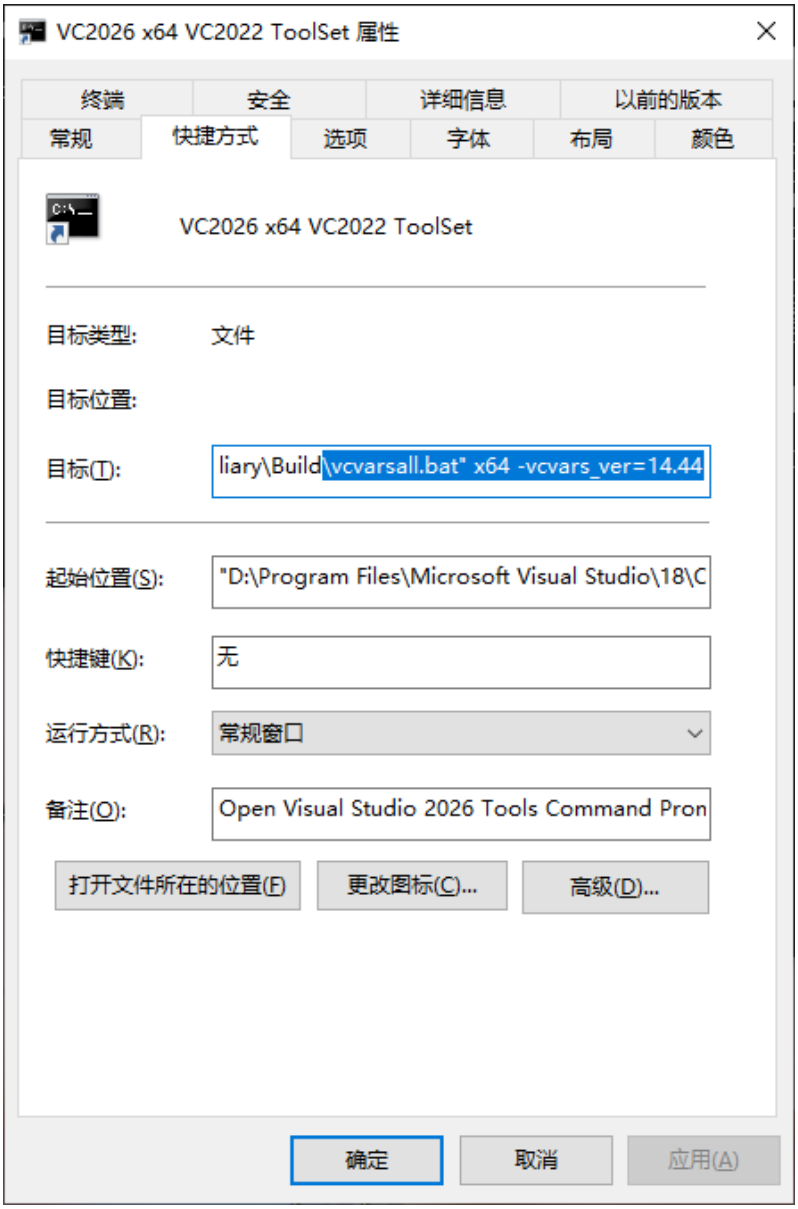
Visual Studio 2022 Community (社区版) 已不提供明确下载链接，目前最新版是Visual Studio 2026。但同样可以通过它安装 MSVC 2022 的编译器套件。

1. 下载是MSVC 2026 Community 版本。

2. 双击安装。选中“使用 C++ 的桌面开发”，再在右侧选中“**MSVC v143- VS 2022 C++ x64/x86 生成工具 (v14.44)**”，然后根据向导下一步即可。



3. 安装完成后，找到“x64 Native Tools Command Prompt for VS”，复制快捷方式，右击-> 属性-> 修改“目标”为：`\vcvarsall.bat" x64 -vcvars_ver=14.44`，这样即可使用 MSVC 2022 编译套件，参考下图:



4. (可选) 双击运行上一步复制修改的快捷方式，在命令行窗口运行“cl”（MSVC 编译器）及”link”（MSVC 链接器）。输出如下，表明 MSVC 安装成功，及确认版本：

```
>cl
用于 x64 的 Microsoft (R) C/C++ 优化编译器 19.44.35222 版
版权所有(C) Microsoft Corporation。保留所有权利。

用法: cl [ 选项... ] 文件名... [ /link 链接选项... ]
```

使用龙语言 MSVC 版本

下载龙语言编译器 lyy-x.x.x-msvc.zip，解压到合适目录，如：“D:\lyy\”，再打开上面的命令窗口“x64 Native Tools Command Prompt ...”，把龙语言初始“lyyenv.bat”脚本，拖动到该命令窗口，按回车键执行，这样就初始好了龙语言编译环境。输入命令 lyy，即可查看龙语言编译器版本信息。

```
> lyy
龙语言中文程序编译器版本: x.x.x
https://loong-lang.com
邮箱:loong-lang@qq.com
命令格式: lyy [选项列表] 源文件 [-o 目标文件名]
```

A.1.2　MSYS2

MSYS2是一套工具和库，在 Windows 上提供一套类似 Linux shell 工作环境。MSYS2 有不同的环境，这里选择 CLANG64。安装包使用清华镜像 MSYS2下载，比较快。

1. 下载 MSYS2 安装包。
2. 安装完成后，从开始菜单找到“MSYS2 CLANG64”，打开命令行窗口，运行以下命令，替换软件包镜像网址。(以下是一行命令)

   ```
   sed -i "s#https\?://mirror.msys2.org/#https://mirrors.
   tuna.tsinghua.edu.cn/msys2/#g" /etc/pacman.d/mirrorlist*
   ```

3. 同步软件包数据库，与更新客户端，运行：pacman -Syu。若是从旧版本更新，会覆盖镜像修改，需要再运行一遍上面的更改。
4. 打开“MSYS2 CLANG64”命令窗口，运行以下命令安装 clang。龙语言依赖 clang 的链接器和链接库，及其它工具。

   ```
   pacman -S mingw-w64-clang-x86_64-clang
   ```

   若磁盘空间足够，可以安装以下工具组（包含以上工具包）。

   ```
   pacman -S mingw-w64-clang-x86_64-toolchain
   ```

5. (可选)，运行以下命令，查看信息。

```
$ clang -v
clang version ...

$ clang -print-search-dirs
programs: ...
```

使用龙语言 MSYS2 版本

下载龙语言编译器 lyy-x.x.x-msvc.zip，解压到合适目录，如：“D:\lyy\”。更建议使用 linux 风格路径，解压到“/opt/lyy/”下面。再打开上面的命令窗口“MSYS2 CLANG64”，输入”source ”，注意后面加个空格，再把龙语言初始“lyyenv.sh”脚本，拖动到该命令窗口。注意 MSYS2 里面路径格式，/d/代表 D 盘，路径用/表示，形如：

```
$ source /d/lyy/lyyenv.sh
```

或者：

```
$ source /opt/lyy/lyyenv.sh
```

按回车键执行，这样就初始好了龙语言编译环境。输入命令 lyy，即可查看龙语言编译器版本信息。

```
 > lyy
 龙语言中文程序编译器版本: x.x.x
 https://loong-lang.com
 邮箱:loong-lang@qq.com
 命令格式: lyy [选项列表] 源文件 [-o 目标文件名]
```

为了避免每次运行以上脚本，可以把上面初始脚本加入到启动：编辑:

```
{MSYS2安装主目录}/home/{username}/.bashrc
```

在文件最后加入：

```
source /{您的路径}/lyy/lyyenv.sh
```

这样每次启动 MSYS2 命令窗口，即可自动初始好龙语言环境。

A.2 Ubuntu Linux

1. 打开“终端”(Shell)，输入以下命令，更新软件包信息。

```
sudo apt update
```

2. 安装基本开发工具包。

```
sudo apt install build-essential
```

3. (可选)，运行以下命令，查看信息。

```
$ gcc --version
gcc (Ubuntu ...

$ gcc -print-search-dirs
install: /usr/lib/gcc ...
```

使用龙语言 Ubuntu Linux 版本

下载龙语言编译器 lyy-x.x.x-ubuntu.tar.gz，解压到合适目录，如：“/opt/lyy”。命令如入：

```
sudo mkdir -p /opt/lyy
sudo tar -zxvf lyy-x.x.x-ubuntu.tar.gz -C /opt/lyy/
source /opt/lyy/lyyenv.sh
```

与 MSYS2 版本类似，也可把初始脚本加入到~/.bashrc自动启动。

附录 B　命令行参数

命令行参数简要说明

- 查看版本

```
$ lyy
龙语言中文程序编译器版本: x.x.x
http://loong-lang.com
loong-lang@qq.com
命令格式: lyy [选项列表] 源文件 [-o 目标文件名]
```

- 查看使用的 LLVM 版本

```
$ lyy --version
LLVM (http://llvm.org/):
LLVM version 20...
```

- 编译极简形式，默认生成 a.exe(Windows) 或 a.out(Linux)

```
$ lyy 测试.lyy
编译成功: a.exe
用时: 0.xxx 秒
```

- 编译指定输出文件名，*.exe(Windows) 或 *.out(Linux)

```
$ lyy test.lyy -o test.out
编译成功: test.out
用时: 0.xxx 秒
```

注意，发布程序时，务必使用 -O3 或 -O2 优化代码，目标代码优化与否差别很大，重要！

结语

不知不觉，本人工作于此项目已将近一年。一年前，我产生做“中文编译器”的念头，于是买来编译原理方面的书籍，外加网上所能找到所有的编译方面的信息，选择了一条道，闭门造车，几乎没有了所有社交活动，全职聚焦工作于此，才终于有了您眼前的龙语言编译器及本文档。其中艰难之处，不足道也。有时遇到难题，苦思冥想，一度想要放弃，好在第二日或后几日终有解决方案。虽然目前已基本可用，而且有比较独特的特性与品味，但它离我心中的目标与成熟度，还差比较远。不过我认为，假以时日，星星之火，终成燎原之势。

《金刚经》云，“以七宝满尔所恒河沙数三千大千世界，以用布施”，其福德，不若“以此经为他人说”。若本人，**以“中文编程法”，布施全球当前 15 亿华人及后裔**，其福德，又当如何？读者亦当秉持此心。一枝独秀不是春，众人拾柴火焰高。若大部分人如此，何愁国家民族不兴?!

前路漫漫，道阻且长，他日顶峰相见，道一声，“道友，龙语者长存”!

捐赠

最后不能免俗，请有能力的道友捐赠一个盒饭吧。“我为人人，人人为我”。“爱出者爱返，福往者福来”。或者希望有能力与条件的道友，贡献源码，传播与传承，无量福德！

(a) (b)

图 B.1: 捐赠 9.9

(a) (b)

图 B.2: 捐赠任意额“张”某