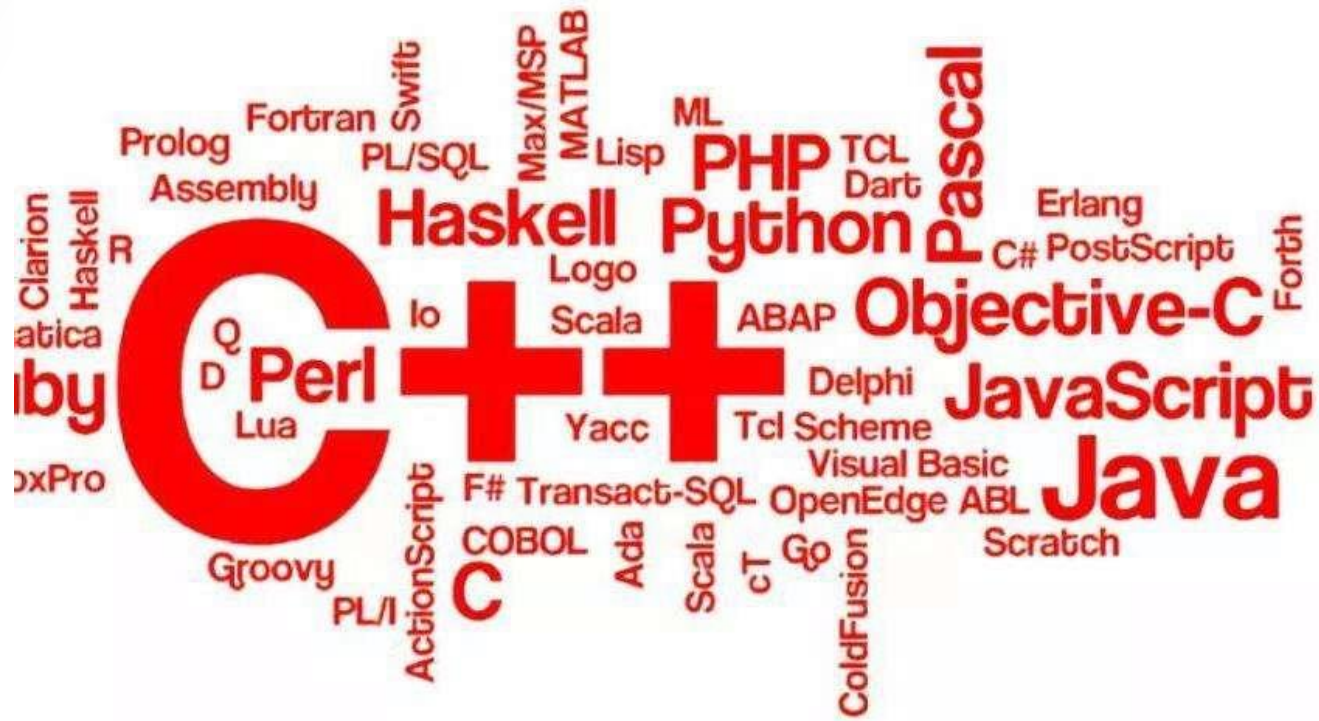


- **5.1 程序设计语言**
- **5.2 编程习惯**
- **5.3 程序的效率**
- **5.4 编码管理**

第五章 实现

介绍软件实现的语言、编程的习惯和注意事项、如何编写高效率的程序，如何管理不同版本的代码以及代码的更改。



5.1. 程序设计语言

- 引言
- 概念
- 特点
- 分类
- 选择要素

5.1.1.引言

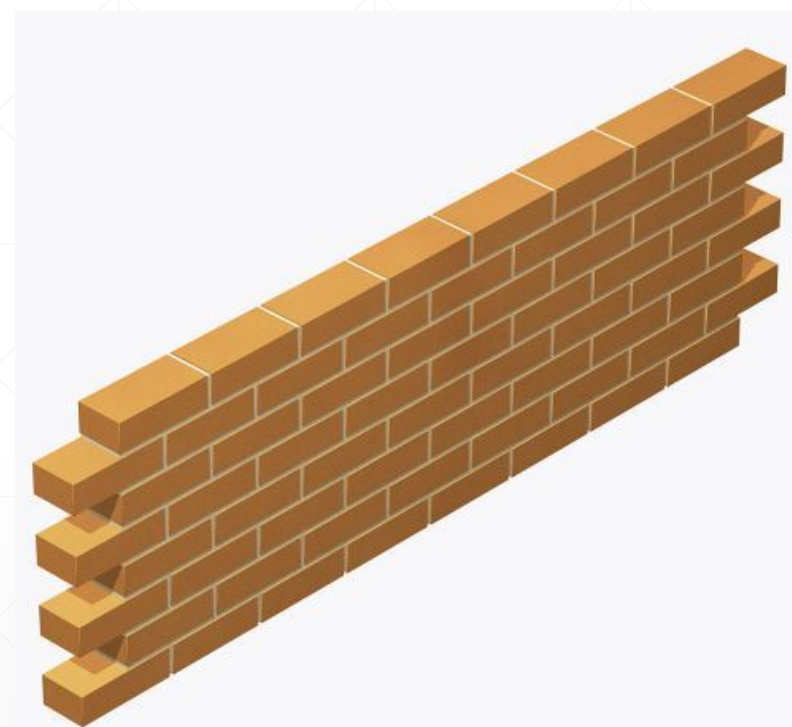
建筑工程中的砌砖：是工程的基础,是设计实现的具体实施。



手艺太差了，还不规范



屋里的墙→砖的应用不合适



手艺不错，规范

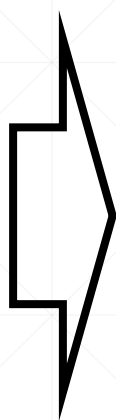
5.1.1.引言

砖的特点——程序语言特点

砌墙的手艺——编程基本功

砌砖风格——编程风格

砌墙的规范——编程规范



选择合适的程序设计语言——**有效**

良好的编程基本功——**正确**

良好的编程习惯——**易理解、简单、自然**

遵守规范和规则——**可拓展、可伸缩**

× 程序只要通过调试和测试就好，满足需求定义就好

✓ 程序应该正确、有效、易理解、简单、自然、可拓展、可伸缩

5.1.2.概念—程序设计语言

程序设计语言是：人与计算机通信的最基本工具。

语法 + 语义 + 语用



程序语言特性的影响

开发人员的思路和解决问题的方式

代码的可理解性、可维护性



编码之前的一项重要工作就是选择一种适当的编程语言

5.1.3.特点——不同的语言适用于不同的应用

COBOL语言

数据处理程序等

PHP语言

网页处理程序等

C语言

系统软件开发等

JAVA语言

跨平台的应用软件开发等

5.1.3.特点——技术方面

需要复杂的
数据结构

仔细衡量有哪些语言能提供这些复杂的数据结构描述

要求高性能及
实时处理能力

选用适合于实时处理的语言，如C或汇编语言

要求许多输出报告
或繁杂的文件处理

选用PowerBuilder、Delphi或SQL比较合适

5.1.3.特点——技术方面

程序设计语言

基本成分分类

数据成分

指明该语言能接受的数据，如各种类型的变量、数组、指针、记录等

运算成分

指明该语言可执行的运算，如 $+$ ， $-$ ， $*$ 、 $/$

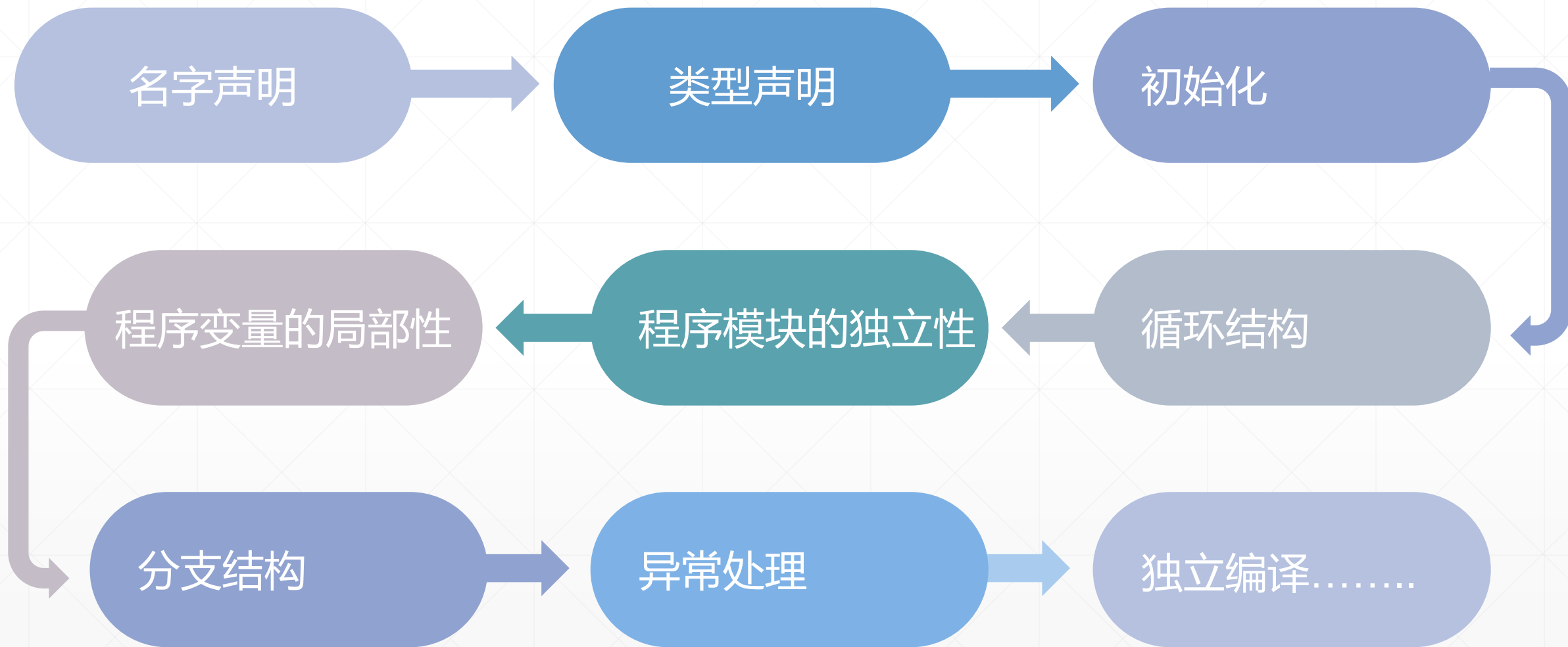
控制成分

顺序结构、条件选择结构和循环结构

传输成分

数据的传输方法，例如输入、输出函数

5.1.3.特点——语言本身



5.1.3.特点——软件工程方面

详细设计

- 能够直接地翻译成程序代码

源程序的可移植性

- 应该严格遵守相应的标准编写程序代码

编译程序效率较高

- 首先应该支持独立编译，纠错功能强，辅助程序员提高程序调试效率

应用代码生成工具

- 许多语言都有与它相应的编译程序、连接程序、调试程序、代码格式化程序、交叉编译程序、宏处理程序和标准子程序库等

可维护性

- 源程序的可读性、语言的文档化特性是影响可维护性的重要因素

5.1.4.分类

语言 级别

低级语言（机器语言、汇编语言等）

高级语言（C、Java等）

应用 范围

通用语言（PASCAL，并发PASCAL，
MODULA，XCY，ADA等）

专用语言（BLISS语言专用于PDP-10机
器）

用户 要求

过程式语言（FORTRAN、COBOL、
PASCAL、C、Basic等）

非过程式语言（VB、VC、C#等）

成分 性质

顺序语言（C、COBOL等）

并发语言（JR、ConcurrentC++等）

分布式语言（DCDL、Elang、
Scala等）

使用 方式

交互式语言（BASIC等）、

非交互式语言（FORTRAN、
COBOL、ALGOL69、PASCAL、
C等）

5.1.4.分类——发展历程

- 机器指令代码、不同机器语言不同
- 二进制形式
- 使用绝对地址
- 运行效率高、但是出错率高
- 可读性差

第一代语言：
机器语言

- 符号指令对应于机器指令
- 存储空间由机器安排
- 不同类型处理器具有不同汇编语言

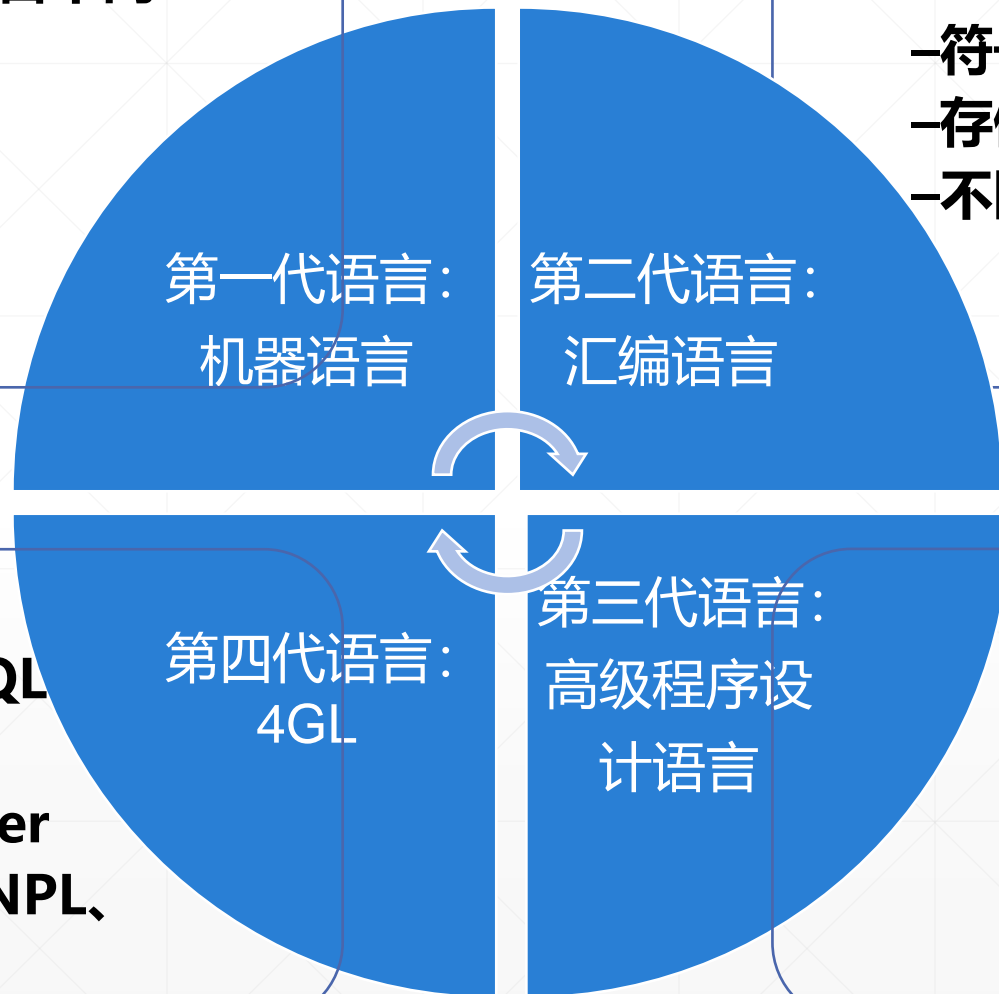
第二代语言：
汇编语言

- 数据库领域
- 查询语言和报表生成器：SQL
- 图形语言：SQL Windows
- 应用生成器：Power Builder
- 形式化规格说明语言：Z、NPL、SPECINT

第四代语言：
4GL

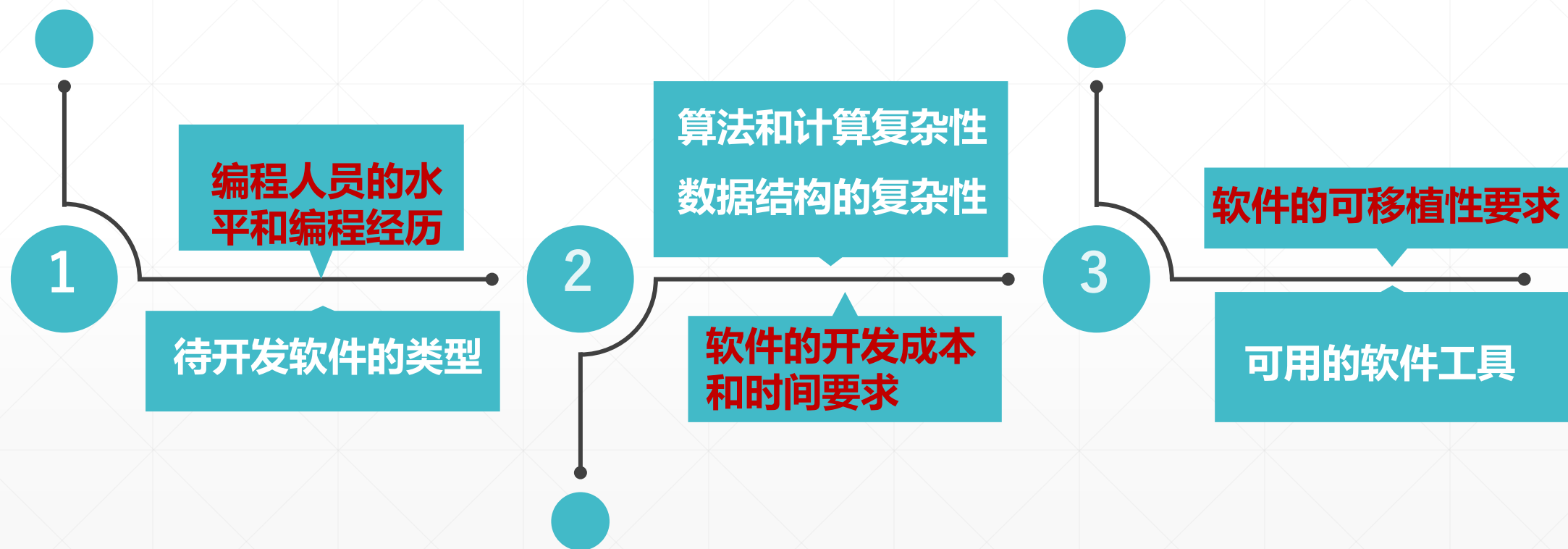
第三代语言：
高级程序设计语言

- 从20世纪50年代开始
- Fortran、Basic、Pascal、C、C++、C#、Java等



5.1.4.分类——选择要素

根据应用要求和要求的相对重要性选择

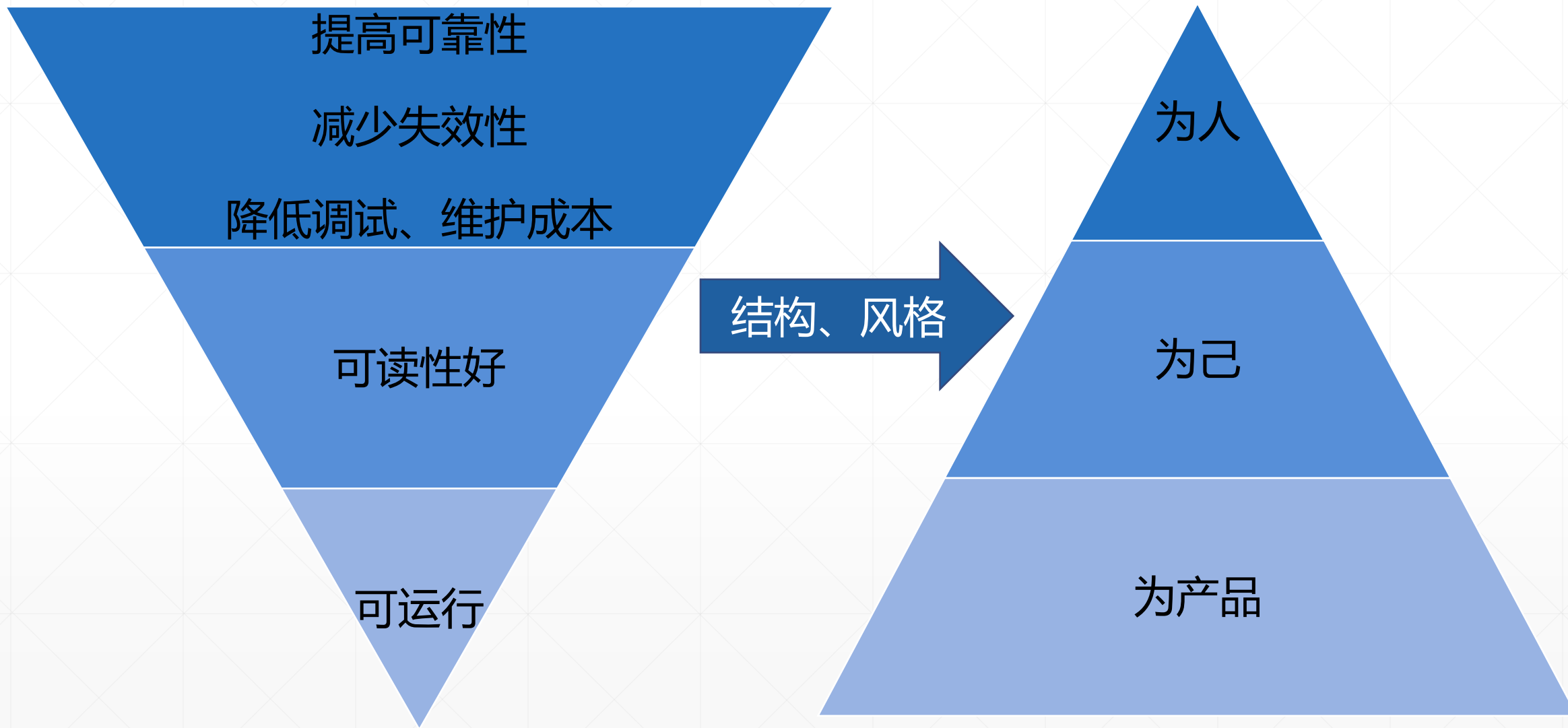




5.2. 编程习惯

- 重要性
- 结构化程序设计原则
- 程序设计风格

5.2.1.重要性



5.2.2.结构化程序设计原则

简单结构

- 尽量使用语言提供的基本控制结构，即顺序结构、选择结构和重复（循环）结构。

组合嵌套

- 复杂结构应该用基本控制结构组合或嵌套实现。

一致性

- 语言中没有的控制结构，可用一段等价的程序段模拟，但要求该程序段在整个系统中应前后一致。

块机制

- 将程序组织成容易识别的块，每块只有一个入口和一个出口。

GOTO语句

- 严格控制GOTO语句。

5.2.2.结构化程序设计原则

自顶向下

先考虑总体，后考虑细节；先考虑全局目标，后考虑局部目标

逐步细化

对于复杂问题，进行分解为子目标。

模块化

再将子目标分解为小目标，予以结构化实现。

自顶向下、逐步细化

[例] 用筛选法求100以内的素数。

[细化]：具体做法就是从2到100中去掉2, 3, ..., 9, 10的倍数，剩下的就是100以内的素数。

```
main( )
```

```
{
```

```
    建立2到100的数组A[ ], 其中A[i]= i;                -----1
```

```
    建立2到10的素数表B[ ], 存放2到10以内素数;        -----2
```

```
    若A[i] = i是B[ ]中任意一个数的倍数, 则剔除A[i];   -----3
```

```
    输出A[ ]中所有没有被剔除的数;                      -----4
```

```
}
```

5.2.3.程序设计风格

程序设 计风格

- 1) 基本要求
 - 2) 可读性要求
 - 3) 正确性与容错性要求
 - 4) 可移植性要求
 - 5) 输入和输出要求
 - 6) 重用性要求
-

5.2.3.程序设计风格—基本要求



程序结构清晰且简单易懂，单个函数的行数一般不要超过100行。



算法设计应该简单，代码要精简，避免出现垃圾程序。



尽量使用标准库函数（类方法）和公共函数（类方法）。



最好使用括号以避免二义性。

5.2.3.程序设计风格—可读性要求

注 释



程序头,函数头说明; 接口说明; 子程序清单,有关数据的说明; 模块位置; 开发历史等



主要变量(结构、联合、类或对象):含义的注释。



应保持注释与代码完全一致。



处理过程的每个阶段和典型算法前都有相关注释说明, 但是不要对每条语句注释。

5.2.3.程序设计风格—可读性要求

格 式

- 程序格式清晰:
- 一行只写一条语句, 不要密密麻麻, 分不出层次
- 显示程序的逻辑结构, 利用空格、空行和缩进进行, 缩进量一般为4个字符。

if (A<-17)AND NOT(B<=49)OR Cthen A=A-1
if A>100 thenA=A*2 endif elseA=A+1
endif
写成

```
if (A < -17) AND NOT (B <= 49) OR C
    then A=A-1
        if A>100
            then A=A*2
        endif
    else A=A+1
end if
```


5.2.3.程序设计风格—可读性要求

程序本身

语句力求简单、清晰，不要片面追求效率,程序编写得过于紧凑,使语句复杂化。

例如:

V 是一个 $N \times N$ 单位矩阵,

当 $i \neq j$ 时, $V(i,j)=0$;

当 $i = j$ 时, $V(i,j)=1$ 。

```
for (i=1; i<=n; i++)  
    for (j=1; j<=n; j++)  
        V[i][j] = (i/j) * (j/i)  
    else  
        V[i][j] = 0;
```

写成

```
for (i=1; i<=n; i++)  
    for (j=1; j<=n; j++)  
        if (i==j)  
            V[i][j] = 1;
```

5.2.3.程序设计风格—可读性要求

程序本身-续

- 简单变量的运算速度比下标（数组）变量的运算要快,程序员可能把语句:

$X = A[I] + 1/A[I]$

写成

$AI = A[I]; X = AI + 1/AI$

编程时尽可能使用已有的库函数。

尽量用公共过程或子程序代替具有独立功能的重复代码段。使用括号清晰地表达算术表达式和逻辑表达式的运算顺序。尽量使用三种基本控制结构编写程序,使用IF THEN ELSE结构实现分支;使用DO UNTIL或DO WHILE来实现循环。

避免采用过于复杂的条件测试,少用含有“否定”运算符的条件语句,例如:

IF NOT ((CHAR <= ' 0') OR (CHAR >= ' 9')) THEN

改成

IF (CHAR > ' 0') AND (CHAR < ' 9')
THEN

5.2.3.程序设计风格—可读性要求

程序本身-续

- 避免使用空的ELSE语句和IF THEN IF语句

```
IF(CHAR>=' A' ) THEN
```

```
IF(CHAR<=' Z' ) THEN
```

```
PRINT "This is a letter. "
```

```
ELSE//这个语句的配套IF逻辑上不明确
```

```
PRINT "This is not a letter. "
```

5.2.3.程序设计风格—可读性要求

程序本身-续



避免使用ELSE GOTO和ELSE RETURN结构。



避免过多的循环嵌套和条件嵌套。



数据结构要有利于程序的简化。



模块功能尽可能单一化,模块间的耦合能够清晰可见。利用信息隐蔽确保每一个模块的独立性。



对递归定义的数据结构尽量使用递归过程。



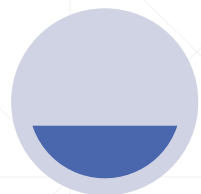
尽量不要修补结构差的程序,而应重新设计和编码。



对太大的程序,要分块编写、测试,后再集成

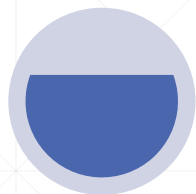
5.2.3.程序设计风格—可读性要求

数据说明



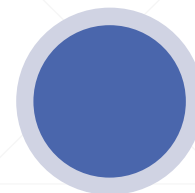
数据说明的先后次序规范化

简单变量类型说明、数组说明、公用数据块说明、文件说明



每个类型说明中可按如下顺序排列

整型量说明、实型量说明、字符量说明、逻辑量说明



同一条说明语句中可按字母顺序排列

**例如：INTEGER
cost, length,
price, width**

5.2.3.程序设计风格—正确性与容错性要求



程序首先是正确，其次是考虑优美和效率。



对所有的用户输入，必须进行合法性和有效性检查。



不要单独进行浮点数的比较。



所有变量在调用前必须被初始化。



改一个错误时可能产生新的错误，因此修改前首先考虑其影响。



单元测试也是编程的一部分，提交联调测试的程序必须通过单元测试。



单元测试时，必须针对类里的每一个public方法进行测试，测试其正确的输入，是否得到正确的输出；错误的输入是否有容错处理。

5.2.3.程序设计风格—可移植性要求



应当尽量使用语言的标准部分，避免使用第三方提供的接口，以确保程序不受具体的运行环境影响，和平台无关。



对数据库的操作，使用符合语言规范的标准接口类例如JDBC，除非程序是运行于特定的环境下，并且有很高的性能优化方面的要求。



程序中涉及到的数据库定义和操纵语句，尽量使用标准 SQL 数据类型和 SQL 语句

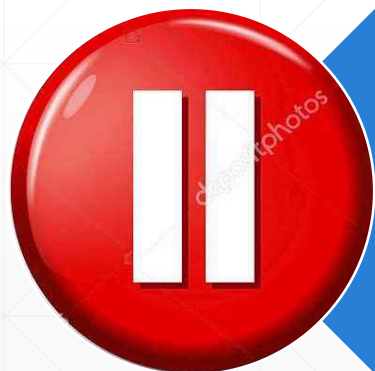
5.2.3.程序设计风格—输入和输出要求

- ✓ 任何程序都会有输入输出，输入输出的方式应当尽量方便用户的使用。在需求分析和设计阶段就应确定基本的输入输出风格，要避免因设计不当带来操作和理解的麻烦。
 - ✓ 对所有的输入数据进行检验，从而识别错误的输入，以保证每个数据的有效性。
 - ✓ 检查输入项各种重要组合的合理性，必要时报告输入状态信息。
 - ✓ 输入的步骤和操作尽可能简单，并且要保持简单的输入格式。
 - ✓ 有些输入信息应提供缺省值。
 - ✓ 输入一批数据时，最好使用输入结束标志，而不要由用户指定输入数据数目。
 - ✓ 在以交互式方式进行输入时，要显示提示信息、选择项和取值范围，便于操作。同时，在输入数据的过程和输入数据结束时，也要在屏幕上给出状态信息。
 - ✓ 当程序设计语言对输入格式有严格要求时，应保持输入格式与输入语句的要求一致。
 - ✓ 给所有的输出加上注解信息。
 - ✓ 按照用户的要求设计输出报表格式。
-

5.2.3.程序设计风格—重用性要求



可重复使用的、功能相对独立的算法或接口。应该考虑封装成公共的控件或类。



相对固定和独立的程序实现方式和过程，应考虑做成程序模板，增强对程序实现方式的复用

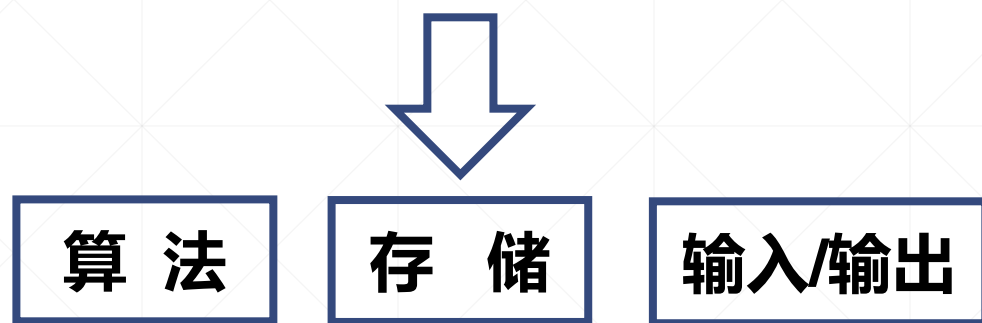


5.3. 程序的效率

- 概念
- 效率的影响因素

5.3.1.概念

程序的效率：程序的执行速度及程序所需占用的存储空间。



程序编码是最后提高运行速度和节省存储的机会，因此在此阶段不能不考虑程序的效率。

5.3.1.概念

程序效率的几条准则

效率是一个性能要求，应当在需求分析阶段给出。软件效率以需求为准，不应以人力所及为准。

好的设计可以提高效率。

程序的效率与程序的简单性相关。

一般说来，任何对效率无重要改善，且对程序的简单性、可读性和正确性不利的程序设计方法都是不可取的。

5.3.2.效率的影响因素——算法

源程序的效率与详细设计阶段确定的算法的效率直接有关。在设计翻译转换代码，算法效率反映为程序的执行速度和存储容量的要求。

设计向程序转换过程中的指导原则

- 简化** 在编程序前，尽可能化简有关的算术表达式和逻辑表达式；
- 检查** 仔细检查算法中的嵌套的循环，尽可能将某些语句或表达式移到循环外面；
- 选用** 选用等效的高效率算法；
- 采用** 采用“快速”的算术运算；尽量采用整数算术表达式和布尔表达式；
- 避免** 尽量避免使用多维数组；尽量避免使用指针和复杂的表；不要混淆数据类型，避免在表达式中出现类型混杂；

5.3.2.效率的影响因素——存储器

大中型计算机系统,存储限制不是主要问题

内存分页功能的
虚拟存储管理。
效率与系统的分
页功能相关。

结构化程序设计,
将程序功能合理
分块, 模块 (群)
体积与页容量相
匹配, 减少调度。

在微型计算机系统,
存储容量对软件设计
和编码的制约很大。

选择可生成较短目
标代码且存储压缩
性能优良的编译程
序, 甚至汇编程序。

提高存储器效
率的关键是程
序的简单性。

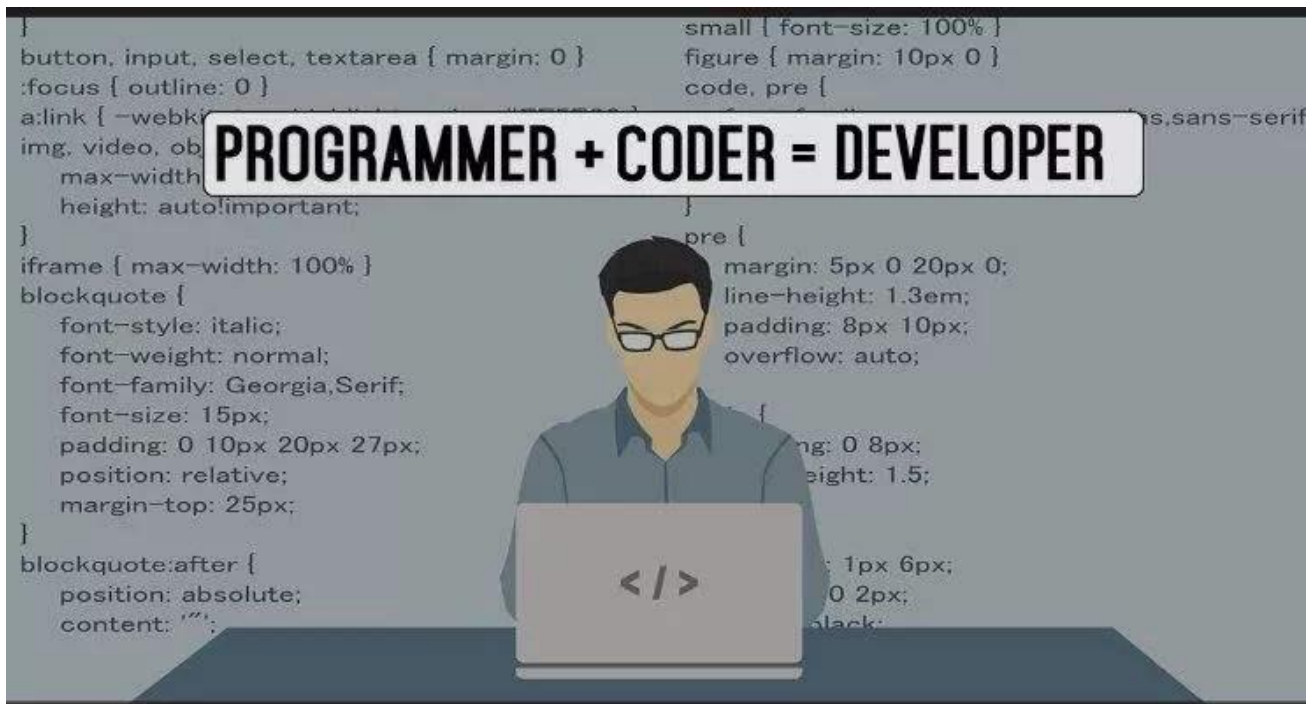
5.3.2.效率的影响因素——输入/输出

面向人(操作员)的输入 / 输出

**方便、简单地录入输入数据
直观、一目了然地了解输出信息**

面向设备的输入 / 输出

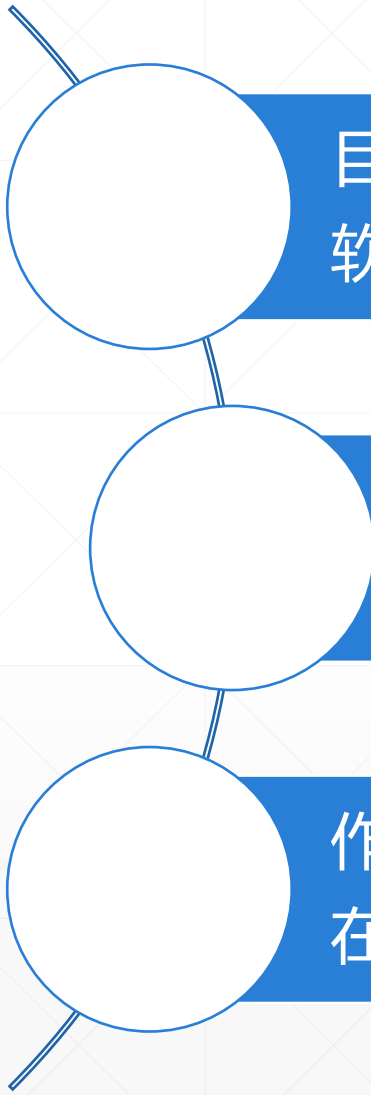
**输入/输出的请求应当最小化;
所有的操作, 安排适当的缓冲区, 以减少交换。
对辅助存储, 选择尽量简单, 可接受的存取方法
对辅助存储的输入/输出, 应当成块传送;
对终端或打印机的输入/输出, 应考虑设备特性,
尽可能改善输入/输出的质量和速度;
任何不易理解的, 对改善输入/输出效果关系不大的措施都是不可取的;
任何不易理解的所谓“超高效”的输入/输出是毫无价值的;**



5.4. 编码管理

- 版本管理的意义
- 常见软件版本
- 常见版本命名规则
- 面向对象程序设计
- 编码策略
- 编码规范

5.4.1.版本管理的意义



目的：标识、控制和追踪软件开发和实施过程中产生的各种软件产品版本。

适用范围：适用于软件源代码、产品版本的管理。通常内部使用，不对外发布。

作用：主要是开发者自己对产品进行测试，检查产品是否存在缺陷、错误，验证产品功能与说明书、用户手册是否一致。

5.4.2.常见软件版本

Alpha版

内部测试版，用作内部测试

Beta版

外部测试版，典型性用户的外部测试

Demo版

演示版，演示部分功能，为发售造势

Enhanced版

增强版或加强版，增加了新功能、新游戏场景的正式发售版本。

Free版

自由版，没有版权，免费给大家使用的版本。

Full Version版

完全版，最终正式发售的版本。

Shareware版

共享版，为了吸引客户，带有限制的版本。

Release版等。

发行版，可从Internet上免费下载

5.4.3.常见版本命名规则



目前主要有三种版本控制工具：CVS、SVN、Git.

5.4.4.面向对象程序设计

主要概念

对象、类、数据抽象、继承、动态绑定、数据封装、多态性、消息传递。

含义

面向对象程序设计是以建立模型体现出来的抽象思维过程和面向对象的方法。

02

03

方法

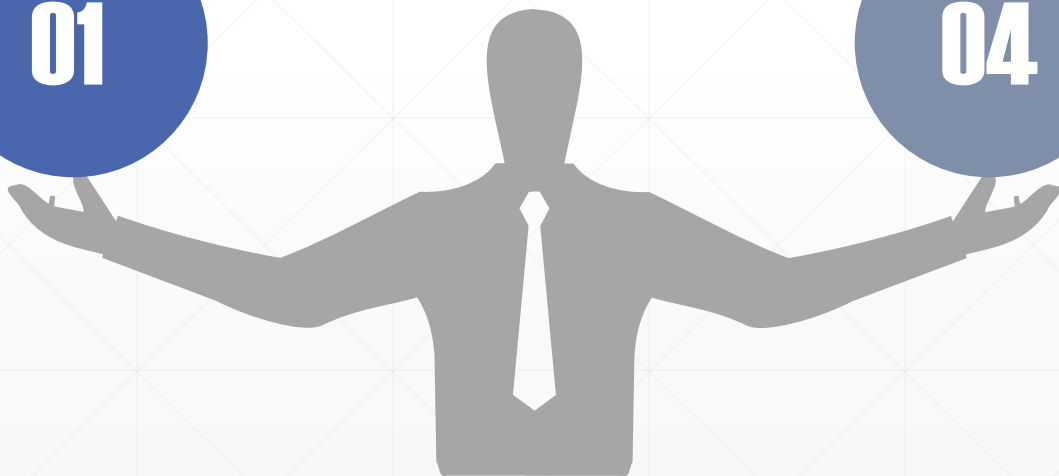
选择程序设计语言、类的实现、方法的实现、用户接口的实现等

01

04

语言

C++、JAVA、VB、VC++、C#



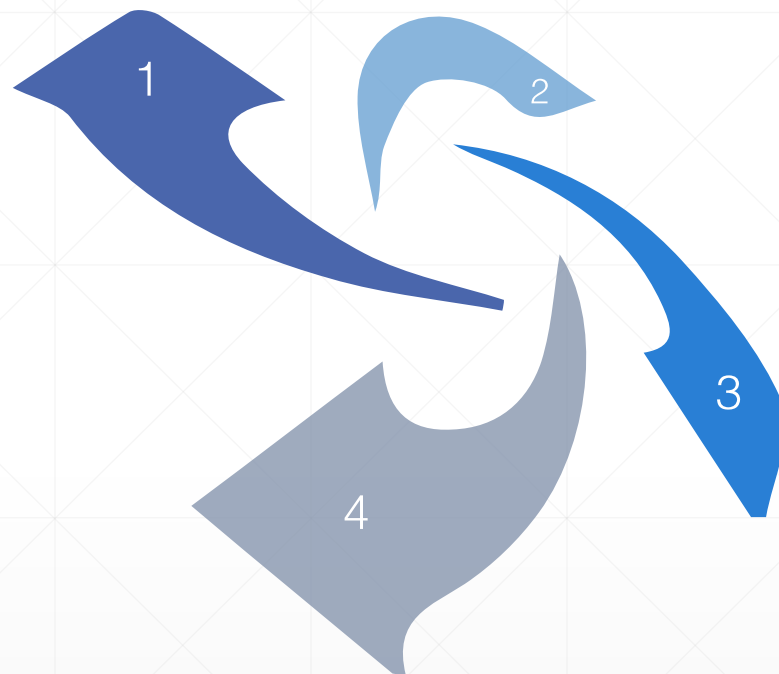
5.4.5.编码策略

自顶向下

从模块的最高层次逐步向下编码

自底向上

从类继承的底层开始向上编码



复合模式

自顶向下与自底向上相结合
可以先总体界面、交互，到
基础模块

线程模式

线程：执行关键功能的最小
模块集合
先线程，级关键构建，后其
他模块

5.4.6.编码规范

编码规范：一个团队、企业给出的内部开发最佳做法的建议、最好方式的推荐和经验总结。

使开发人员有据可依

代码易读

易于测试和维护

易于定位错误、变更管理

