CHAPTER 6

Coding or Implementation

Outline

(1) Introduction to coding

(2) Choice of programming language

(3) Coding standards or styles

(4) Measurement of program complexity

Coding 概述

- 软件工程过程的一个阶段，程序编码是设计的继续。
- 编码是把软件设计结果转换成程序设计语言书写的程序的过程。
- 为了保证程序编码的质量，程序员必须深刻理解、熟练掌握并正确地运用程序设计语言的特性。
- 要求源程序具有良好的结构性和良好的程序设计风格。

好的代码

- 程序质量主要取决于软件设计的质量
- **Good code**  高质量程序
  - ✓ **Simple**
  - ✓ **Readable**
  - ✓ **Maintainable**

Two decisions in coding phase

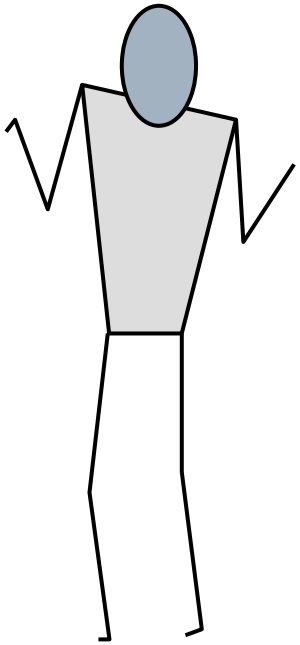
- 编码阶段的两个重要决策
  - ✓ **Choice of programming language**
  - ✓ **Coding standards or styles**

怎样选择编程语言？

- Catalogs of programming language
- Characteristics of programming language
- Experience in using programming language

Languages & Coding

Ada, Visual Basic, C, C++, Java, Cobol, Python, Fortran, Lisp, PL/1, Prolog, Modula, …



究竟选择哪个？

编程语言的分类

（1）Machine language

（2）Assembler language

（3）High level language

From application point of view

科学工程计算: Basic, Fortran, Pascal, C,PL/1, C++

数据处理和数据库: Cobol, SQL, 4GL

实时处理: 汇编语言x86，Ada

系统软件: C, Pascal, Ada

人工智能: Lisp, Prolog

面向对象编程： C++, Java, Dephi, Vb

组件编程：Corba, Dcom, EJB

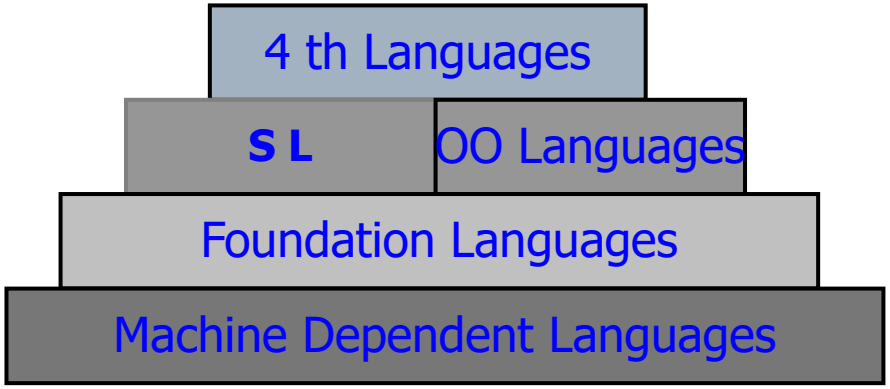
网络浏览: Asp, Jsp, PHP

具体语言的应用领域

Language	Approximate Date of Introductaon	General Application Areas
FORTRAN	1957	Numerically oriented language Most applicable to scientific, mathematical, and statistical problem areas. Very widely used and very widely available.
ALGOL	1960	Also a numerically oriented language but with new language features. Widely used in Europe.
COBOL	1960	The most widely used business-oriented computer language.
LISP	1961	Special-purpose language developed primarily for list processing and symbolic manipulation. Widely used in the area of artificial intelligence.
SNOBOL	1962	Special-purpose language used primarily for character string processing. This includes applications such as text editors, language processors, and bibliographic work.
BASIC	1965	A simple interactive programming language widely used to teach programming in high schools and colleges. It is an interpretive language
PL/1	1965	An extremely complex, general-purpose language designed to incorporate the numeric capabilities of FORTRAN, the business capabilities of COBOL, and many other features into a single language.
APL	1967	An operator-oriented interactive language that introduced a wide range of new mathematical operations that are built directly into the language.
Pascal	1971	A general-purpose language designed specifically to teach the concepts of computer programming and allow the efficient implementation of large programs.
Ada	1980	A new systems implementation language designed and built for the Department of Defense.
C	1970	An extremely flexible and powerful language, capable of both low-level (close to assembler) and high-level (structured) programming. Another strength of C is its adaptability to diverse applications and operating systems. C is a typed, structured, and poi
C++	1983	A powerful, object-oriented, strongly typed programming language (a superset of C). C++ fully supports (multiple) inheritance, polymorphism, and encapsulation.
Visual Basic	1991	One of the successors of BASIC. An object-based, visual programming, and RAD language for Microsoft Windows applications.
Java	1995	An object-oriented programming language (a successor but not superset of C++), featuring powerful automatic memory management, bound checking, and strong typing. Contrary to C/C++, it is an interpretive language. Its small size and platform independent su

4 th Generation language

High level language
+ User interface
+ Databases



目的是使得一般的应用软件编程实现简单和迅速

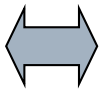
Turbo C, Visual Basic, Dephi, Forms

What is 5 th Generation language ?

编程语言选择

语言方面：

- 名字说明
- 类型说明
- 选择控制结构
- 循环控制结构
- 程序对象的局部性
- 变量的局部共享
- 异常处理
- 独立编译
- 解释型，编译型

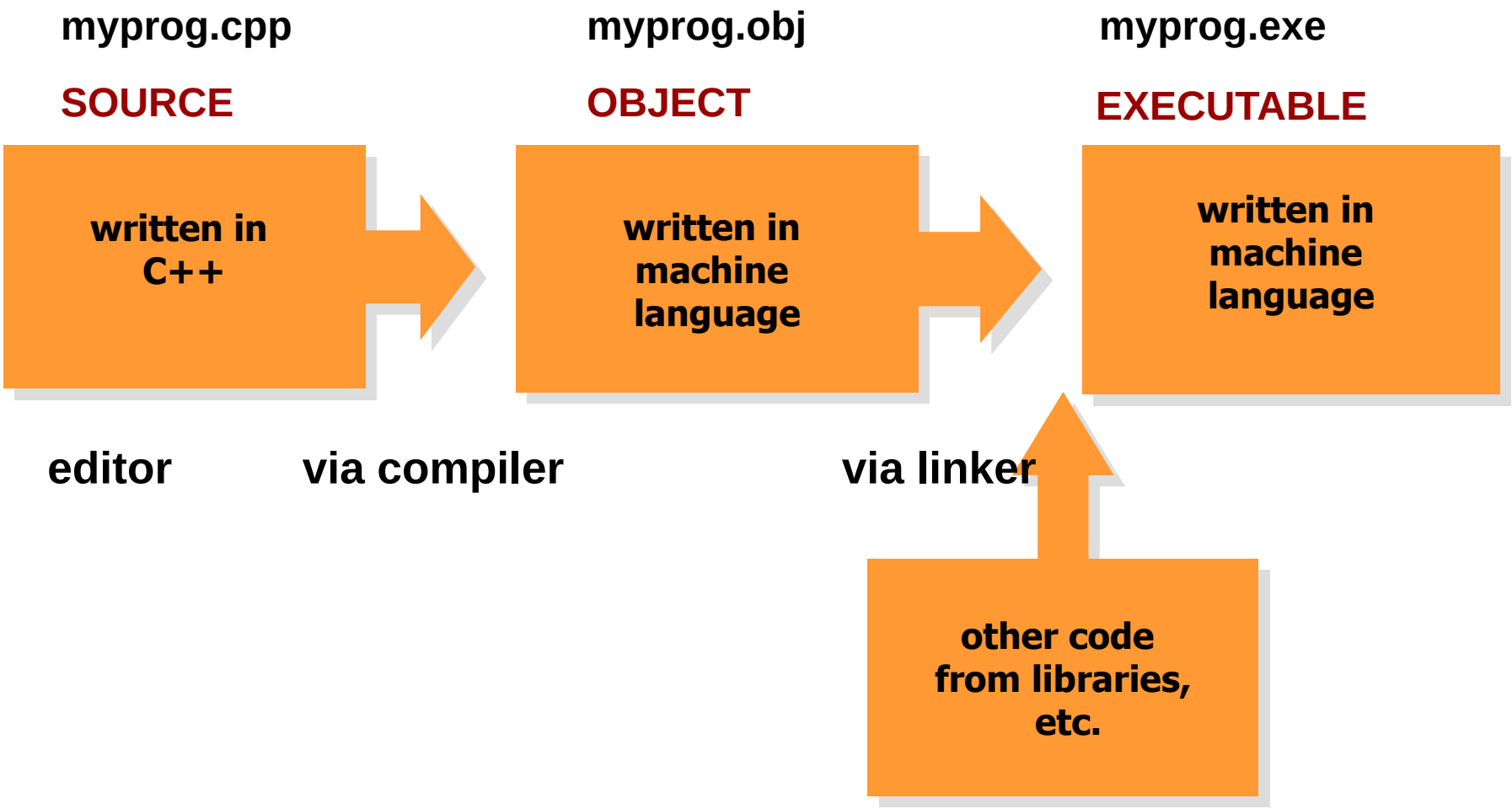


考虑方面：

- ✓ 项目的应用领域
- ✓ 算法和计算复杂性
- ✓ 软件的执行环境
- ✓ 性能因素
- ✓ 数据结构的复杂性
- ✓ 软件开发人员的水平
- ✓ 可用的编译系统和**IDE**工具

Software tools

C++ Program Stages



Kinds of Software Tools

- 机械工具能够放大人的体力
- 软件工具能够放大人的智力

IDE ?

CASE ?

Power Designer ?

SDK ?

…… photoshop, mapinfo, CAD,….

程序开发环境应该具备的特性

- ✓ 通用性：适用于不同的语言、不同的应用领域和开发方法；
- ✓ 适应性：通过开关设置，能配制出不同需要的程序设计支撑环境实例；
- ✓ 开放性：能方便地增加新工具；
- ✓ 支持复用：能支持可复用模块的存储、索引和查找；
- ✓ 自控性：保证自身操作的正确与协调；
- ✓ 自带数据库：提供数据库机制，存储、管理已开发的软件产品；
- ✓ 保证质量：有助于提高所开发软件的质量；
- ✓ 吸引用户：用户愿意使用；
- ✓ 具有市场竞争力：能真正提高软件生产力。

编码标准和风格

What are Coding Standards?

- ✓ **Coding standards are guidelines for code and documentation.**
- ✓ **The dream is that any developer familiar with the guidelines can work on any code that followed them.**
- ✓ **Standards range from a simple series of statements to involved documents.**

为什么需要编码规范 ？

程序实际上也是一种供人阅读的文章，有文章书写阅读的风格问题。应该使程序具有良好的风格。

- ✓ **Create consistency between developers**
- ✓ **Communicate among team**
- ✓ **Easier to develop and maintain codes**
- ✓ **Save time and money**

Why code standard ?

- ✓ 使程序员进行“无私程序设计”
- ✓ 避免程序员与其产生的代码之间的关系过于密切
- ✓ 多个程序员写的代码如同一个人所写
- ✓ 不是追求“程序美的地方”
- ✓ 提高程序代码的规范化程度
- ✓ 使程序代码易读、易懂、易修改，重用
- ✓ 实现程序员之间相互进行程序测试和维护

编码风格的具体体现

1. 标识符（符号名、变量名）的风格
2. 注释的风格
3. 数据说明的风格
4. 语句结构的风格
5. 输入 / 输出的风格
6. 程序layout风格

1、标识符的命名风格

- 标识符：常量名，变量名，统称为符号名
- 标识符：文件名、模块名、程序名、函数名、过程名、变量名、数据区名、缓冲区名、语句序号名等。
- 命名（取名）规则：读名知义，名字应能反映它所代表的实际东西，应有一定实际意义。

  例如，表示次数的量用*Times*，表示总量的用*Total*，表示平均值的用*Average*，表示和的量用*Sum*等。

  例如：不用a，b，c，……

  不用i，j，k ，x,y,z,……

1、标识符的命名风格

- 名字不长不短，应当选择精炼的、意义明确的名字。宜长不宜短，长比短好。

- 名字唯一，在一个程序中，一个变量名字只作一种用途。

  不要：时而为整型变量，时而为实型变量

  时而为函数名，时而为语句标号

2、注释的风格

- 注释：表达设计用意的自然语言说明性文字，编译时自动忽略。
- 夹在程序中的注释是程序员与日后的程序读者之间通信的重要手段。
- 注释决不是可有可无的。
- 一些正规的程序文本中，注释行的数量占到整个源程序的1／3到1／2，甚至更多。
- 注释分为序言性注释和功能性注释。

序言性注释

- 通常置于每个程序模块的开头部分，给出程序的整体说明，对于理解程序本身具有引导作用。有些软件开发部门对序言性注释做了明确而严格的规定，要求程序编制者逐项列出如下项：

✓ 程序标题

✓ 有关本模块功能和目的的说明

✓ 主要算法

✓ 接口说明：包括调用形式，参数描述，子程序清单；

✓ 有关数据描述：重要的变量及其用途，约束或限制条件，以及其它有关信息；

✓ 模块位置：在哪一个源文件中，或隶属于哪一个软件包；

✓ 开发简历：模块设计者，复审者，复审日期，修改日期及有关说明等。

功能性注释

功能性注释： 嵌在源程序体中，用以描述其后的语句或程序段是在做什么工作，或是执行了下面的语句会怎么样。而不要解释下面怎么做。

举例：如果注明把月销售额计入年度总额，便使读者理解了下面语句的意图：

```
    /* Add monthly-sales to  annual total */
total =  amount +total
```

3、数据说明的风格

- 为了使程序中数据说明更易于理解和维护，必须注意以下几点。
  - ✓ 数据说明的次序应当规范化
  - ✓ 说明语句中变量安排有序化
  - ✓ 使用注释说明复杂数据结构

数据说明举例

例如：在FORTRAN程序中数据说明次序

① 常量说明
② 简单变量类型说明
③ 数组说明
④ 公用数据块说明
⑤ 所有的文件说明

可按如下顺序进一步要求：

① 整型量说明
② 实型量说明
③ 字符量说明
④ 逻辑量说明

数据变量定义有序化

当多个变量名在一个说明语句中说明时，应当对这些变量按字母的顺序排列。

例如：

integer ***size, length, width, cost, price***

integer ***cost, length, price , size, width***

4、语句结构的风格

(1) 在一行内只写一条语句

- ✓ 在一行内只写一条语句，并且采取适当的移行和缩进格式，使程序的逻辑、层次和功能变得更加明确。
- ✓ 许多程序设计语言允许在一行内写多个语句。但这种方式会使程序可读性变差。因而不可取。

例如：排序程序

```
FOR I:=1 TO N－1 DO BEGIN T:=I；FOR J:=I＋1 TO N DO
IF A[J]＜A[T] THEN T:=J；IF T≠I THEN BEGIN
WORK:=A[T]；A[T]:=A[I]；A[I]:=WORK；END  END；
```

- 由于一行中包括了多个语句，掩盖了程序的循环结构和条件结构，使其可读性变得很差。

```
FOR  I:=1  TO  N-1  DO     //改进布局
    BEGIN
        T:=I;
        FOR  J:=I + 1  TO  N  DO
            IF  A[J] < A[T]  THEN  T:=J;
        IF  T≠I  THEN
          BEGIN
            WORK:=A[T];
            A[T]:=A[I];
            A[I]:=WORK;
          END
    END;
```

(2) 程序编写首先应当考虑清晰性

程序编写首先应当考虑清晰性，不要刻意追求技巧性，使程序编写得过于紧凑。

例如：有一个用 C 语句写出的程序段，实现交换功能

```
A[I] = A[I]+A[T];
A[T] = A[I]-A[T];
A[I] = A[I]-A[T];
```

改写出：

```
WORK = A[T];
A[T] = A[I];
A[I] = WORK
```

(3) 程序要能直截了当地说明程序员的用意

程序编写得要简单，写清楚，直截了当地说明程序员的用意。

例如：

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

C语言中，除法运算（/）在除数和被除数都是整型量时，其结果只取整数部分，而得到整型量。

写成以下的形式，就能让读者直接了解程序编写者的意图。

```
for ( i=1; i <= n; i++ )
    for ( j=1; j <= n; j++ )
        if ( i == j )
            V[i][j] = 1.0;
        else
            V[i][j] = 0.0;
```

(4) 除非对效率有特殊的要求，　程序编写要做到

清晰第一，效率第二。不要为了追求效率而丧失了清晰性。事实上，程序效率的提高主要应通过选择高效的算法来实现。

(5) 首先要保证程序正确

保证程序正确，然后才要求提高速度。反过来说，在使程序高速运行时，首先要保证它是正确的。

(6) 避免使用临时变量而使可读性下降

例如：有的程序员为了追求效率，往往喜欢把表达式

```
A[I]＋1／A[I];
```

写成：

```
AI＝A[I];
X＝AI＋1／AI;
```

这样将一句分成两句写，会产生意想不到的问题。

（7）让编译程序做简单的优化

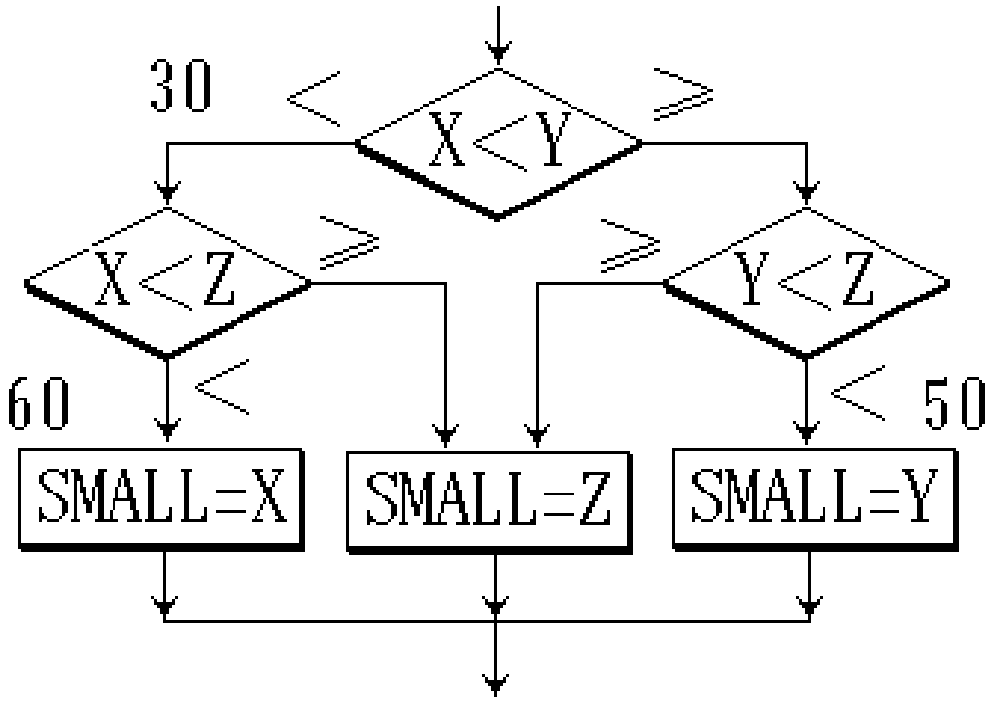
（8）尽可能使用库函数

（9）避免不必要的转移

同时如果能保持程序可读性，则不必用GOTO语句。

例如：有一个求三个数中最小值的程序：

```
      IF ( X <  Y )  GOTO 30
       IF (Y <  Z)  GOTO 50
          SMALL = Z
          GOTO 70
30   IF ( X  <  Z)  GOTO 60
         SMALL = Z
         GOTO 70
50   SMALL = Y
     GOTO 70
60  SMALL = X
70  CONTINUE
```



程序只需编写成：

```
small=x;
if ( y < small ) small=y;
if ( z < small ) small=z;
```

所以程序应当简单，不必过于深奥，避免使用GOTO语句绕来绕去。

(10) 避免使用空的else

这种结构容 易使读者产生误解。例如，

```
if ( char >= 'a' )
if ( char <= ' z' )
        cout << "This is a letter。";
 else
 cout << "This is not a letter。";
```

可能产生二义性问题。

(12) 避免采用过于复杂的条件测试

(13) 尽量减少使用“否定”条件语句

例如，如果在程序中出现

```
if ( !( char< ‘0’  || char > ‘9’ ) )
    ……
```

改成

```
if ( char >= '0’ && char <= '9’ )
    ……
```

不要让读者绕弯子想

(14) 尽可能用通俗易懂的伪码来描述程序的流程，然后再翻译成必须使用的语言。

(15) 尽量单入口单出口。

(16) 要模块化，使模块功能尽可能单一化，模块间的耦合能够清晰可见

(17) 利用信息隐蔽，确保每一个模块的独立性

(18) 从数据出发去构造程序

(19) 不要修补不好的程序，要重新编写。也不要一味地追求代码的复用，要重新组织。

(20) 对太大的程序，要分块编写、测试，然后再集成。

(21) 对递归定义的数据结构尽量使用递归过程

……

5、输入和输出的风格

- ✓ 输入和输出信息跟用户的操作直接相关。输入和输出的方式和格式应当尽可能方便用户的使用。一定要避免因设计不当给用户带来的麻烦。
- ✓ 在软件需求分析阶段和设计阶段，就应基本确定输入和输出的风格。系统能否被用户接受，有时就取决于输入和输出的风格。
- ✓ 不论是批处理的输入／输出方式，还是交互式的输入／输出方式，在设计和编码时都应考虑下列原则：

（1）对所有的输入数据都要进行检验，识别错误的输入，以保证每个数据的有效性

（2）检查输入项的各种重要组合的合理性，必要时报告输入状态信息

（3）使得输入的步骤和操作尽可能简单，并保持简单的输入格式

（4）输入数据时，应允许使用自由格式输入

（5）应允许缺省值

（6）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

（7）在交互式输入输入时，要在屏幕上使用提示符明确提示交互输入的请求，指明可使用选择项的种类和取值范围。同时，在数据输入的过程中和输入结束时，也要在屏幕上给出状态信息

（8）当程序设计语言对输入／输出格式有严格要求时，应保持输入格式与输入语句的要求的一致性；

（9）给所有的输出加注解，并设计输出报表格式。

输入／输出风格还受到许多其它因素的影响。如输入／输出设备（例如终端的类型，图形设备，数字化转换设备等）、用户的熟练程度、以及通信环境等。

6、视觉组织、 空格、空行和移行

- ✓ 恰当地利用空格，可以突出运算的优先性，避免发生运算的错误。

  例如 ：(A＜－17)ANDNOT(B＜＝49)ORC
  (A＜－17)　AND　NOT (B＜＝49) OR　C

- ✓ 程序段之间可用空行隔开。

- 向右缩格。它是指程序中的各行不必都在左端对齐，都从第一格起排列。这样做使程序完全分不清层次关系。

- 对于选择语句和循环语句，把其中的程序段语句向右缩进。使程序的逻辑结构和层次性更加清晰。

- 例如，两重选择结构嵌套，写成下面的移行形式，层次就清楚得多。

```
if (…) then
        if (…) then
            ……
        else
            ……
        endif      ……
else
    ……
    ……
endif
```

程序风格的一个例子（本实验室）

Coding Style for C、Java、Jsp in our project

Measurement of program complexity

- 程序复杂性： 模块内程序代码的复杂程度，例如行数，if 条件判断的个数，loop循环的圈数等。
- Why? 它直接关联到软件开发费用的多少，开发周期的长短和软件内部潜伏错误的多少。
- 减少程序复杂性，可提高软件的简单性和可理解性，软件开发费用减少，开发周期缩短，软件内部潜藏错误减少。跟测试标准有直接关系。

算法的复杂性

- 程序复杂性 VS. 算法的复杂性
- 算法的时间复杂性：（执行的步数）

  冒泡排序：$O(n^2)$

  矩阵相乘：$O(n^3)$

- 算法的空间复杂性：（占用内存空间的多少）

复杂性度量方法

- 代码行度量法
- McCabe度量法
- Halstead的软科学法

代码行度量法的依据

➢ 普遍经验

- ✓ 程序复杂性随着程序行数和规模的增加不均衡地增长。
- ✓ 控制程序规模的方法最好是采用分而治之的办法，将一个大程序分解成若干个简单的可理解的程序段。
- ✓ 统计一个程序模块的源代码行数目，并以源代码行数做为程序复杂性的度量。

代码行度量法在测试中的作用

- Thayer指出，程序出错率的估算范围是从0.04%～7%之间，即每100行源程序中可能存在0.04～7个错误。他还指出，每行代码的出错率与源程序行数之间不存在简单的线性关系。
- Lipow指出，对于小程序，每行代码出错率为1.3%～1.8%；对于大程序，每行代码的出错率增加到2.7%～3.2%之间，这只是考虑了程序的可执行部分，没有包括程序中的说明部分。

代码行度量法不足？

McCabe度量法

- McCabe度量法，又称环路复杂性度量，是一种基于程序控制流的复杂性度量方法。
- 它基于一个程序模块的程序图中环路的个数，因此计算它时，首先要画出程序图。
- 程序图是退化的程序流程图。流程图中每个处理都退化成一个结点，流线变成连接不同结点的有向弧。

McCabe度量法

- 程序图仅描述程序内部的控制流程，完全不表现对数据的具体操作，以及分支和循环的具体条件。

- 计算环路复杂性的方法：根据图论，在一个强连通的有向图G中，环的个数由以下公式给出：

$$V(G)=m-n+p$$

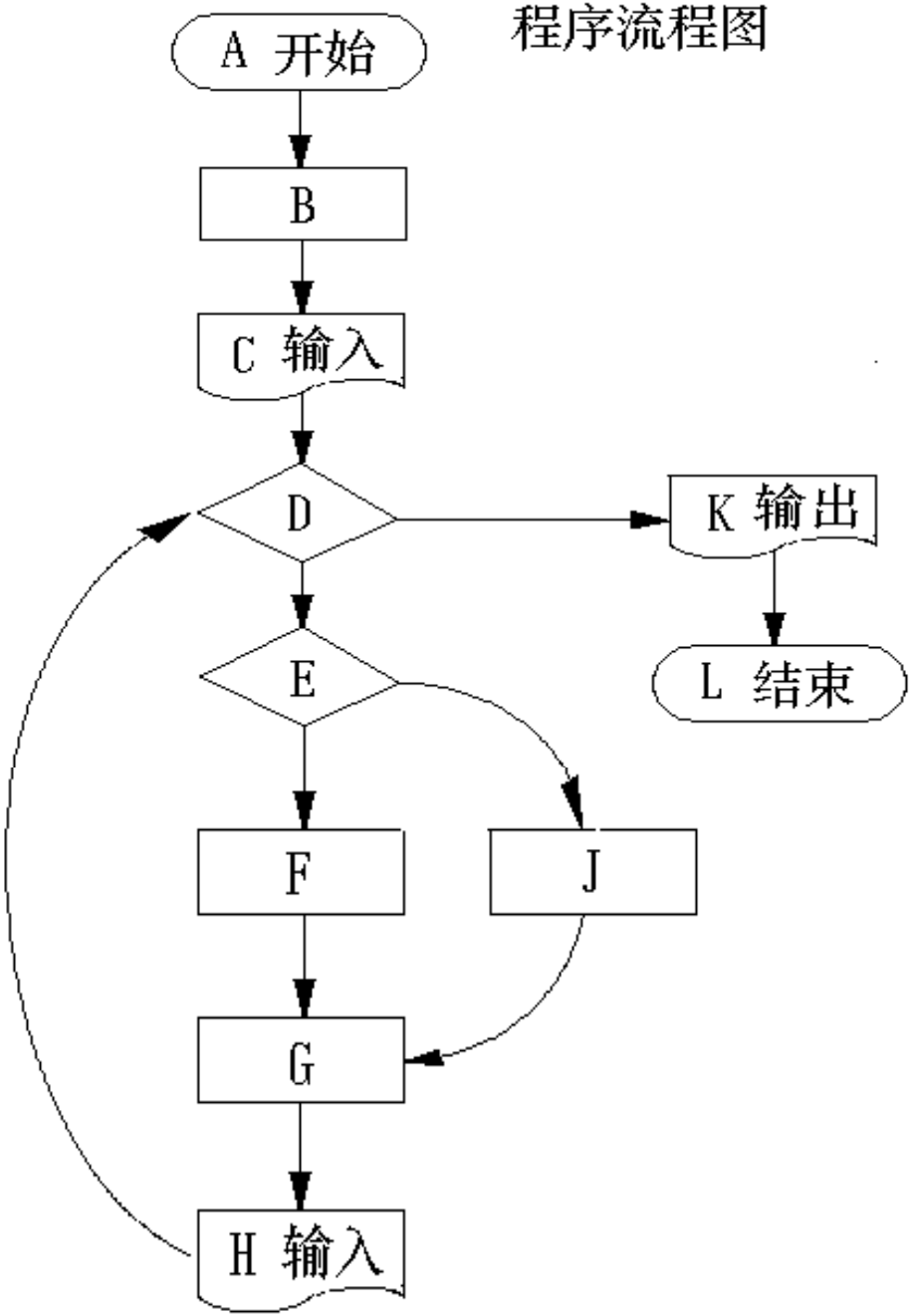
其中，V(G)是图G中环路个数，m 是图G中弧数，n 是图G中结点数，p 是图G中的强连通分量个数。

McCabe度量法

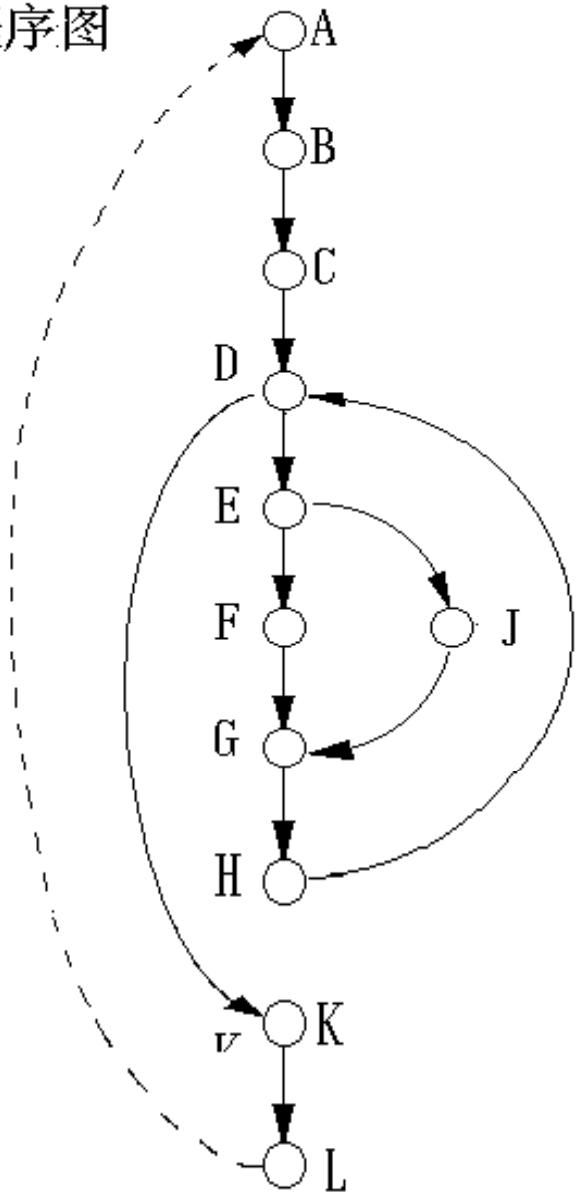
- 为使图成为强连通图，从图的入口点到出口点加一条用虚线表示的有向边，使图成为强连通图。这样就可以使用上式计算环路复杂性。
- 在例示中，结点数n＝11，弧数m＝13，p＝1，则有

  V(G)＝m－n＋p＝13－11＋1＝3.
- V(G)等于程序图中弧所封闭的区域数。
- V(G)等于程序图中线性无关的圆圈数。

程序流程图



程序图



几点说明

- 环路复杂度取决于程序控制结构的复杂度。当程序的分支数目或循环数目增加时其复杂度也增加。环路复杂度与程序中覆盖的路径条数有关。
- 环路复杂度是可加的。例如，模块A的复杂度为3，模块B的复杂度为 4，则模块A与模块B的复杂度是7。

几点说明

- McCabe建议，对于复杂度超过10的程序，应分成几个小程序，以减少程序中的错误。Walsh用实例证实了这个建议的正确性。在McCabe复杂度为10的附近，存在出错率的间断跃变。

- McCabe环路复杂度隐含的前提是：错误与程序的判定加上例行子程序的调用数目成正比。加工复杂性、数据结构、录入与打乱输入卡片的错误可以忽略不计。

缺点

- 对于不同种类的控制流的复杂性不能区分
- 简单IF语句与循环语句的复杂性同等看待
- 嵌套IF语句与简单CASE语句的复杂性是一样的
- 模块间接口当成一个简单分支一样处理
- 一个具有1000行的顺序程序与 1 行语句的复杂性相同

Halstead的软科学法

- Halstead 软科学法研究确定计算机软件开发中的一些定量规律，它采用一组基本的度量值，对程序复杂性进行度量。
- 这组度量值通常在程序产生之后得出，或者在设计完成之后估算出。
- 度量值？

Halstead程序长度

- 度量值
- 操作码，操作数；运算符，运算对象
- 程序长度定义：

  在程序中，设 N1为实际出现的运算符总个数，N2为实际出现的运算对象总个数，则Halstead程序长度N定义为：

$$N = N1 + N2$$

- **在定义中，运算符包括：**

  算术运算符　　赋值符(=或:=)

  逻辑运算符　　分界符(，或；或:)

  关系运算符　　括号运算符

  子程序调用符　　数组操作符

  循环操作符等。

- **特别地，成对的运算符，例如**

  “begin…end”、“for…to”、

  “repeat …until”、“while…do”、

  “if…then…else”、“（…）”等都当做单一运算符。

举例:

FORTRAN语言写出的交换排序的例子

```
      SUBROUTINE SORT ( X, N )
        DIMENSION  X( N )
```

```
      IF ( N .LT. 2 ) RETURN
      DO  20  I=2, N
          DO  10  J=1, I
              IF ( X(I) .GE. X(J) ) GO TO 10
              SAVE = X(I)
              X(I) = X(J)
              X(J) = SAVE
10        CONTINUE
20    CONTINUE
      RETURN
      END
```

统计

运算符	计数	运算对象	计数
可执行语句结束	7	X	6
数组下标	6	I	5
＝	5	J	4
IF（ ）	2	N	2
DO	2	2	2
,	2	SAVE	2
程序结束	1	1	1
.LT.	1	n2＝7	*N2*＝22
.GE.	1		
GO TO 10	1		
*n*1=10	*N1*＝28		

计算

对于上面的例子，统计得到N1，N2，可以计算得：

$$N = 28+22 = 50$$

问题：

- ✓ 已知具体程序代码
- ✓ 不知具体程序代码

（在详细设计阶段，尚未进行编码实现）

能否预测未来程序的Halstead长度？

Halstead程序长度的预测方法

n1 表示程序设计时不同运算符(包括保留字)的个数，

n2 表示程序设计时不同运算对象的个数，

H 表示“程序长度”，则有

$H=n1\times\log_2 n1+n2\times\log_2 n2$

H是程序长度的预测值

问题：预测值是否等于实际值？

n1=10，n2=7

$H = 10 * \log_2 10+7 * \log_2 7 = 52.87$

N=50

Halstead长度的作用

- ✓ 分析程序的潜在错误数

- ✓ Halstead长度可用来预测程序中的错误。经验预测公式为：

$$B = (N1+N2)*\log_2(n1+n2) \ / \ 3000$$

B为该程序的错误数。

它表明程序中可能存在的差错 B 应与程序量V成正比。

Halstead的结论小节

✓程序的实际Halstead长度N，可以由词汇表n算出。即使程序还未编制完成，也能预先算出程序的实际Halstead长度N，虽然它没有明确指出程序中到底有多少个语句。

✓这个结论非常有用。经过多次验证，预测的Halstead长度与实际的Halstead长度是非常接近的。

Halstead度量的缺点

- ✓ 不区别自己编的程序与别人编的程序差异，这是与实际经验相违背的。这时应将外部调用乘上一个大于1的的常数Kf（应在1～5之间，它与文档资料的清晰度有关)。

- ✓ 没有考虑非执行语句。补救办法：在统计n1、n2、N1、N2时，可以把非执行语句中出现的运算对象，运算符统计在内。

- ✓ 在允许混合运算的语言中，每种运算符与它的运算对象相关。
- ✓ 如果一种语言有整型、实型、双精度型三种不同类型的运算对象，则任何一种基本算术运算符(＋、－、×、／)实际上代表了 $A_3^2 = 6$ 种运算符。在计算时应考虑这种因数据类型而引起差异的情况。

- ✓ 忽视了嵌套结构（嵌套的循环语句、嵌套IF语句、括号结构等)。一般地，运算符的嵌套序列，总比具有相同数量的运算符和运算对象的非嵌套序列要复杂得多。解决的办法是对嵌套结果乘上一个嵌套因子。