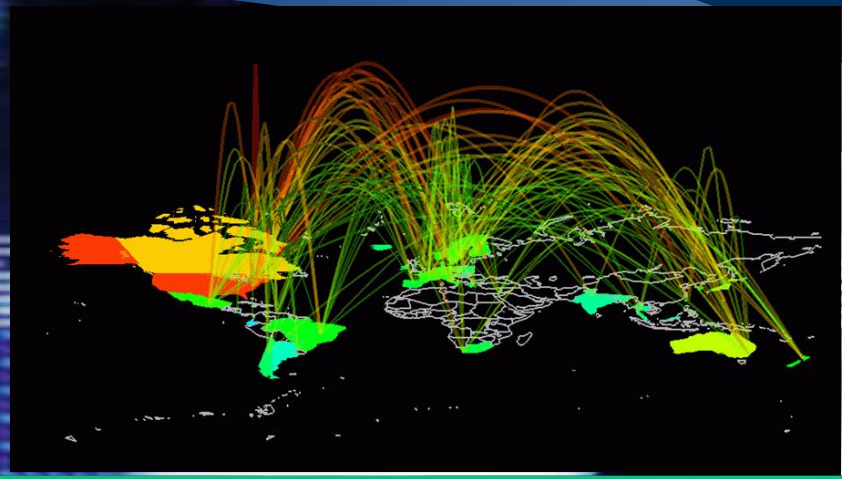
USTC

Chapter 22
Software ideals and history



王子磊 (Zilei Wang)
Email: zlwang@ustc.edu.cn
http://vim.ustc.edu.cn

历史和理念

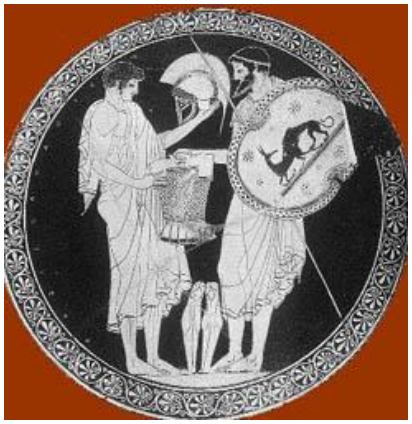
- 一种观点
  - 历史是一堆废话
- 另一种观点
  - 不能记住历史的人注定要重复
- 我们认为
  - 如果对历史没有一定了解，就不能达到专业水平
    - 如果你几乎不了解你所在领域的背景
      - 历史抛弃了那些貌似正确但不能良好工作的思想
        - » “过河拆桥”
  - 思想和理念对实际应用是至关重要的
    - 它们是历史的真正内容

什么是编程语言？

- 指示机器操作的一种工具
- 算法的符号表示
- 与其他程序员交流的工具
- 进行实验的工具
- 控制计算机控制器械的一种手段
- 控制电脑设备的一种手段
- 表达各种概念之间关系的一种方法
- 表达上层设计的一种方法

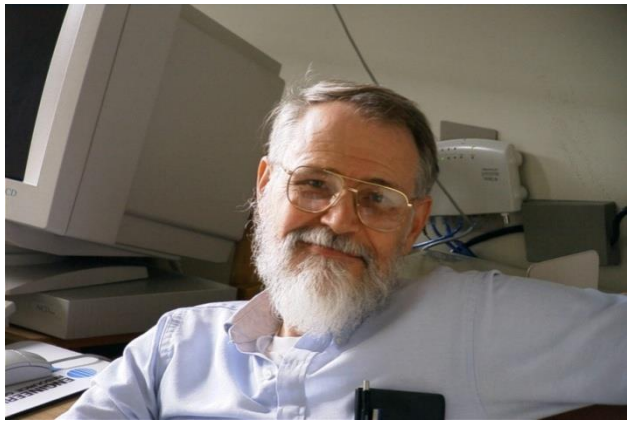
- 以上答案都对！
  - 而且还有其他答案

电脑英雄



- 每种文化和职业一定有它的理念和英雄
  - 物理： Newton, Einstein, Bohr, Feynman
  - 数学： Euclid, Euler, Hilbert
  - 医学： Hippocrates, Pasteur, Fleming

电脑英雄



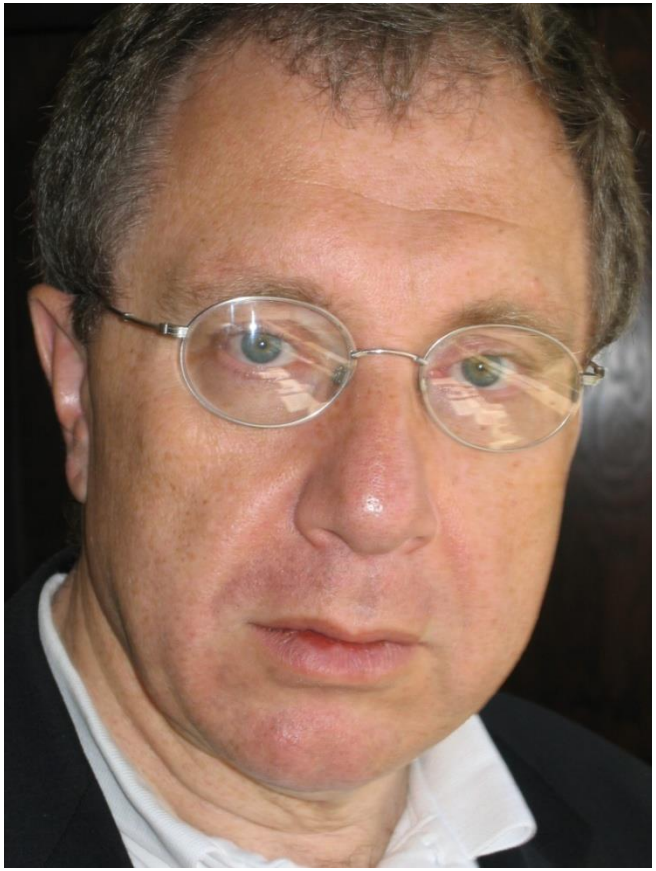
- Brian Kernighan
  - 杰出的程序员和作家
- Dennis Ritchie
  - C的设计和原始实现者
- 杰作：《The C Programming Language》，K&R

另一位电脑英雄

- Kristen Nygaard
  - Simula67 的合作发明者(与Ole-Johan Dahl)，以及面向对象编程和面向对象设计

还有一位…

- Alex Stepanov
  - STL的发起者和泛型编程的先驱



两种极端

- 学院派：漂亮、完美、纯洁
- 商业：便捷

- 两者的压力都是巨大的
- 两种极端一定要避免——折中
  - 它们主张结果的夸张(炒作)，以及替代者或先前语言的轻描淡写

理念

- 好的设计的基本目标是
  - 直接用代码表达思想
  - 用代码独立地表达相互独立的思想
  - 用代码直接表达思想之间的关系
  - 自由组合代码表达的思想
    - 当且仅当这种组合有意义时
- 这些都遵从
  - 正确性
  - 可维护性
  - 性能
- 将这些应用到尽可能广泛的程序设计上

理念是有实际用途的

- 在项目开始阶段，检视它们以获得思想
- 当你在深夜徘徊时，回头看看你的代码是否已经偏离了我们的理念－这往往是bug最容易潜藏和设计问题最可能发生之处
  - 不要仅仅“在相同地方反复查看，用相同技术反复寻找错误”
    - “错误总是存在于你没有查看的地方，否则你早就找到了”

理念是个性化的

- 谨慎地选择你认为是好的.....

风格/范型

- 过程式程序设计 Procedural programming
- 数据抽象 Data abstraction
- 面向对象程序设计 Object-oriented programming
- 泛型程序设计 Generic programming

- 函数式程序设计、逻辑程序设计、基于规则的程序设计、基于约束的程序设计、面向方面的程序设计 .....

风格/范型

```
template<class Iter> void draw_all(Iter b, Iter e)
{
    for_each(b, e, mem_fun(&Shape::draw));     // draw all shapes in [b:e)
}
```

```
Point p(100,100);
Shape* a[] = { new Circle(p,50), new Rectangle(p, 250, 250) };
draw_all(a,a+2);
```

- 此处我们使用的风格/范型有哪些？
  - 过程式、数据抽象、OOP 和 GP

组合多种风格，尽可能好地表达解决方案

一些基础要求

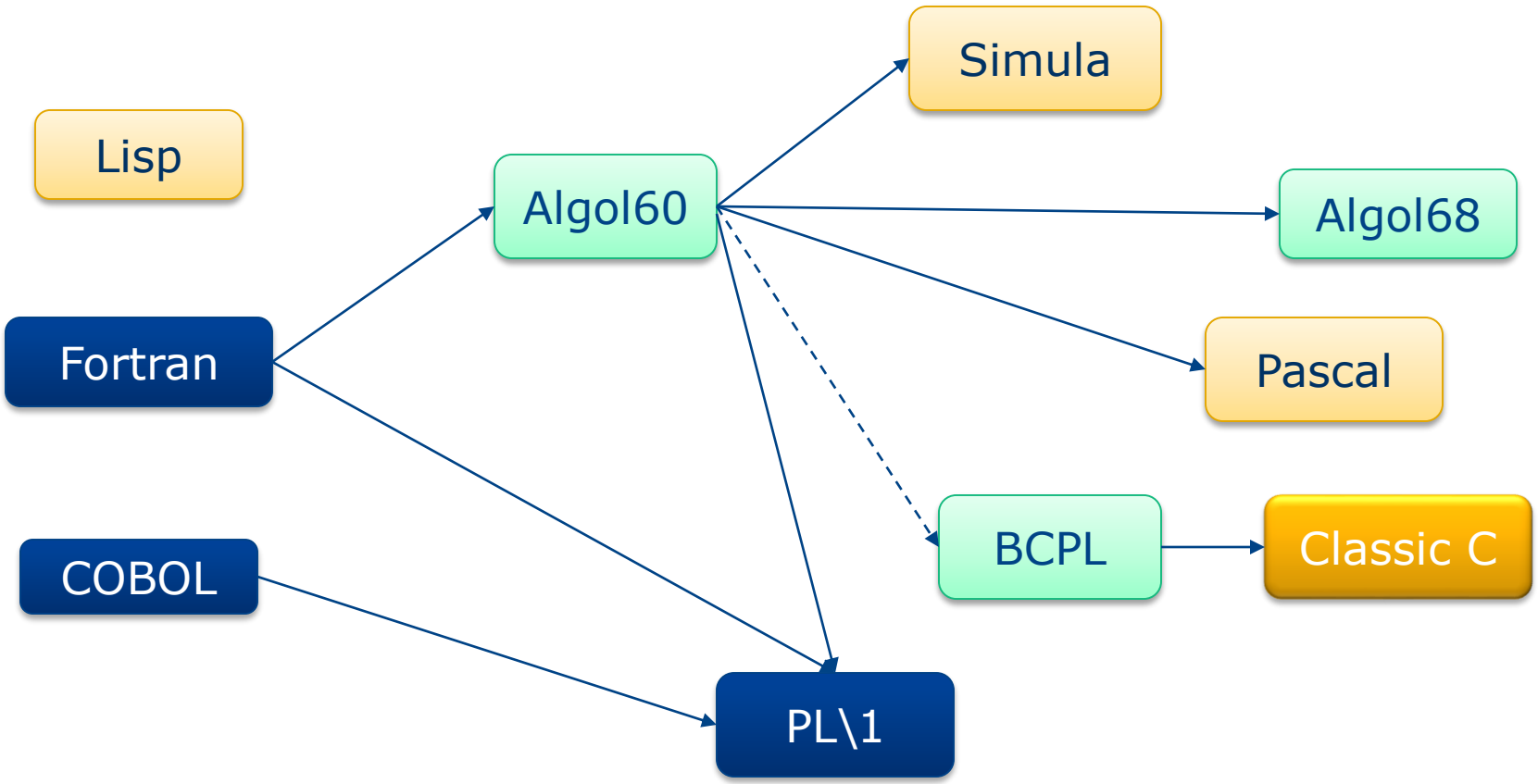
- 可移植是好的
- 类型安全是好的
- 高性能是好的
- 容易调试是好的
- 能访问系统资源是好的
- 长期稳定是好的
- 容易学习是好的
- 轻量级是好的
- 有利于程序分析是好的
- 提供大量工具是好的
- ……

- 你不可能同时拥有所有这些：学会工程折中

编程语言

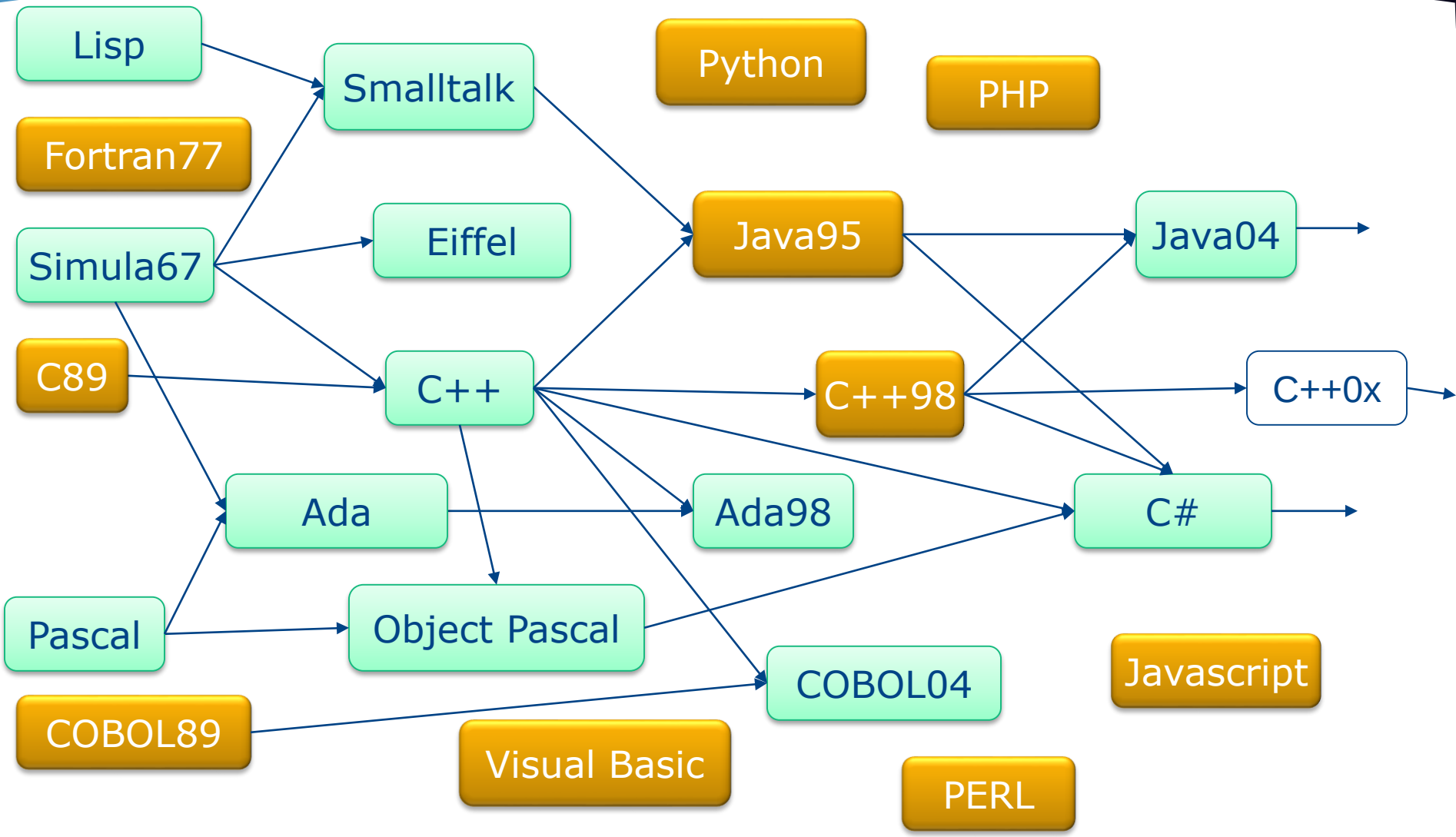
- 机器代码
  - 位、八进制、至多十进制数字
- 汇编
  - 寄存器、装载、存储、整数加、浮点数加……
  - 每一种新的机器都有自己的汇编器
- 高层次语言
  - 先例：Fortran 和 COBOL
- 语言发明的速度
  - 10年至少2000个
- 当今主要的语言
  - 真实精确的统计数据是很难得到的
    - IDS：当今大约9000万个专业程序员
  - COBOL, Fortran, C, C++, Visual Basic, PERL, Java, Javascript
    - Ada, C#, PHP, …

早期编程语言



蓝色：主要是商业使用
浅黄色：产生了重要的“后代”

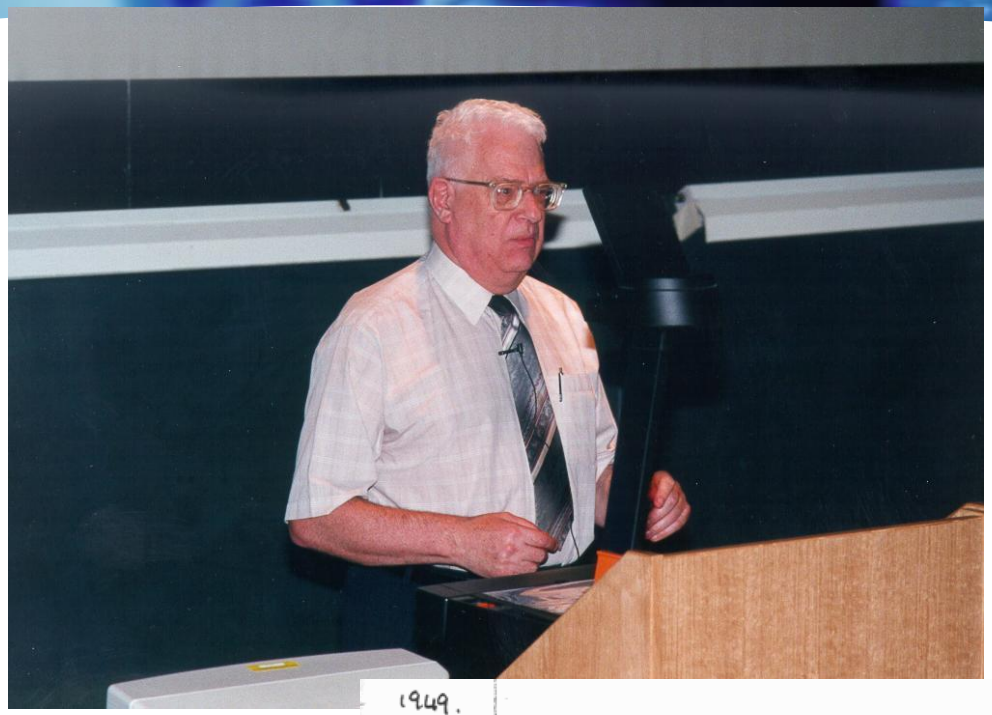
现代编程语言

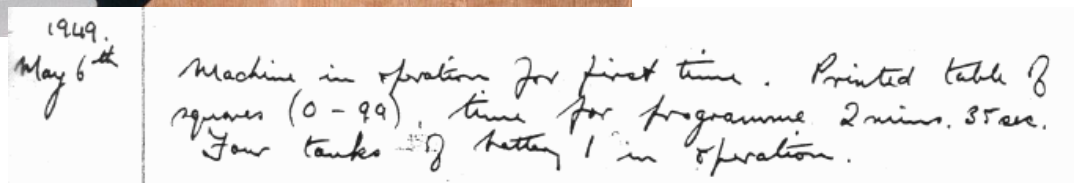

Lisp
Smalltalk
Python
PHP
Fortran77
Eiffel
Java95
Java04
Simula67
C89
C++
C++98
C++0x
Ada
Ada98
C#
Pascal
Object Pascal
COBOL04
Javascript
COBOL89
Visual Basic
PERL

我们为什么要设计和变革语言？

- 存在不同的应用领域需求
  - 没有一门语言能够在所有方面都是工作最好的
- 程序员有多样化的背景和技能
  - 没有一门语言对所有人都是最好的
- 问题的变化
  - 随着时间的变化，计算机被应用到新的领域、解决新的问题
- 计算机的变化
  - 过去几十年，硬件特征发生了很大变化
- 方法的进展
  - 过去若干年，我们已经学会了更好的方法去设计和实现语言

第一个现代计算机 – 第一个编译器

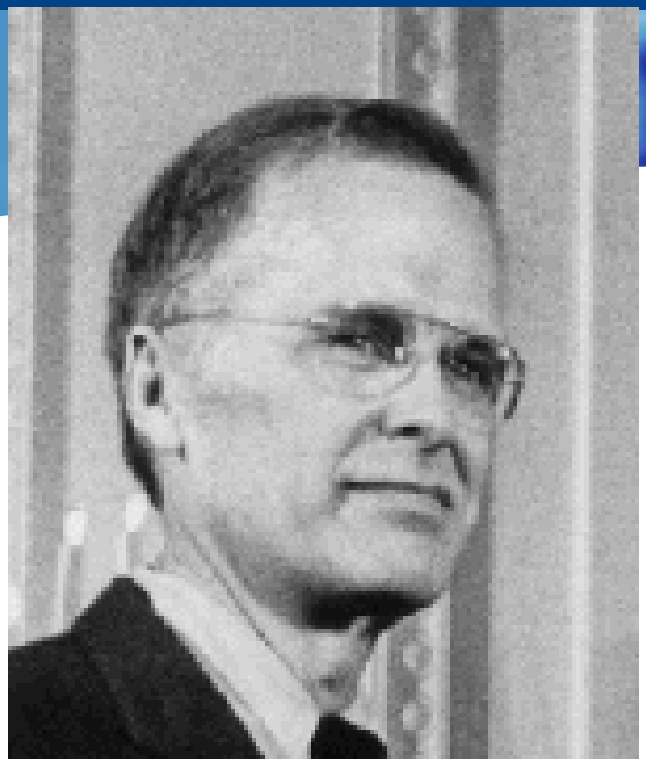




- David Wheeler
  - University of Cambridge
  - 杰出的问题解决专家：硬件、软件、算法、库
  - 是第一个计算机科学专业博士学位获得者 (1951)
  - 合作完成了第一篇描述如何编写正确、可重用及可维护代码的论文 (1951)
  - Bjarne Stroustrup 的博士论文导师 ☺

早期语言– 1952

- 每种机器都有一种语言
  - 处理器特殊的特性
  - 操作系统特殊的特性
  - 大多数有类似于汇编的工具
    - 容易理解指令是如何产生的
  - 代码不可移植



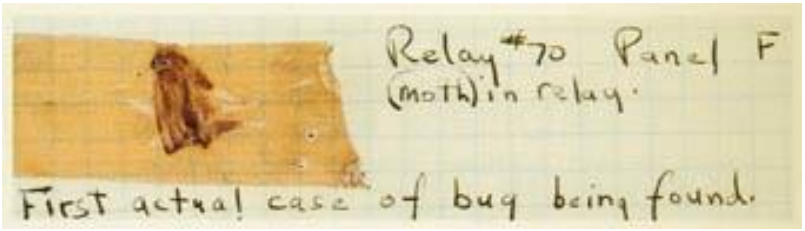
- John Backus
  - IBM
  - FORTRAN，开发的第一个高级计算机语言
    - 我们不知道我们想要什么以及如何去做，它仅仅是一种增长
  - Backus-Naur Form (BNF)，描述高级编程语言文法的一种标准技法
  - 函数式程序设计 (FP)，倡导一种程序设计的数学方法

Fortran – 1956

- 表示“公式转换”—— Formula Translation
- 让程序员写出更多的线性代数运算
  - 数组和循环
  - 标准的数学函数
    - 库
  - 用户自己的函数定义
- 符号大都是机器无关的
  - 只需要很少的改动，Fortran 代码就可以从一台计算机移植到另外一台计算机上
- 这是一个重要的改进
  - 编程语言历史上最大的单个改进(有争辩 ☺)
- 后续演进：II, IV, 77, 90, 0x

COBOL

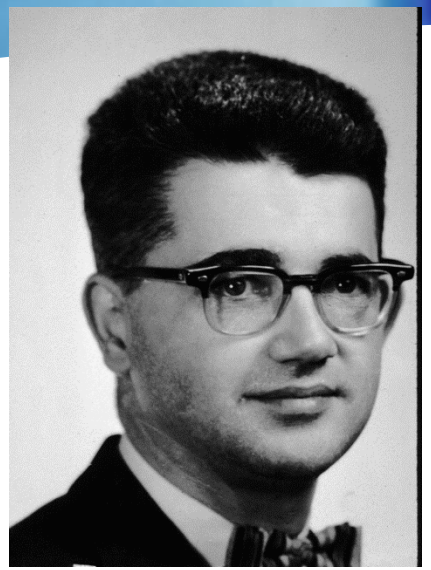
- The Common Business-Oriented Language
- “海军少将 Grace Murray Hopper 博士 (美国海军) 在早期的计算机程序设计领域做出了杰出的贡献。它将软件开发思想的研究作为一生的事业，作为这个领域的领路人，是她引领了从原始的程序设计技术到使用复杂编译器的转变。她坚信’我们原来就是这么做的’不是继续这么做的必然理由”
- 将计算机中的错误称之为“bug”的第一人

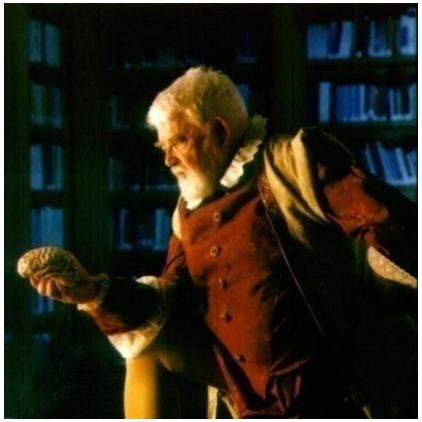


Cobol – 1960

- Cobol 主要是面向商业程序员的语言 (有时仍然是)
  - Fortran 主要是面向科学计算应用程序员的语言 (有时仍然是)
- 其重点是数据操作
  - 拷贝
  - 存储和提取 (如记账)
  - 打印 (如报表)
- 计算被看做是不重要的
- Cobol 被期望与商业英语非常接近，管理者也能够编程，而不再需要程序员
  - 一直没有实现
- 后续演进：60, 61, 65, 68, 70, 80, 90, 04

Lisp



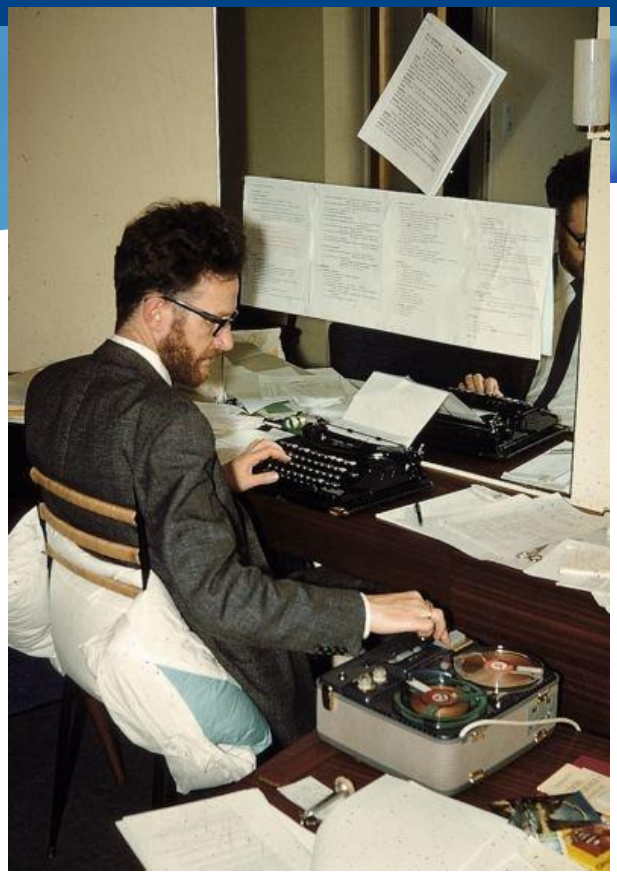


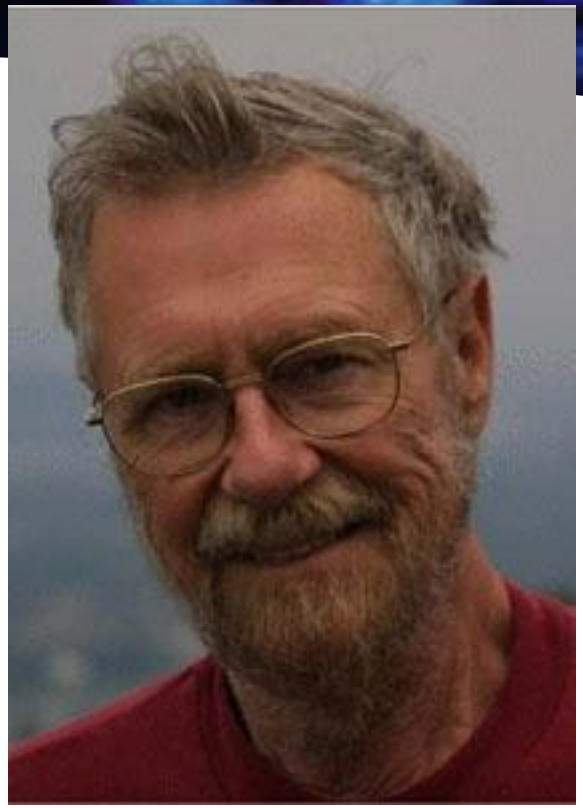
- John McCarthy
  - Stanford University
  - AI (人工智能) 先驱

Lisp – 1960

- 主要用于链表和符号处理 (LISt Processing)
- 最初 (现在通常也是)，Lisp是解释型语言
- Lisp有几十 (可能上百) 种变种
  - “Lisp 默认就是复数(s)”
  - Common Lisp
  - Scheme
- 该语言家族曾经是人工智能领域研究的支柱
  - 尽管发布的产品通常使用 C 或 C++ 实现

Algol

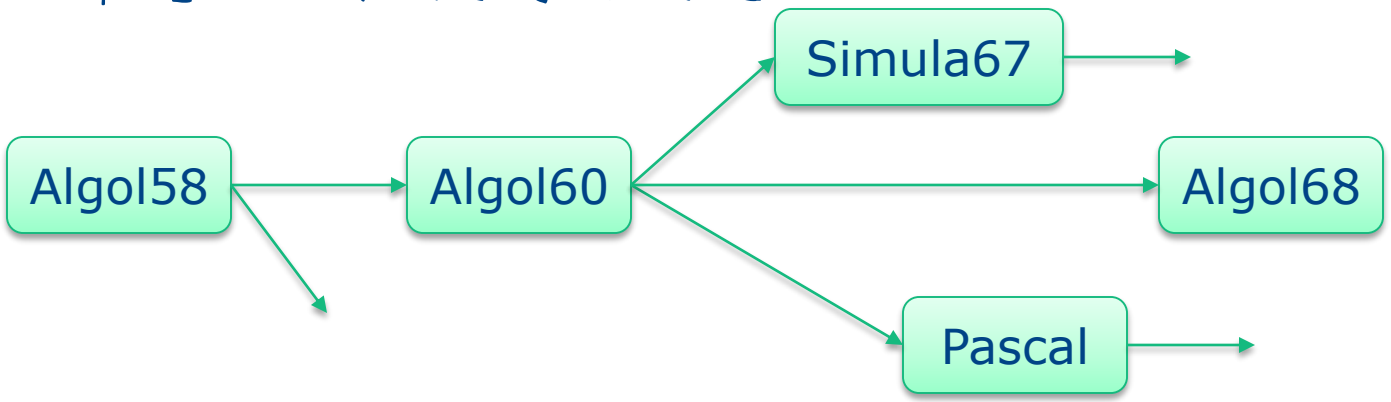




- Peter Naur
  - Danish Technical University 和 Regnecentralen (丹麦)
  - BNF
- Edsger Dijkstra
  - Mathematisch Centrum, Amsterdam, Eindhoven University of Technology, Burroughs Corporation , University of Texas (Austin)
  - 程序设计中的数学逻辑、算法
  - THE 操作系统

Algol – 1960

- 现代程序设计语言概念的突破
  - 语言描述
    - BNF、词汇分割、文法、语义规则
  - 作用域
  - 类型
  - “程序设计语言可用于一般性的目标”
    - 之前，语言要么是科学计算 (e.g., Fortran)，要么商业(e.g., Cobol)，要么字符串操作(e.g., Lisp)，要么仿真，…
- 实际却是，主要用于学术研究



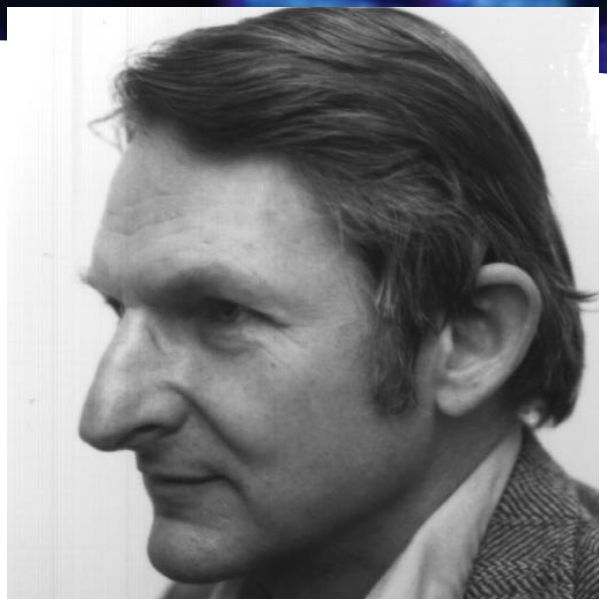
Simula 67

- Kristen Nygaard 和 Ole-Johan Dahl
  - Norwegian Computing Center —— 挪威计算中心
  - Oslo University
  - 面向对象编程和面向对象设计的开端

Simula 1967

- 处理所有的应用领域而不是特定领域
  - 解决 Fortran, COBOL, etc. 已经做的
  - 目标是成为一个通用程序设计语言
- 用代码建模现实世界现象
  - 用类和类对象表示思想
  - 用类层次表示层次关系
- 类、继承、虚函数、面向对象设计
- 程序编程一组相互作用的对象，而不是单个庞然大物
  - 对错误率有很大的改进作用

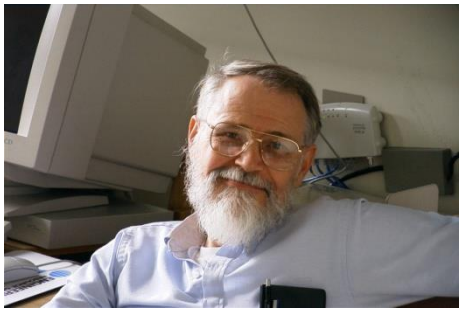
C

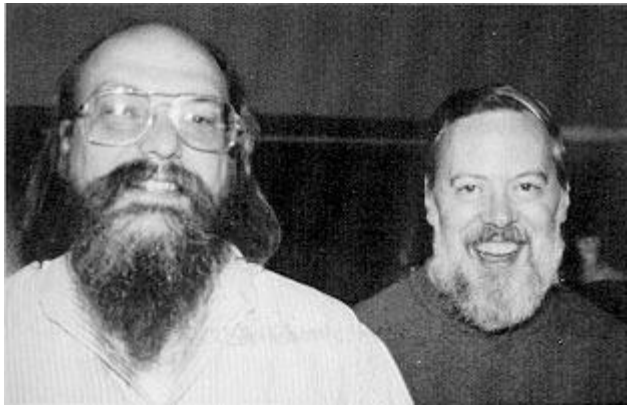


- Dennis Ritchie
  - Bell Labs
  - C and helped with Unix
- Ken Thompson
  - Bell Labs
  - Unix

- Doug McIlroy
  - Bell Labs
  - 一个有想法的人：流行的 C、C++、Unix, 等等
  - 是每个人都喜欢的评论者、讨论对象

Bell Labs – Murray Hill

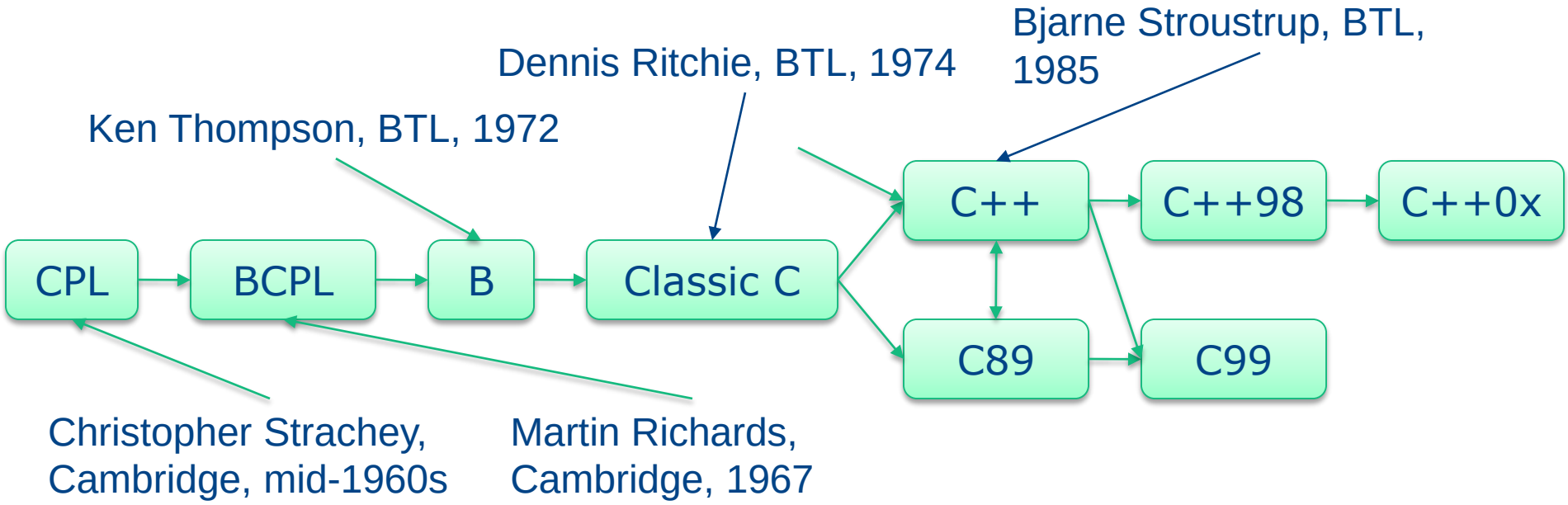




C – 1978

- 相对而言，C是为系统编程的高级程序设计语言
  - 广泛使用，相对底层处理，强类型，系统编程语言
  - 与Unix、Linux、开源运动等息息相关
  - 由 Dennis Ritchie 设计和实现 1974-78

一种强类型、弱检查的语言



C++

- Bjarne Stroustrup
  - AT&T Bell labs
  - Texas A&M University
  - 使抽象技术对于主流项目来说，其应用代价可以承受并易于管理
  - 将面向对象和泛型程序设计技术用于对性能要求较高的应用领域之先驱

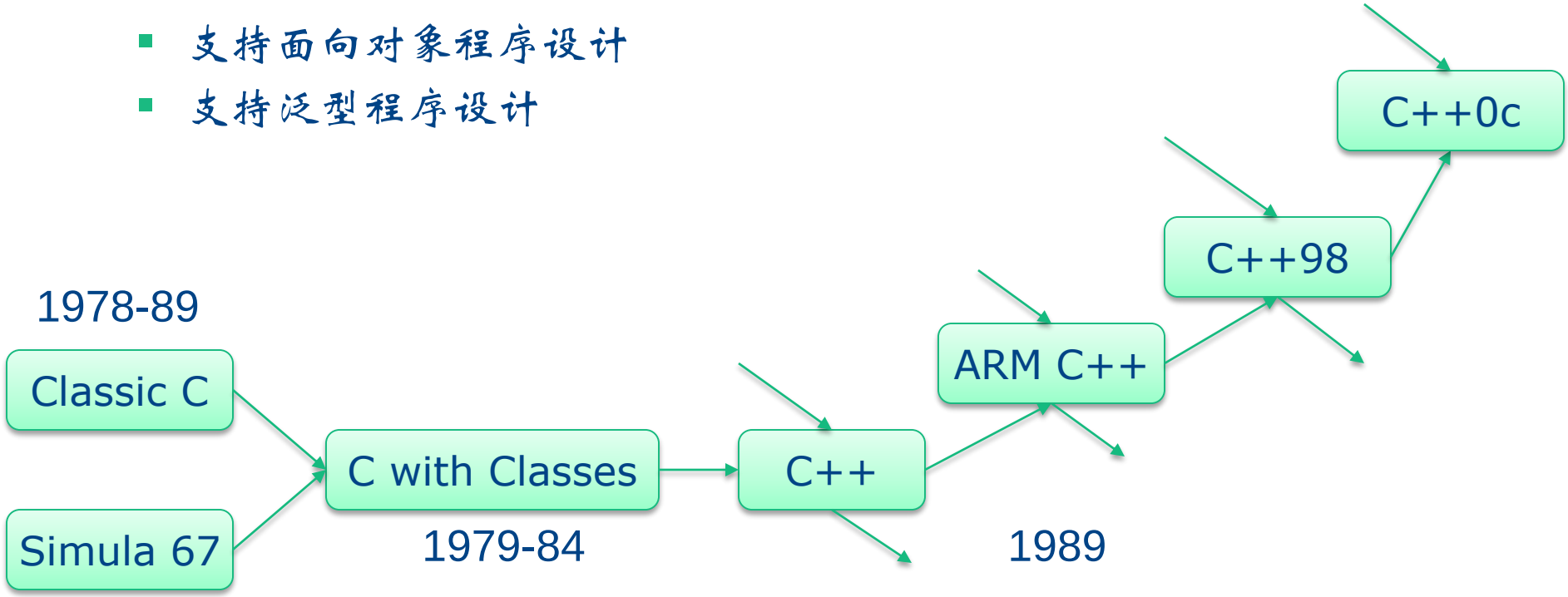
BS 的理念 – in 1980 and still in 2008

- “使严谨的程序员生活更容易”
  - i.e., 首先是自己以及朋友同事
  - 我爱写代码
  - 我喜欢读代码
  - 我恨调试

- 优美和高效的代码
  - 事实上，我并不喜欢在两者之间选择
  - 优美、效率和正确性在很多应用领域是密切相关的
    - 不优美/冗长是bugs和低效的主要原因

C++ – 1985

- C++ 是一种偏向于系统编程的通用程序设计语言，它的特点是：
  - 可以看做是更好的 C
  - 支持数据抽象
  - 支持面向对象程序设计
  - 支持泛型程序设计



更多信息

- 更多语言设计者的网页/图片
  - http://www.angelfire.com/tx4/cus/people/
- 若干语言示例
  - http://dmoz.org/Computers/Programming/Languages/
- 教材
  - Michael L. Scott, *Programming Language Pragmatics*, Morgan Kaufmann, 2000, ISBN 1-55860-442-1
  - Robert W. Sebesta, *Concepts of programming languages*, Addison-Wesley, 2003, ISBN 0-321-19362-8
- 历史书籍
  - Jean Sammet, *Programming Languages: History and Fundamentals*, Prentice-Hall, 1969, ISBN 0-13-729988-5
  - Richard L. Wexelblat, *History of Programming Languages*, Academic Press, 1981, ISBN 0-12-745040-8
  - T. J. Bergin and R. G. Gibson, *History of Programming Languages – II*, Addison-Wesley, 1996, ISBN 0-201-89502-1