

Lecture 1

编译原理课程介绍

徐 辉

xuh@fudan.edu.cn



大纲

1. 课程简介
2. 编译流程概览
3. 编译器类型
4. 编程语言范式和简史

大纲

1. 课程简介
2. 编译流程概览
3. 编译器类型
4. 编程语言范式和简史

为什么学习编译原理？

- 编译器是程序员和计算机沟通的桥梁；
- 通过便于理解的高级语言提升软件开发效率。



源代码

```
int main(){  
    printf("hello,  
        compiler!\n");  
    return 0;  
}
```

汇编

```
push    rax  
mov     edi, offset s  
call    _puts  
xor     eax, eax  
pop     rcx  
retn
```

机器码

```
50 BF 04 20  
40 00 E8 F5  
FE FF FF 31  
C0 59 C3 90
```

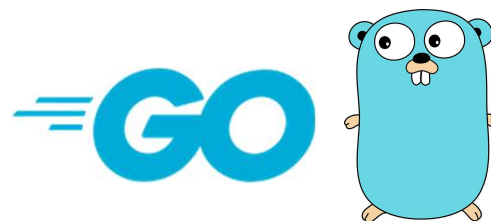


问：第一门高级编程语言是？

新型编程语言层出不穷



Mozilla (浏览器引擎)
Graydon Hoare
2006-2014 (v1)



Google (多核、分布式服务)
R. Griesemer, R. Pike, K. Thompson
2007-2012 (v1)

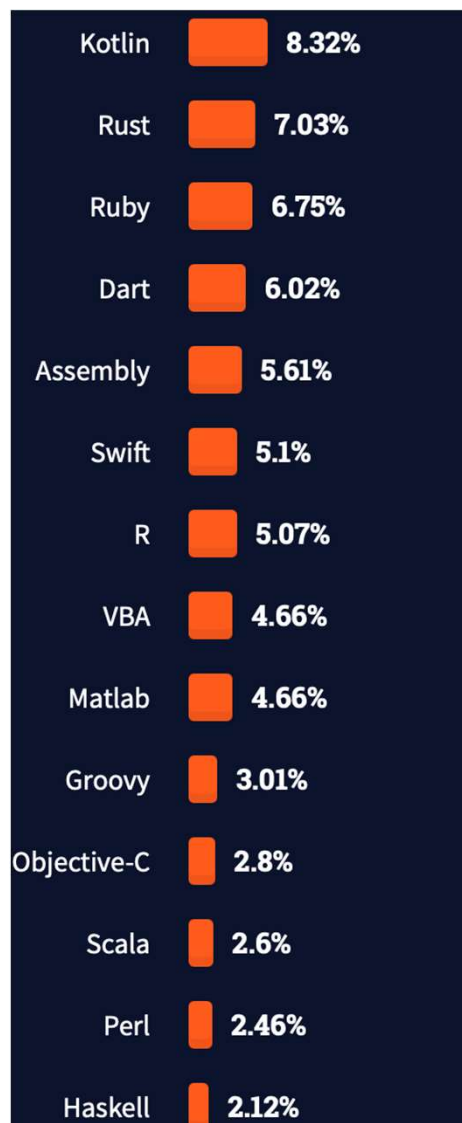
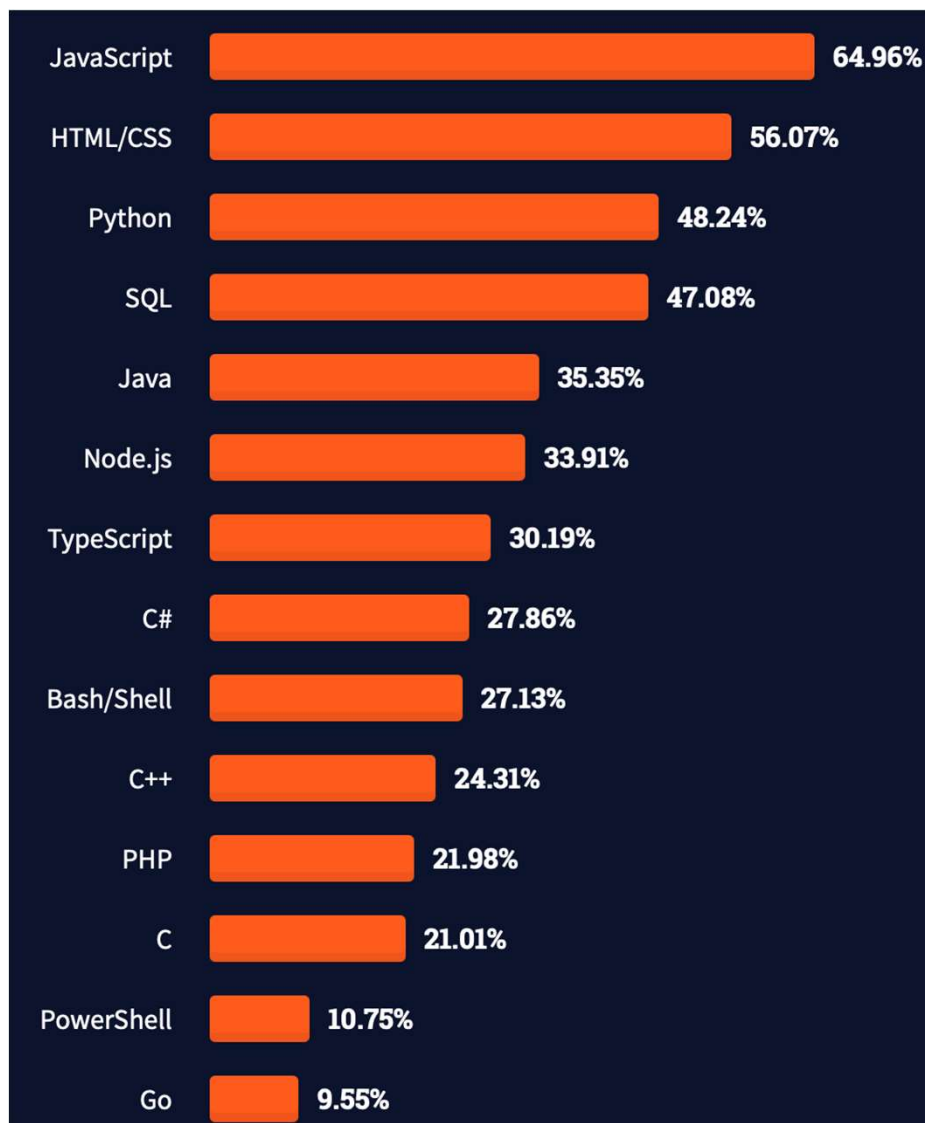


Apple (应用程序)
Chris Lattner
2010-2014 (v1)



Julia Computing (数值计算)
J. Bezanson, S. Karpinski, V. Shah,
A. Edelman
2009-2012 (v1)

Stackoverflow语言使用统计排名



编译器和编程语言的重要性

- 计算机领域最高荣誉：
 - ACM Turing Award,
 - IEEE John von Neumann Medal

图灵奖得主



Liskov (subtype)
substitution
principle

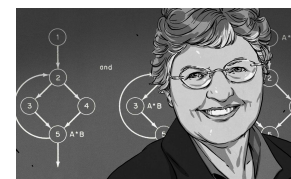
2020年
Aho & Ullman

2008年
Barbara Liskov

面向对象
(Smalltalk语言)

BNF范式/ALGOL 60

Optimizing
Compilers

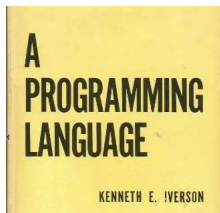


2003年
Alan Kay

2005年
Peter Naur

2006年
Frances Allen

$\{\emptyset\}P\{\psi\}$
Hoare Logic



FORTRAN
(语言)


ALGOL 60
(语言)

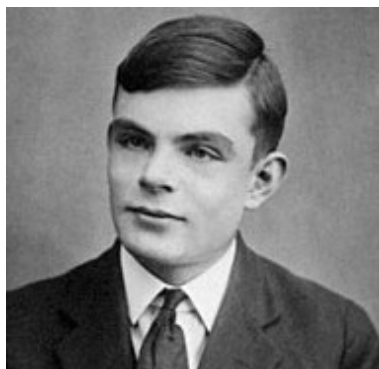
1980年
Tony Hoare

1979年
Kenneth Iverson

1977年
John Backus

1972年
Edsger Dijkstra

图灵



Alan M. Turing (1912年-1954年)

计算机科学与人工智能之父

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO
THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]



1931年，进入剑桥大学国王学院（数学一等荣誉）

1935年，国王学院院士（证明中心极限定理）

1936年，发表《论可计算数及其在判定问题上的应用》

1936-1938年：普林斯顿大学（数学系博士），被冯·诺依曼邀请做博士后但谢绝

1939年，与维特根斯坦辩论（把数学抬得太高反而不能发现任何绝对真理）

1939年，军情六处（布莱切利庄园）、两年后破解恩尼格玛密码机

1945-1948年，国家物理实验室，负责自动计算引擎研究

1947年，英国奥运代表队进行了试训，马拉松2小时46分

1949年，曼彻斯特大学，计算机实验室副主任

1950年，提出图灵测试实验

1952年，从事生物数学研究，关注植物结构的斐波那契数，论文《形态发生的化学基础》

冯诺伊曼



John von Neumann (1903年—1957年)

理论计算机科学与博弈论的奠基者

1913年，他的父亲马克斯·诺伊曼被授予世袭贵族头衔

1924年，苏黎世联邦理工学院(ETH)，化学工程师

1926年，布达佩斯大学数学博士学位

1930年，普林斯顿大学客座教授的职位

1945年，First Draft of a Report on the EDVAC（101页报告），美国陆军阿伯丁试验场的弹道研究实验室顾问

1951年，冯诺依曼结构的二进制计算机EDVAC宣告完成。

文本解析问题

- 问题1：如何描述一个金额字符串？如 “¥1000”、 “\$1,000”。
 - 以币种符号（\$、¥、£、或€）开头
 - 币种符号后紧跟非零数字；
 - 含有若干个数字
 - 如含有逗号，则每三个数字前含有一个逗号；
 - 以数字结尾

正则表达式： $(\$|\yen|\pounds|\text{€})([1-9]|([1-9][0-9]|([1-9][0-9])^2)(\text{' , '},[0-9]^3))^*$

- 问题2：判断一个句子是不是存在语病？
 - 主语+谓语+宾语
 - ...
- 问题3：分析一个XML文档是否存在结构错误？
 - 每个<any>开始标签对应一个结束标签</any>
 - 标签嵌套正确：<any> <some> XXX </some> </any>

问：自然语言可以编程吗？




抽象级别

自然语言
(外祖母编程语言)

高级编程语言

汇编语言

机器码

 Explore Problems Interview New Contest Di

Description Solution Discuss (9... Submission...

1143. Longest Common Subsequence

Medium 7998 91 Add to List Share

Given two strings `text1` and `text2`, return *the length of their longest **common subsequence***. If there is no **common subsequence**, return `0`.

A **subsequence** of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

- For example, `"ace"` is a subsequence of `"abcde"`.

A **common subsequence** of two strings is a subsequence that is common to both strings.

Example 1:

Input: `text1 = "abcde", text2 = "ace"`

Output: `3`

Explanation: The longest common subsequence is `"ace"` and its length is 3.

教学大纲和目标

- 第一阶段：实现一个计算器（前端基础知识）
 - 词法分析
 - 语法分析
 - 工具使用：Flex + Bison
- 第二阶段：实现一个编译器（核心部分）
 - 设计编译器语法
 - 中间代码
 - 类型检查
 - 后端实现
 - 工具案例：LLVM
- 第三阶段：高级特性（视课程进度决定）
 - 静态分析和代码优化
 - 自动内存管理
 - 高级类型系统
 - 异常处理
 - 函数式编程

知乎上的讨论

如何学习编译原理？

<https://www.zhihu.com/question/21515496>



知乎用户

编程话题下的优秀答主

434 人赞同了该回答

如何学习编译原理？个人不太建议一上手就拿起龙书、虎书等等来看。

学过编译原理课程的同学应该有体会，各种文法、各种词法语法分析算法，非常消磨人的耐心和兴致；中间代码^o生成和优化，其实在很多应用场景下并不重要（当然这一块对于“编译原理”很重要）；语义分析要处理很多很多细节，特别对于比较复杂的语言；最后的指令生成，可能需要读各种手册，也比较枯燥。



CompilerCoder

GPU编译器工程师

138 人赞同了该回答

大学的时候学过一门编译原理的课程，当时老师讲课主要讲的是词法分析、语法分析等，对于后端基本没讲。当时讲各种文法的时候一上来就是各种符号，各种概念非常绕，最后为了考试只能硬学。



ddss

79 人赞同了该回答

這是個好問題，我光是發現怎麼學習編譯原理^o就花了不少時間，也買了不少書，但每本書的實作都不同，讓學習更難了。

最後我想到一個方法：

我要實作 c 語言編譯器，畢竟書上寫的 pascal 實作我一點都不感興趣，我又沒在用 pascal，我在使用的是 c/c++ 語言，實作一個自己沒在用的語言實在是沒有動力。

课程信息

- 课程内容包括课堂教学+上机实践两部分
- 课堂教学：
 - 时间：星期五 6-8节（1:30pm-4:10pm）[1-16周]
 - 地点：第三教学楼205（H3205）
- 上机实践：
 - 时间：[每双周] 1-2节（8:00am-9:40am）
 - 地点：H逸夫楼202
- 课程平台：
 - Elearning
 - WeChat
 - 课程主页：<https://hxuhack.github.io/lecture/compiler>

教学团队

- 授课教师：徐辉
 - Ph.D, CUHK
 - 研究方向：程序分析、软件可靠性
 - 办公室：江湾校区交叉二号楼D6023
 - Email: xuh@fudan.edu.cn
 - 主页: <https://hxuhack.github.io/>



- 助教：



陈澄钧

Email: cjchen20@fudan.edu.cn
Office: 江湾校区交叉二号楼D6010



张业鸿

Email: 21210240006@m.fudan.edu.cn
Office: 江湾校区交叉二号楼D4004

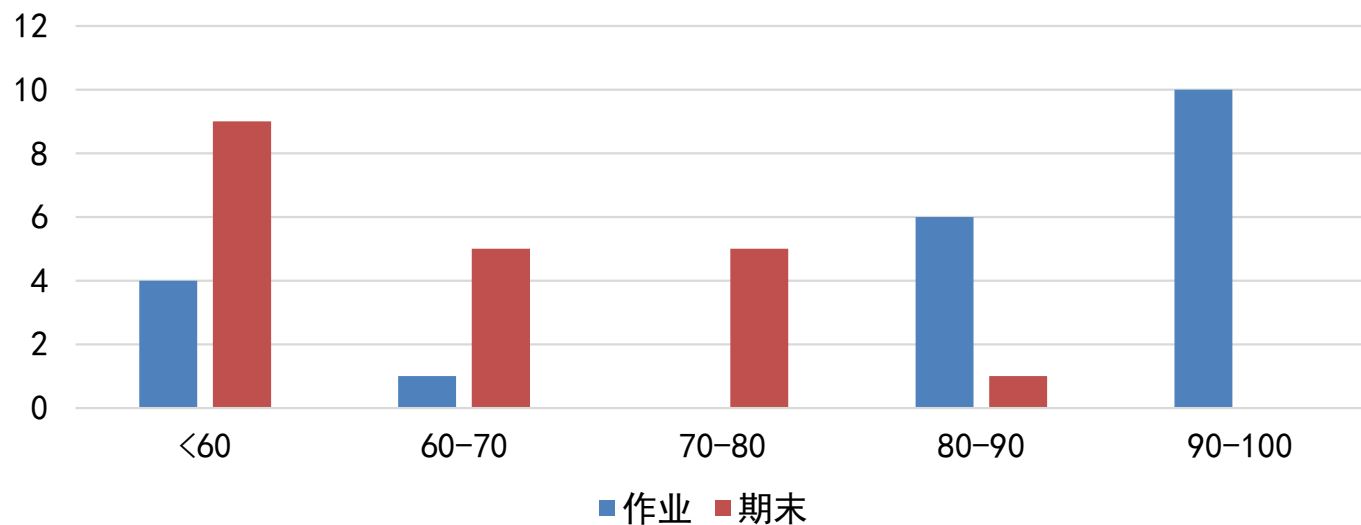
课程考核

- 课程作业：50%
 - 5次上机实验
 - 1次实验汇报
 - 第16周当堂汇报
 - 每人10+2分钟
- 闭卷考试：50%

上机课程		作业
第二周	Flex + Bison	1. 计算器程序
第四周	LLVM Lexer	2. 编写前端
第六周	LLVM Parser	
第八周	LLVM IR Gen	3. 编写中间代码
第十周	LLVM IR More	
第十二周	LLVM Table Gen	4. LLVM后端
第十四周	LLVM Table Gen 2	5. 功能扩展
第十六周	课程报告	

去年的成绩情况

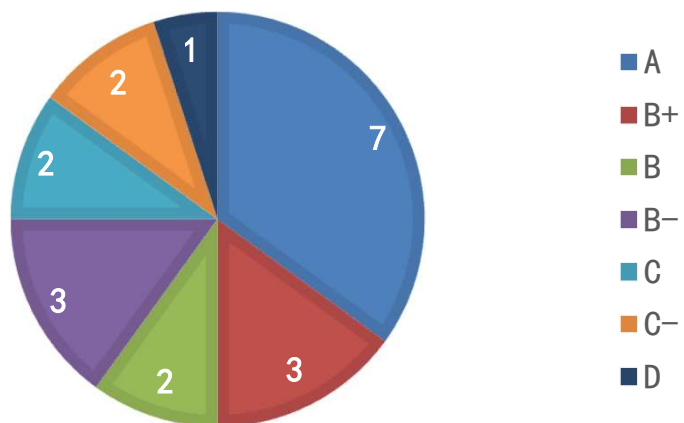
2021年学生成绩分布



作业=[36, 100]

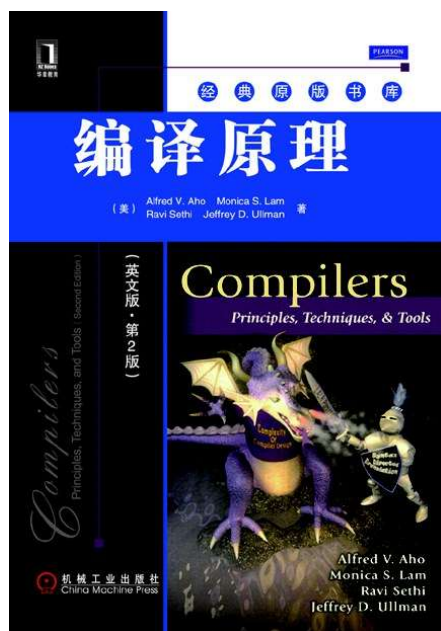
期末=[37, 81]

最终成绩



主要参考书

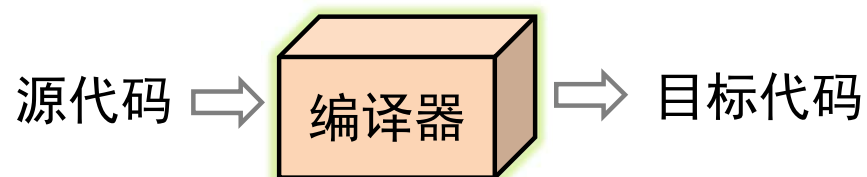
- 自编讲义
- 其他参考资料
 - 《编译原理（第2版）》 Alfred V. Aho 等著
 - 主要参考词法分析、语法分析等算法
 - 该书编写较早，对中间代码和后端的介绍较弱
 - 《编译器设计（第2版）》 Keith Cooper等著
 - 工程系统性较强
 - 电子版：<https://dl.acm.org/doi/pdf/10.5555/2737838>



大纲

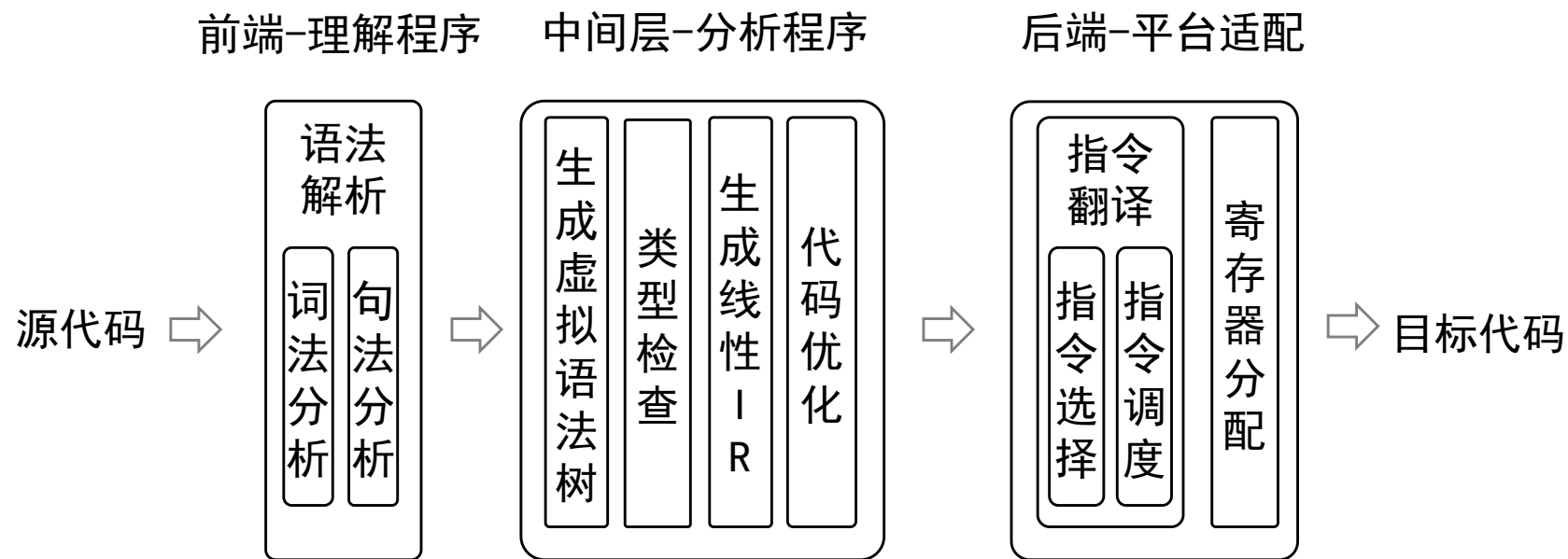
1. 课程简介
2. 编译流程概览
3. 编译器类型
4. 编程语言范式和简史

编译器的概念



- 将计算机程序从一种语言转换为另一种语言的程序
 - 一般从源代码转化为机器码或中间代码
 - 源代码：C/C++、Rust、Java、javascript等
 - 机器码：某种处理器的指令集，如x86或ARM
 - 基本要求：保持语义等价

编译器基本框架



词法分析: Lexical Analysis

- 编译器的一趟 (pass), 将字符串转换为单词流 (Tokenize)。
 - 语法规则一般是基于词类/词性定义的。

句子 → 主语 动词 宾语 结束符

推导出

例句: Compilers are engineered objects.
 名词 动词 形容词 名词 终结符

句法分析: Parsing

- 编译器的一趟 (pass), 分析单词流是否为该语言的一个句子。

语法规则

| 表示或

- [1] 句子 → 主语 动词 宾语 结束符
- [2] 主语 → 名词 | 修饰词 名词
- [3] 宾语 → 名词 | 修饰词 名词
- [4] 修饰词 → 形容词 | 物主代词

句法解析

Compilers are engineered objects.

名词 动词 形容词 名词 终结符

句子

- [1] ⇒ 主语 动词 宾语 结束符
- [2] ⇒ 名词 动词 宾语 结束符
- [3] ⇒ 名词 动词 修饰词 名词 结束符
- [4] ⇒ 名词 动词 形容词 名词 结束符

生成中间代码（语法制导）

- 进行上下文相关分析
 - 语法分析不考虑上下文
 - 语法正确不一定整句有意义，如类型错误
- 生成虚拟语法树（AST）
- 生成线性IR（LLVM IR）

示例：源代码->中间代码

源代码： (1 + 2) * -3



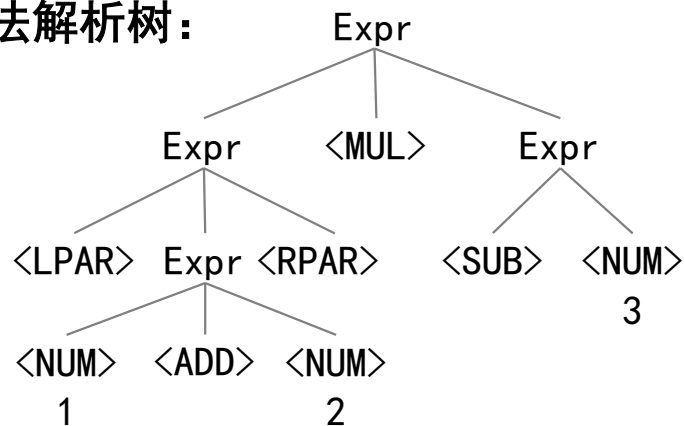
1. 词法分析（标签化）

<LPAR> <NUM(1)> <ADD> <NUM(2)> <RPAR> <MUL> <SUB> <NUM(3)>



2. 句法分析

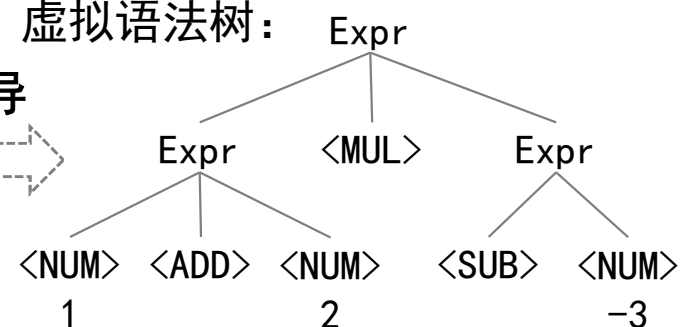
语法解析树：



3. 语法制导

虚拟语法树：

3. 语法制导



线性IR：

```
t0 = 1 + 2;
t1 = -3;
t2 = t0 * t1;
```

优化: Optimization

- 数据流分析
 - 常量传导
 - 循环优化
 - 尾递归
 - ...

```
for (i=1; i<100; i++){  
    int d = getInt();  
    a = 2 * a * b * c * d;  
}
```

优化
⇒

```
int t = 2 * b * c;  
for (i=1; i<100; i++){  
    int d = getInt();  
    a = a * t * d  
}
```

指令选择: Instruction Selection

- 将中间代码翻译为目标机器指令集。
 - 一般假设寄存器的数目是不受限的。

IR

```
define dso_local i32 @ident(i32 %0) #0 {  
    %2 = load i32, i32* @global_var, align 4  
    %3 = add nsw i32 %0, %2  
    %4 = mul nsw i32 %3, 2  
    ret i32 %4  
}
```



汇编代码

```
pushq    %rbp  
movq     %rsp, %rbp  
movl     %edi, -4(%rbp)  
movl     -4(%rbp), %eax  
addl     global_var, %eax  
shll     $1, %eax  
movl     %eax, -8(%rbp)  
movl     -8(%rbp), %eax  
popq     %rbp  
retq
```

指令调度：Instruction Reordering

- 根据计算性能瓶颈优化指令顺序。
- 假设特定的计算机指令会消耗固定的时钟周期：
 - 1: ADD
 - 2: MOV
 - 3: MUL
 - 7: DIV
- 假设后一条指令的操作符可用时会进入下一条指令，无需等待。

开始	结束	指令
1	2	MOV \$-12(%rsp), r1
2	3	MOV \$-16(%rsp), r2
4	4	ADD r2, r1
5	6	MOV \$-20(%rsp), r2
6	7	MOV \$-24(%rsp), %eax
8	14	DIV r2,
15	16	MOV \$-28(%rsp), r2
17	19	MUL r1, r2
18	19	MOV %eax, \$-24(%rsp)
20	21	MOV r2, \$-28(%rsp)

优化
→

开始	结束	指令
1	2	MOV \$-20(%rsp), r3
2	3	MOV \$-24(%rsp), %eax
4	10	DIV r3
5	6	MOV \$-12(%rsp), r1
6	7	MOV \$-16(%rsp), r2
8	8	ADD r2, r1
9	10	MOV \$-28(%rsp), r4
11	13	MUL r1, r4
12	13	MOV %eax, \$-24(%rsp)
13	14	MOV r4, \$-28(%rsp)

寄存器分配: Register allocation

- 如何使用数量最少的寄存器?
 - 指令选择假设寄存器有无限多，而实际寄存器数目有限；
 - 如果超出了寄存器数量需要将数据临时保存到内存中；
 - 通过寄存器分配降低数据存取开销。

```
x1 = 0
x2 = 1
x3 = x1 + x2
x4 = x2 + x3
x5 = x3 + x4
%eax = x5
ret
```



```
MOV $0, %eax
MOV $1, %edx
ADD %edx, %eax
ADD %eax, %edx
ADD %edx, %eax
ret
```

大纲

1. 课程简介
2. 编译流程概览
3. 编译器类型
4. 编程语言范式和简史

编译执行和解释执行

- 编译执行
 - 通过编译器将源代码翻译为可以直接运行的机器码；
 - 如C/C++、 Rust等语言。
- 解释执行
 - 在目标机器上将源代码翻译成计算机指令并执行；
 - 如javascript、 python等各种脚本语言；
 - 平台无关，但存在一定的运行时解释开销。
- 混合模式
 - 如Java等语言通过编译器将源代码翻译为中间代码；
 - 平台无关，省略代码解析开销。

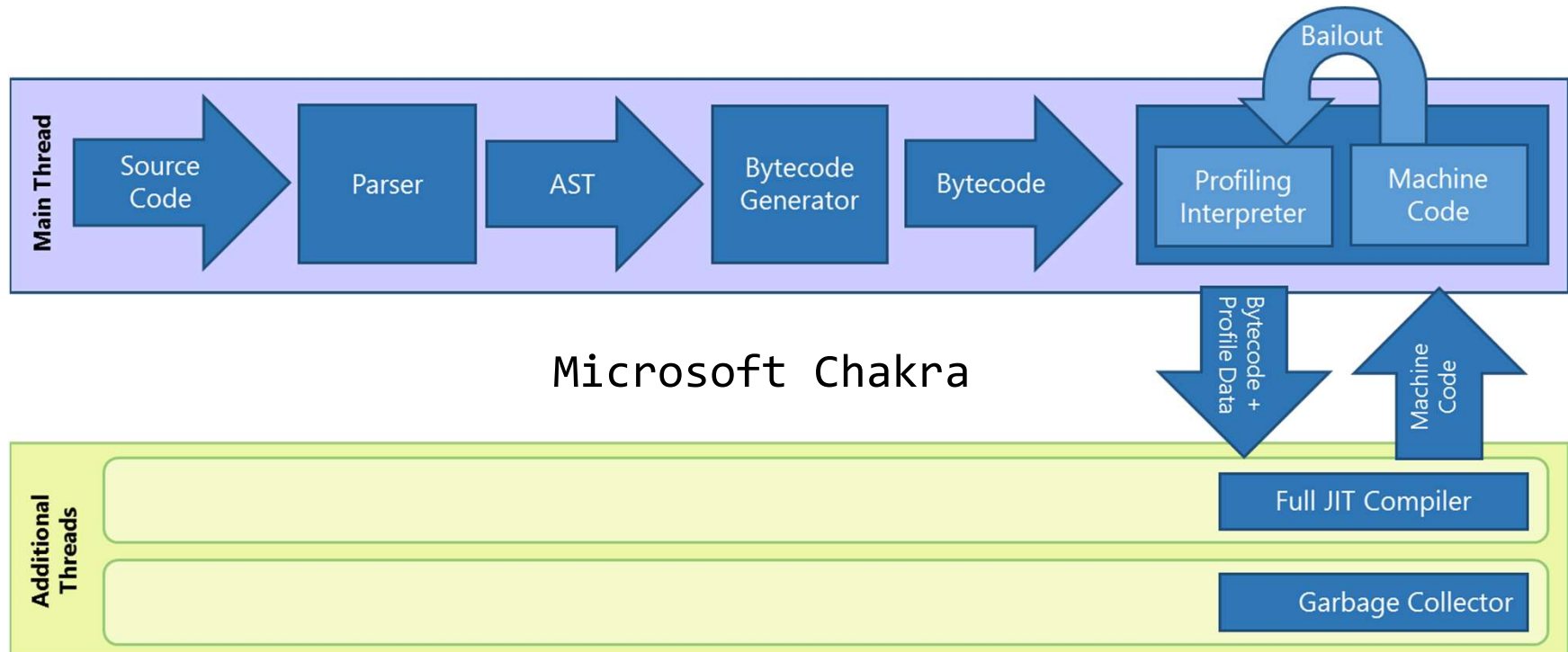
运行时编译just-in-time (JIT) compilation

- JIT将一部分需要频繁运行的代码直接编译为机器码。
 - 降低解释执行的运行时开销
- 在javascript、JVM等解释执行引擎中广泛使用。
- 将全部代码提前编译为机器码的技术称为Ahead-of-Time (AOT) compilation。
 - 传统C/C++等编译执行语言
 - 将Java代码直接编译为机器码
 - 安卓智能手机

实例：JavaScript引擎

- 如微软的Chakra和Google 的V8。
- 通过Profiler监控代码运行，分析可通过JIT优化的代码。
 - 如循环内部的代码；
 - 优化是有风险的，尤其js是动态类型，可能存在类型问题；
 - 通过Bailout回退到解释执行。

```
for (var i = 0; i < arr.length; i++) {  
    sum += arr[i];  
}
```



静态类型和动态类型

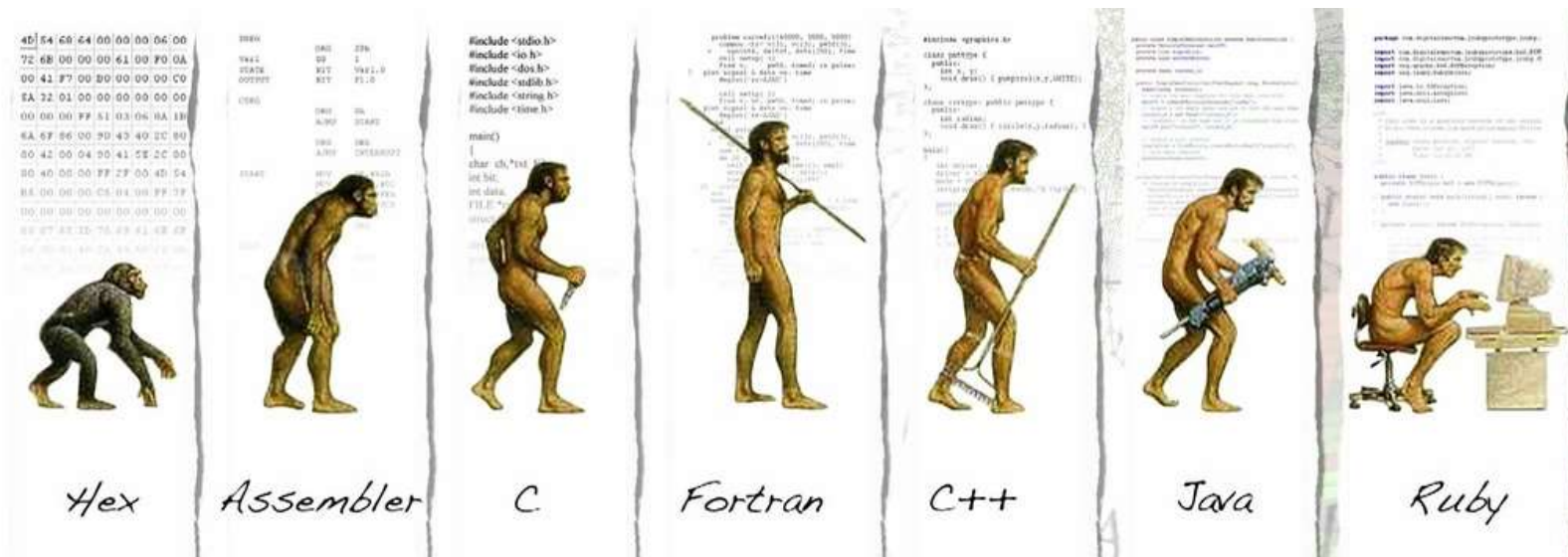
- 静态类型 (static typing)
 - 编译时确定数据类型；
 - 通过类型检查避免运行时错误；
 - 如C/C++、Java等语言；
 - 一般写程序时需要为变量指定具体的类型，但不绝对。
- 动态类型
 - 运行时确定数据类型；
 - 如Python等各种语言；
 - 一般写程序时不用为变量指定具体的类型。

运行时环境和垃圾回收

- 运行时环境（RTE）：软件在系统中运行所需要的环境，一般指动态库依赖等。
 - 如C语言的libc，Java的JRE等；
 - 与具体的操作系统交互，完成内存管理等功能；
 - 有些语言如C、Rust可以不使用运行时（no_std）。
 - 在裸机环境运行
- 垃圾回收：自动内存（堆）管理机制
 - 用户程序和垃圾回收器管理共同的内存空间；
 - 如Java、Go、Python、Ruby等语言；
 - 垃圾回收依赖虚拟机吗？不是。

大纲

1. 课程简介
2. 编译流程概览
3. 编译器类型
4. 编程语言范式和简史

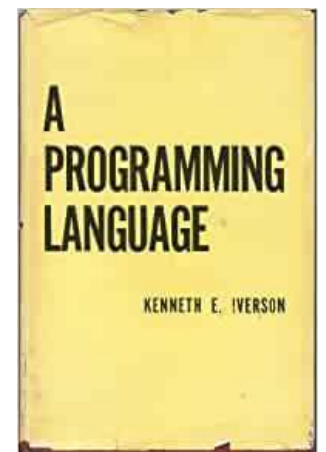


什么是编程语言

《A Programming Language》 - 1962年

Kenneth Iverson

*“Applied mathematics is largely concerned with the design and analysis of **explicit procedures for calculating** the exact or approximate values of various functions. Such explicit procedures are called algorithms or programs. Because an **effective notation for the description** of programs exhibits considerable syntactic structure, it is called a programming language.”*



主要编程范式Programming Paradigm

- 命令式 (Imperative)
- 函数式 (Functional)
- 其它范式:
 - 声明式 (Declarative)
 - 面向对象 (Object-oriented)

主要编程范式的诞生：可计算性问题



大卫·希尔伯特
(数学的完备性问题)

1900年

- 完备性：所有数学命题都可以用一组有限的公理证明或证否；
- 一致性：可以证明的都是真命题。



库尔特·哥德尔
(不完备定理)

1931年

- 反例：“这个命题是不可证的”
- 如果完备性成立，则一致性不成立；
- 如果一致性成立，则完备性不成立。

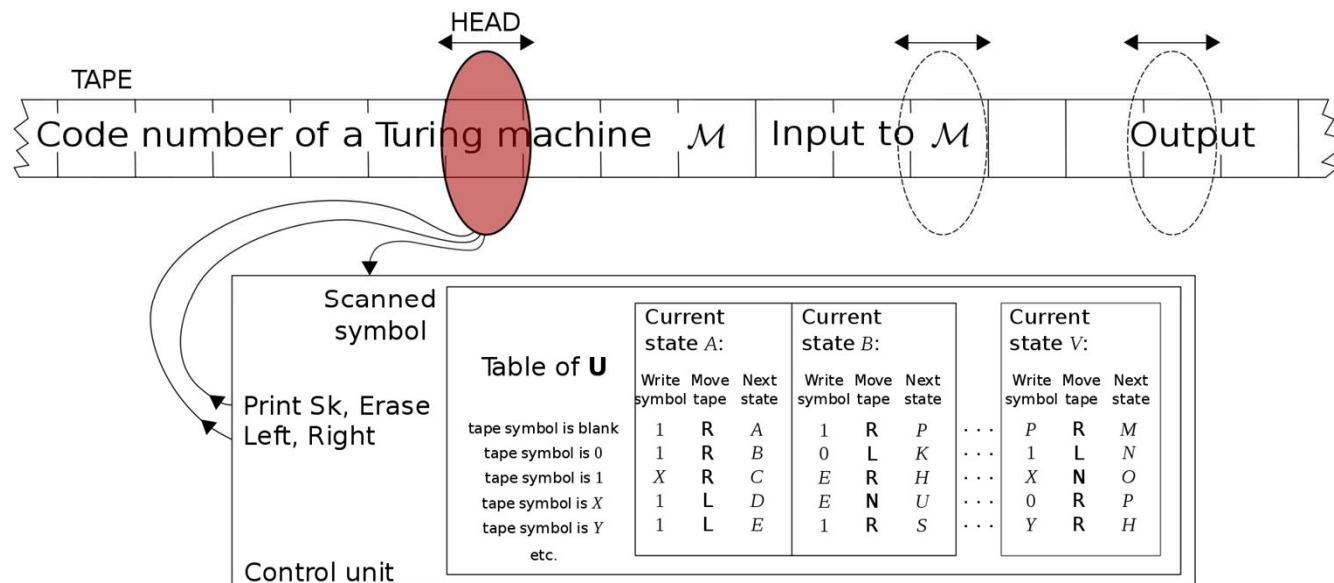


邱奇和图灵
(可计算性)

1936年

- 可判定性：哪些问题是在有限时间内可以计算的？
- 邱奇：可计算函数和Lambda演绎
- 可计算函数可以等价于图灵机能计算的函数
 - 《论可计算数及其在判定问题中的应用》

图灵机启发命令式编程



- 通用图灵机：
 - 一条无限长的纸带TAPE
 - 一个读写头HEAD
 - 一个状态寄存器
 - 一套控制规则TABLE：根据当前的寄存器值和读写头所指的格子上的符号确定下一步的动作
- 冯诺依曼结构或哈弗结构

命令式编程

- 命令式编程语言是接近计算机硬件实现的语言设计
 - 很容易翻译成计算机指令
 - 执行顺序很关键
 - 代表语言：Fortran、Basic、C
 - 主要组成：
 - 赋值语句：内存存取、运算、以及更复杂的运算赋值形式
 - 循环语句：for、while等
 - 条件/非条件跳转语句

```
int fact(int n) {  
    int sofar = 1;  
    while (n>0) sofar *= n--;  
    return sofar;  
}
```

Lambda演绎启发函数式编程

组成	形式	举例	
变量	x		
抽象	$(\lambda x.M)$	$f(x) = x + 2 \Rightarrow \lambda x.x + 2$	函数定义
应用	(M, N)	$f(1) \Rightarrow (\lambda x.x + 2) 1$	函数应用

可以接受函数作为函数的输入参数或返回值

$(\lambda f.f\ 1)(\lambda x.x+2)$

应用 β reduction $\Rightarrow (\lambda x.x + 2) 1$

应用 β reduction $\Rightarrow 1 + 2$

函数式编程

- 将电脑运算视为函数运算，避免或减少副作用。
 - 函数是一类“公民”，可以作为参数传递
 - 副作用：除了返回值以外对主调用函数产生的影响。
 - 如修改全局变量或参数。
 - 主要特点：单赋值、多用递归实现
 - 代表语言：Haskell、ML、Lisp等
 - 基于 λ 演算 (λ -calculus) 的思想

```
(defun factorial (n)
  (if (= n 0) 1
      (* n (factorial (- n 1))) ) )

(loop for i from 0 to 16
  do (format t "~D! = ~D~%" i (factorial i)) )
```

Lisp代码

声明式编程

- 描述目标的性质，而非流程，避免或减少副作用。
- 包括以下子类：
 - 函数式编程：如Haskell、ML、Lisp
 - 领域专属语言：如SQL、XML
 - 逻辑式编程：如Prolog
 - 约束式编程：如SMT-Lib

```
fact(X,1) :-  
    X == 1.  
fact(X,Fact) :-  
    X > 1,  
    NewX is X - 1,  
    fact(NewX,NF),  
    Fact is X * NF.
```

Prolog代码

```
(set-logic QF_LIA)  
(declare-const result Int)  
(define-fun-rec  
  fac ((x Int)) Int (  
    ite (<= x 1) 1  
    (* x (fac (- x 1)))  
  )  
)  
(assert (= (fac 4) result))  
(check-sat)  
(get-value(result))
```

SMT-Lib代码

面向对象编程

- 将对象（类的实例）视作程序的基本单元。
 - 将数据和方法封装在对象中；
 - 类可以包含子类，子类可继承父类的数据和方法；
 - 不同的子类表现出多态性，如同一方法的效果多样性。

```
class Num {  
    int val;  
public:  
    Num(int val) {this->val = val;}  
    int factorial() {  
        int i, r = 1;  
        for (i = 1; i <= this->val; i++) {  
            r = r*i;  
        }  
        return r;  
    }  
};  
  
Num obj(10);  
obj.factorial();
```

不同的编程语言的等价性

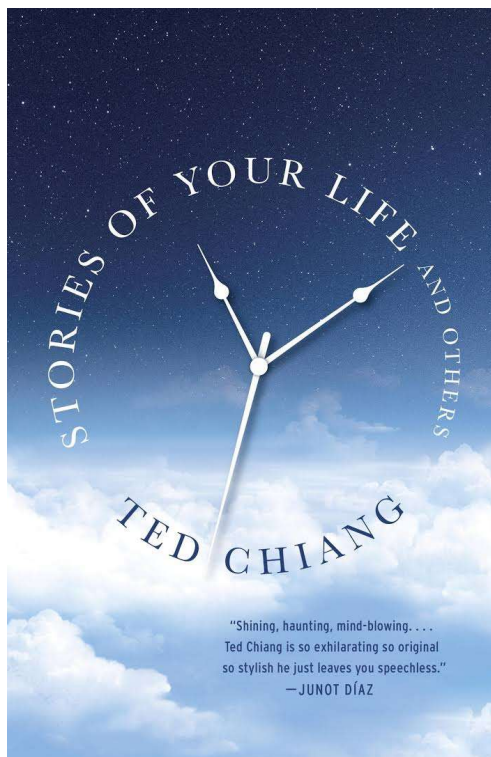
- 邱齐-图灵理论决定了哪些问题是可计算的：
 - 图灵机与Lambda算子能解决的问题集合等价；
 - 不同的通用编程语言（general purpose language）能解决的问题集合等价；
 - 如C、Java、Lisp
 - 领域特定语言（domain-specific language）的表达能力一般是受限的；
 - 如SQL、HTML、CSS
 - 高级语言和汇编语言能解决的问题集合等价。

语言影响编程习惯

- 语言特性决定了其用法
 - 面向对象：大量使用对象
 - 函数式编程：大量使用小的无副作用的函数
 - 声明式编程：在逻辑问题域内进行搜索

沃尔夫假说: Sapir-Whorf hypothesis

人类的思维模式会受其使用语言的影响



降临 Arrival (2016)



导演: 丹尼斯·维伦纽瓦

编剧: 埃里克·海瑟尔 / 姜峯楠

主演: 艾米·亚当斯 / 杰瑞米·雷纳 / 福里斯特·惠特克 / 迈克尔·斯图巴 / 马泰 / 更多...

类型: 剧情 / 科幻

官方网站: www.arrivalmovie.com

制片国家/地区: 美国 / 加拿大

语言: 英语 / 俄语 / 汉语普通话

上映日期: 2017-01-20(中国大陆) / 2016-09-01(威尼斯电影节) / 2016-11-11(美国)

片长: 116分钟

又名: 天煞异降(港) / 异星入境(台) / 你一生的故事 / 抵达 / 抵达者 / Story of Your Life

IMDb: tt2543164

豆瓣评分

7.8  380703人评价

5星  23.1%
4星  47.4%
3星  25.5%
2星  3.2%
1星  0.8%

好于 85% 科幻片

好于 66% 剧情片



利用不动点实现匿名递归

$$(\lambda x.x\ x)(\lambda x.x\ x)$$

应用 β reduction $\Rightarrow (\lambda x.x\ x)(\lambda x.x\ x)$
 β -normal form

$$X = (\lambda x.f(x\ x))(\lambda x.f(x\ x))$$

应用 β reduction $\Rightarrow X = f(\lambda x.f(x\ x)\ \lambda x.f(x\ x))$

替换 $\Rightarrow X = f(X)$

Y Combinator: $\lambda f.(\lambda x.f(x\ x))(\lambda x.f(x\ x))$

特性：对于任意输入参数（函数 f ）都返回不动点：

$$Yf = f(Yf) = f(f(Yf)) = f(f(\dots f(Yf)\dots))$$

匿名递归实现阶乘

令 $F := \lambda f. \lambda x. (\text{if } x == 0 \text{ then } 1 \text{ else } x * f (x-1))$

YF 3

F(YF) 3

$\lambda f. \lambda x. (\text{if } x == 0 \text{ then } 1 \text{ else } x * f (x-1))$ (YF) 3

if 3 == 0 then 1 else 3 * (YF)(3-1)

3 * (YF) 2

3 * F(YF) 2

3 * ($\lambda f. \lambda x. (\text{if } x == 0 \text{ then } 1 \text{ else } x * f (x-1))$ (YF) 2)

3 * (if 2 == 0 then 1 else 2 * (YF)(2-1))

3 * (2 * (YF) 1)

6 * (YF) 1

6 * F(YF) 1

6 * ($\lambda f. \lambda x. (\text{if } x == 0 \text{ then } 1 \text{ else } x * f (x-1))$ (YF) 1)

6 * (if 1 == 0 then 1 else 1 * (YF)(1-1))

6 * (YF) 0

6 * F(YF) 0

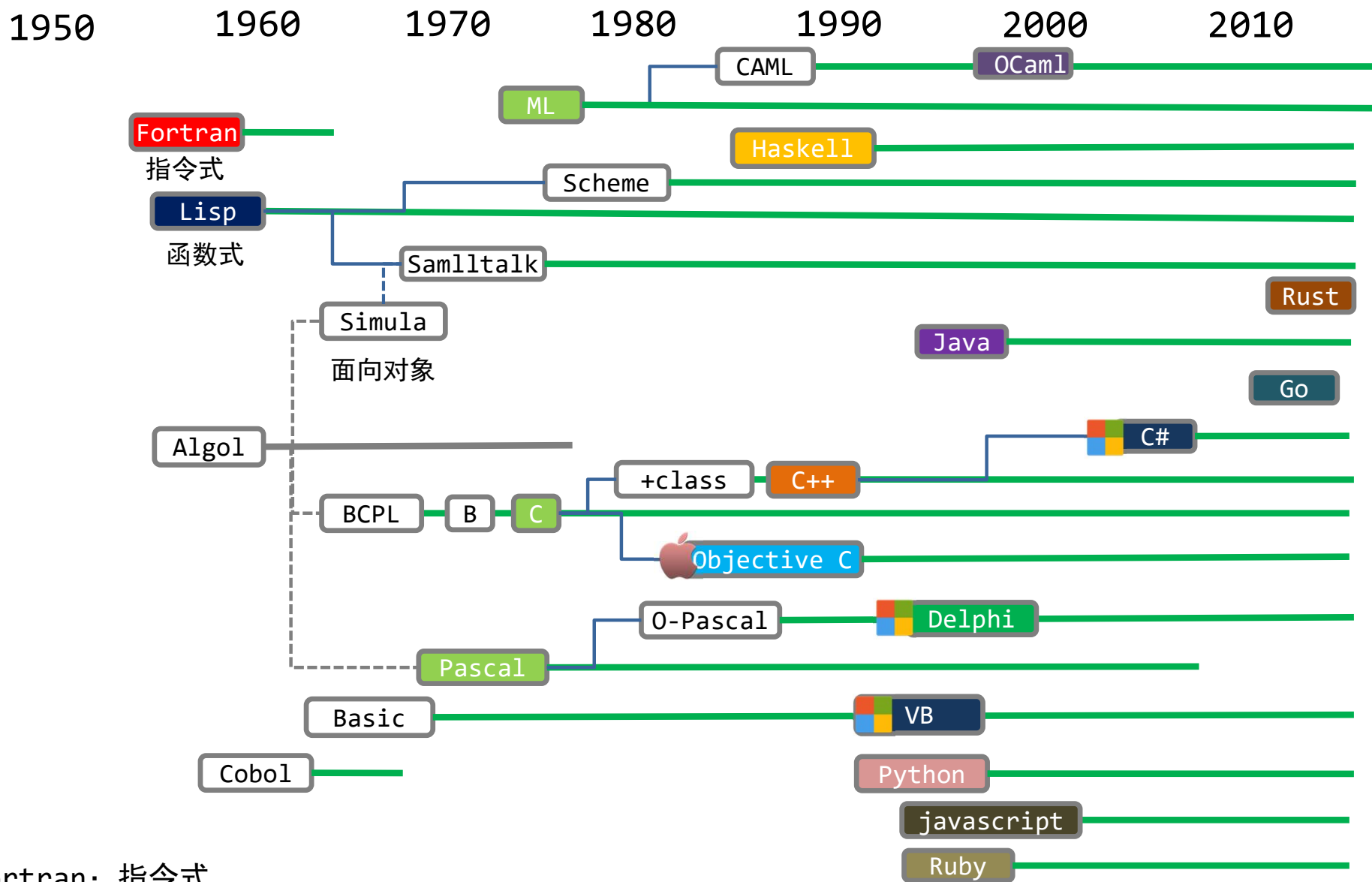
6 * ($\lambda f. \lambda x. (\text{if } x == 0 \text{ then } 1 \text{ else } x * f (x-1))$ (YF) 0)

6 * (if 0 == 0 then 1 else 0 * (YF)(0-1))

6 * 1

6

编程语言发展简史



Fortran: 指令式
Lisp: 函数式
Algol: 指令式
Simula: 面向对象

编程语言发展

- 新的语言不断涌现，旧的语言出现各种方言
- 广泛使用的语言并非创新性强，如Java（1995年）
 - 借鉴了很多C++的概念
 - C++是对C语言的扩充，结合了Simula、ML等语言思想
 - C语言由B语言发展来，B语言继承Algol的特性
- 未广泛使用的语言并非创新性弱，如Algol
 - 最早的编程语言之一，Algol 58、Algol 60、Algol 68
 - 首次引入了很多被广泛借鉴的编程思想

如何对语言进行分类？

- 语言并非纯粹的属于某一编程范式，而是多范式的。
- 很多语言范式或特性的概念是模糊的
 - 坏问题：X语言是函数式编程语言吗？
 - 好问题：X语言支持函数式编程吗？怎样支持的？

编程语言是否还有研究价值？

- 编程语言历史学术会议HOPL (History of programming languages)
 - 2021年是第四届
- 1996年1月，第二界HOPL会议达成的基本共识是编程语言研究已结束，因为所有重要的概念基本都发明了。
 - 但在此之后，Java诞生，并取得巨大成功
- 语言发展的推动力：
 - 语言是有寿命的，不断发展迭代；
 - 新语言刚开始因为某些不可替代的特性在10-20年间活跃；
 - 在此之后通常需要变化、升级来适应需求；
 - 有些语言（如Cobol）因为仅因维护需要还有人在使用。

内容总结

- 编译器的框架：前端-中间代码层-后端
 - 前端（理解程序）：词法分析-句法分析
 - 中间层（分析程序）：AST-类型检查-线性IR-代码优化
 - 后端（翻译程序）：指令选址-指令调度-寄存器分配
- 编译器类型和一些基本概念
 - 编译执行、解释执行、JIT、AOT
 - 静态类型和动态类型
 - 运行时环境和垃圾回收
- 编程语言范式和简史
 - 命令式、函数式、声明式、面向对象