Lecture 0

编译原理课程简介

徐 辉

xuh@fudan.edu.cn

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；
- 通过接近自然语言的高级语言提升软件开发效率。

源代码

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

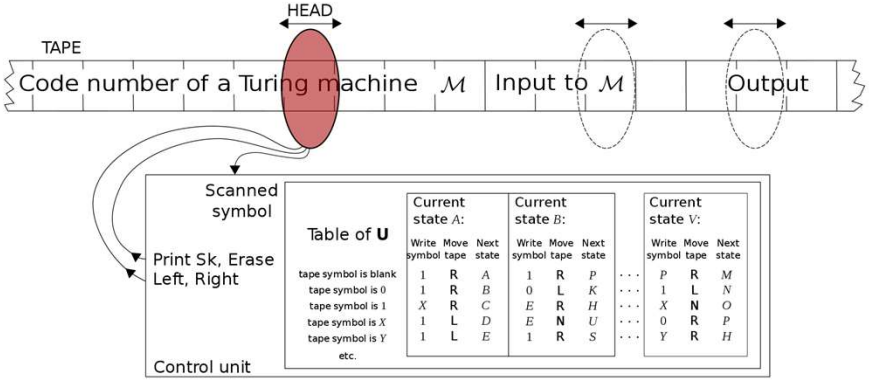
汇编

```
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
```

打孔卡



图灵机

编译器和编程语言设计的重要性

图灵奖得主

Liskov substitution principle

2020年
Aho & Ullman

2008年
Barbara Liskov

面向对象
(Smalltalk语言)

BNF范式/ALGOL 60

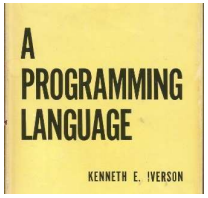
Optimizating Compilers

2003年
Alan Kay

2005年
Peter Naur

2006年
Frances Allen

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



FORTRAN
（语言）

ALGOL 60
（语言）

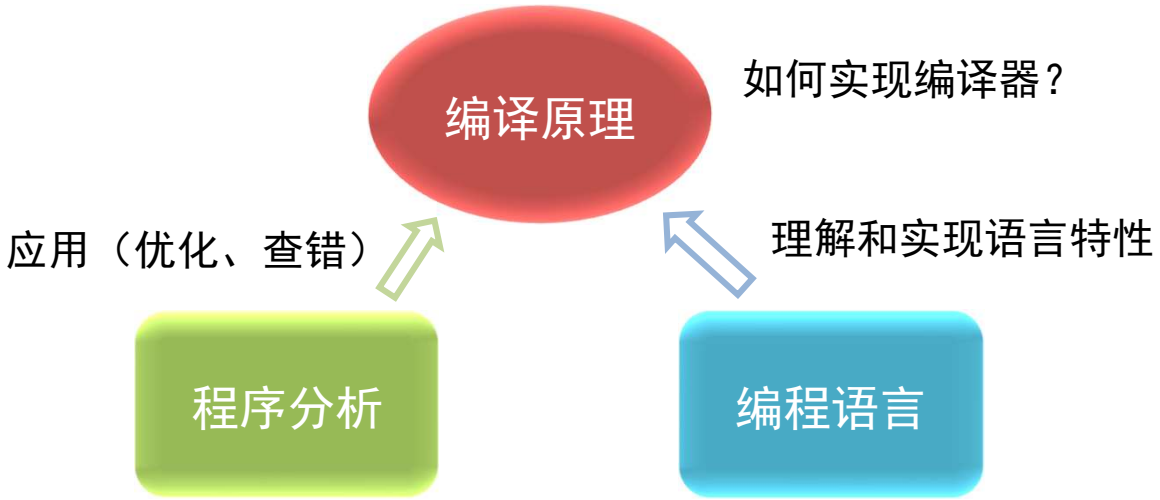
1980年
Tony Hoare

1979年
Kenneth Iverson

1977年
John Backus

1972年
Edsger Dijkstra

教学大纲



- 编译原理：
  - 词法分析
  - 语法分析
  - 中间代码生成
  - 汇编代码生成
- 编程语言：
  - 类型系统
  - 内存管理
  - 异常处理
- 程序分析：
  - 代码优化
    - 数据流分析
    - 指针分析

教学目标

- 理解编译器的工作原理和主要算法
- 掌握基础的的字符串分析问题的解决能力
- 动手实现简单的编译器或为其添加特定功能
- 了解现代编程语言的主要思想及其实现方法

字符串分析问题举例

- 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：($|¥|£|€)([1-9]|[1-9][0-9]|[1-9][0-9]^2)(‘,’ ,[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>

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：光华楼西辅楼310（HGX310）
- 上机实践：
  - 时间：[每双周] 9-10节（4:20pm-6:00pm）
  - 地点：计算中心机房A110
- 课程平台：
  - Elearning
  - WeChat

教学团队

- 授课教师：徐辉
  - Ph.D，CUHK
  - 研究方向：程序分析，软件可靠性和安全性
  - 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：

崔漠寒
Email：mhcui20@fudan.edu.cn
Office：江湾校区交叉二号楼D6010

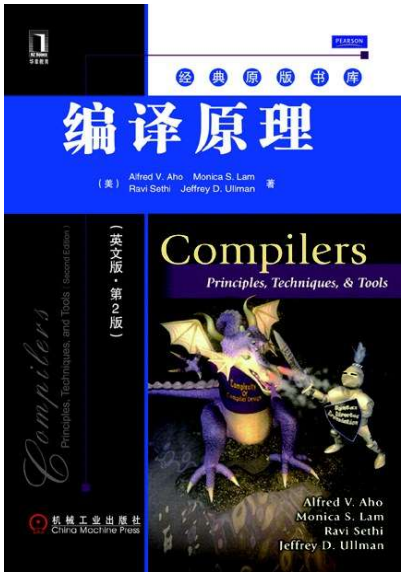
陈澄钧
Email：cjchen20@fudan.edu.cn
Office：江湾校区交叉二号楼D6010

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PPT讲解
    - 每人10分钟
- 闭卷考试：50%

主要参考书

- 《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
- 编程语言的相关内容主要是自制或参考其它老师的课件



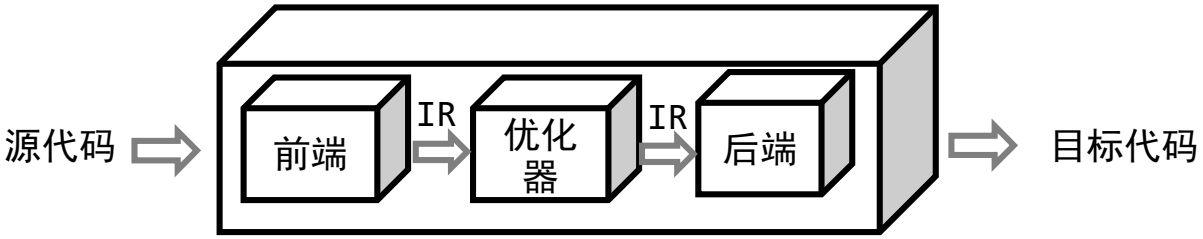
https://dl.acm.org/doi/pdf/10.5555/2737838

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RISC、MIPS
  - 基本要求：保持语义等价

编译器的基本结构

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



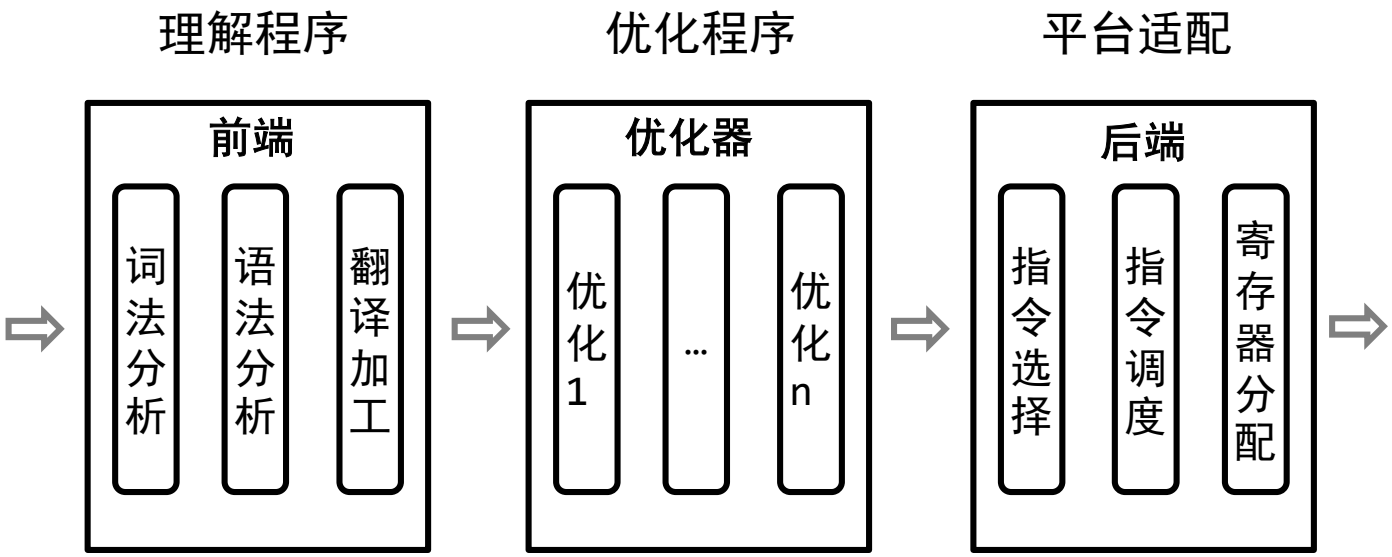

```
`-FunctionDecl 0x1405390 <hellocompiler.c:2:1, line:5:1> line:2:5 main 'int ()'
  `-CompoundStmt 0x1405598 <col:11, line:5:1>
    |-CallExpr 0x1405510 <line:3:3, col:30> 'int'
    | |-ImplicitCastExpr 0x14054f8 <col:3> 'int (*)(const char *, ...)' <FunctionToPointerDecay>
    | | `-DeclRefExpr 0x1405430 <col:3> 'int (const char *, ...)' Function 0x13f4f10 'printf' 'int (const char *, ...)'
    | `-ImplicitCastExpr 0x1405550 <col:10> 'const char *' <NoOp>
    |   `-ImplicitCastExpr 0x1405538 <col:10> 'char *' <ArrayToPointerDecay>
    |     `-StringLiteral 0x1405488 <col:10> 'char [18]' lvalue "hello, compiler!\n"
    `-ReturnStmt 0x1405588 <line:4:3, col:10>
      `-IntegerLiteral 0x1405568 <col:10> 'int' 0
```

虚拟语法树

```
define dso_local i32 @main() #0 {
  %1 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  %2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([18 x i8], [18 x i8]* @.str, i64 0, i64 0))
  ret i32 0
}
```

LLVM IR

编译器结构




```
define dso_local i32 @main() #0 {
  %1 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  %2 = call i32 (i8*, ...)
       @printf(i8* getelementptr inbounds
       ([18 x i8], [18 x i8]* @.str,
       i64 0, i64 0))
  ret i32 0
}
```

LLVM IR

```
push   %rax
mov    $0x402004,%edi
callq  0x401030 <puts@plt>
xor    %eax,%eax
pop    %rcx
retq
```

LLVM IR

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。

$$Sentence \rightarrow Subject||verb||Object||endmark$$

句子　导出　主语　　动词　　宾语　　结束符

例句：Compilers are engineered objects.

(noun, “Compilers”)
(verb, “are”)
(adjective, “engineered”)
(noun, “objects”)
(endmark, “.”)

语法分析：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

[1] $Sentence \rightarrow Subject||verb||Object||endmark$

[2] $Subject \rightarrow noun|modifier\ noun$

[3] $Object \rightarrow noun|Modifier||noun$

[4] $Modifier \rightarrow adjective$

语法解析

(noun, “Compilers”)(verb, “are”)(adjective, “engineered”)(noun, “objects”)(endmark, “.”)

$\boldsymbol{Sentence}$

[1] ⇨ $\boldsymbol{Subject}||verb||Object||endmark$

[2] ⇨ $noun||verb||\boldsymbol{Object}||endmark$

[3] ⇨ $noun||verb||\boldsymbol{Modifier}||noun||endmark$

[4] ⇨ $noun||verb||adjective\ noun||endmark$

语法制导翻译：Translation

- 生成中间代码，去除语法糖（如for/while循环）
- 通过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
  - 如类型检查或推导

```
while (r < 100){
    if (s < r)
        s = s+r;
    else
        r = s+r;
}
```

```
4 (Basic Block):                              ; preds = %19, %0
  %5 = load i32, i32* %3, align 4
  %6 = icmp slt i32 %5, 100
  br i1 %6, label %7, label %20
7 (Basic Block):                              ; preds = %4
  %8 = load i32, i32* %2, align 4
  %9 = load i32, i32* %3, align 4
  %10 = icmp slt i32 %8, %9
  br i1 %10, label %11, label %15
11 (Basic Block):                             ; preds = %7
  %12 = load i32, i32* %2, align 4
  %13 = load i32, i32* %3, align 4
  %14 = add nsw i32 %12, %13
  store i32 %14, i32* %2, align 4
  br label %19
15 (Basic Block):                             ; preds = %7
  %16 = load i32, i32* %2, align 4
  %17 = load i32, i32* %3, align 4
  %18 = add nsw i32 %16, %17
  store i32 %18, i32* %3, align 4
  br label %19
19 (Basic Block):                             ; preds = %15, %11
  br label %4
```

优化：Optimization

- 数据流分析
  - 常量传导
  - 循环优化
  - …

```
int a = 1;
int b = 2;
int c = 3;
for (i=1; i<100; i++){
   int d = getInt();
   a = a x 2 x b x c x d;
}
```

优化

```
int a = 1;
int b = 2;
int c = 3;
int t = 2 x b x c;
for (i=1; i<100; i++){
   int d = getInt();
   a = a x t x 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

```
t0 ← a x 2
t1 ← t0 x b
t2 ← t1 x c
t3 ← t2 x d
a ← t3
```

伪汇编代码

```
load rarp, @a ⟹ ra
load 2 ⟹ r2
load rarp, @b ⟹ rb
load rarp, @c ⟹ rc
load rarp, @d ⟹ rd
mult ra, r2 ⟹ ra
mult ra, rb ⟹ ra
mult ra, rc ⟹ ra
mult ra, rd ⟹ ra
store ra ⟹ rarp @a
```

指令调度：Instruction Reordering

- 根据计算机的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：
  - 3：load，Store
  - 2：mult
  - 1：其余指令

开始，结束

1, 3	load rarp, @a ⟹ r1
4, 4	add r1, r1 ⟹ r1
5, 7	load rarp, @b ⟹ r2
8, 9	mult r1, r2 ⟹ r1
10,12	load rarp, @c ⟹ r2
13,14	mult r1, r2 ⟹ r1
15,17	load rarp, @d ⟹ r2
18,19	mult r1, r2 ⟹ r1
20,22	store r1 ⟹ rarp @a

优化

开始，结束

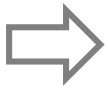
1, 3	load rarp, @a ⟹ r1
2, 4	load rarp, @b ⟹ r2
3, 5	load rarp, @c ⟹ r3
4, 4	add r1, r1 ⟹ r1
5, 6	mult r1, r2 ⟹ r1
6, 8	load rarp, @d ⟹ r2
7, 8	mult r1, r3 ⟹ r1
9,10	mult r1, r2 ⟹ r1
11,13	store r1 ⟹ rarp @a

寄存器分配：Registor 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。

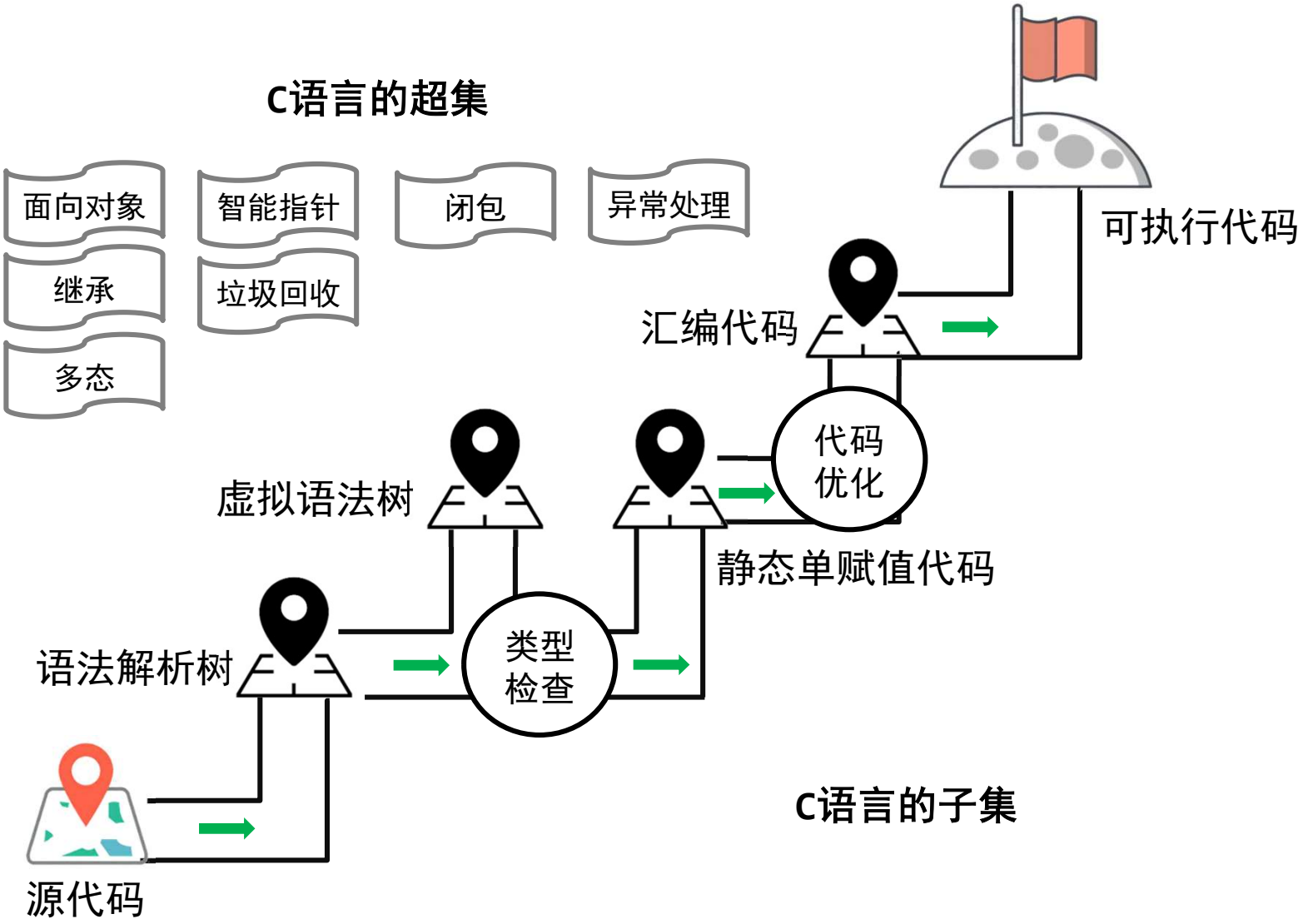
```
load rarp, @a ⟹ ra
load 2 ⟹ r2
load rarp, @b ⟹ rb
load rarp, @c ⟹ rc
load rarp, @d ⟹ rd
mult ra, r2 ⟹ ra
mult ra, rb ⟹ ra
mult ra, rc ⟹ ra
mult ra, rd ⟹ ra
store ra ⟹ rarp @a
```

优化



```
load rarp, @a ⟹ r1
add r1, r1 ⟹ r1
load rarp, @b ⟹ r2
mult r1, r2 ⟹ r1
load rarp, @c ⟹ r2
mult r1, r2 ⟹ r1
load rarp, @d ⟹ r2
mult r1, r2 ⟹ r1
store r1 ⟹ rarp @a
```

学习地图


C语言的超集
面向对象
智能指针
闭包
异常处理
继承
垃圾回收
多态
可执行代码
汇编代码
代码
优化
虚拟语法树
静态单赋值代码
类型
检查
语法解析树
C语言的子集
源代码

问题

- 用Java语言可以为C语言写编译器吗？
- 编译器是用什么工具编写或编译的？
  - 用其它编译器编译
  - 自举（bootstrap）：用该语言旧版本的编译器编译
- 不同的编译器编译同一段代码生成的程序相同吗？
- 如果编译器有bug，会导致什么后果？
- 举例说出编译器以外的哪些软件用到了编译技术？
  - 浏览器
  - Latex
- 你能想到编写编译器存在哪些难点？