

编译原理概述

魏恒峰

hfwei@nju.edu.cn

2020 年 11 月 02 日 (周二)



2 个月 = 8 周 = 16 次课



2 个月 = 8 周 = 16 次课



希望大家都能“安然无恙”地度过这两个月



邀请码: JCZ837HE

每周五晚上布置作业

下周五 23 : 55 前提交作业

$$45 = \textcolor{red}{0} + 5 + 15 + 15 + 10 + \textcolor{red}{5}$$

实验列表

- L0：环境配置（不占分，11月10日-11月15日）
- L1：词法分析（5分，11月12日-11月22日）
- L2：语法分析（15分，11月24日-12月6号）
- L3：语义分析（15分，12月8号-12月27号）
- L4：中间代码生成（10分，12月17号-1月3号）
- L5：目标代码生成（5分，待定）

L0 : 环境配置已经开放

QQ 群号: 711805817



QQ 验证: 2021-编译原理

courses-at-nju-by-hfwei / compilers-lectures (Public)

Code Issues Pull requests Actions Projects

master → compilers-lectures / 2021 /

hengxin +2021/

..

0-overview +2021/

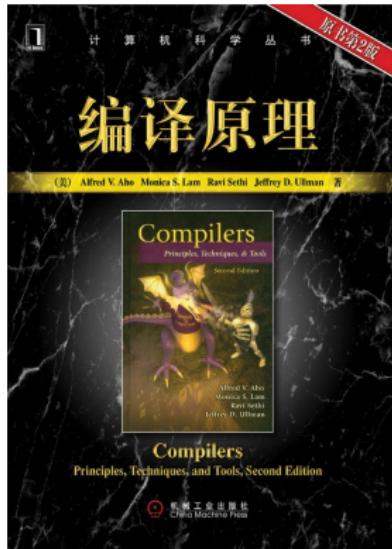
README.md +2021/

README.md

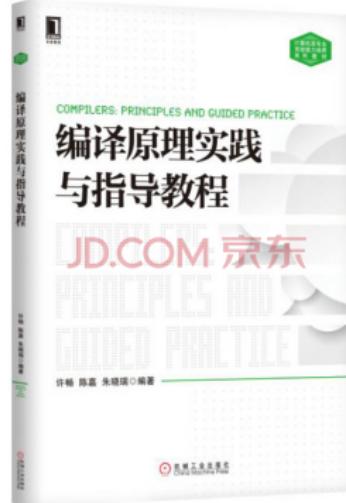
compilers-lectures

Lectures for the class on Compilers@software.nju.edu.cn.

<https://github.com/courses-at-nju-by-hfwei/compilers-lectures/tree/master/2021>



也可使用“本科教学版”



[https://cs.nju.edu.cn/
changxu/2_compiler/index.html](https://cs.nju.edu.cn/changxu/2_compiler/index.html)

The Definitive
ANTLR 4
Reference



Terence Parr

The
Pragmatic
Programmers

The
Pragmatic
Programmers

Language
Implementation
Patterns

Create Your Own Domain-
Specific and General
Programming Languages



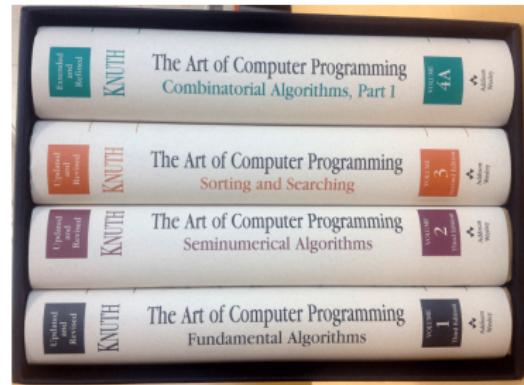
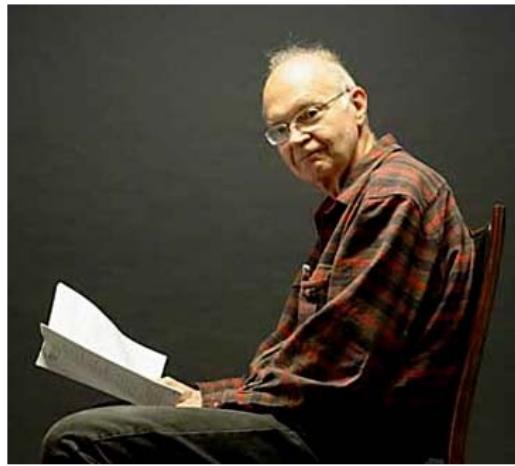
Terence Parr



<https://www.antlr.org/index.html>



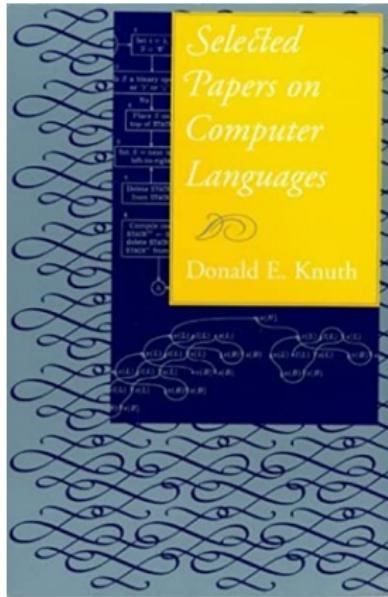
the “father of the analysis of algorithms”



Donald E. Knuth (1938 ~)

Turing Award, 1974

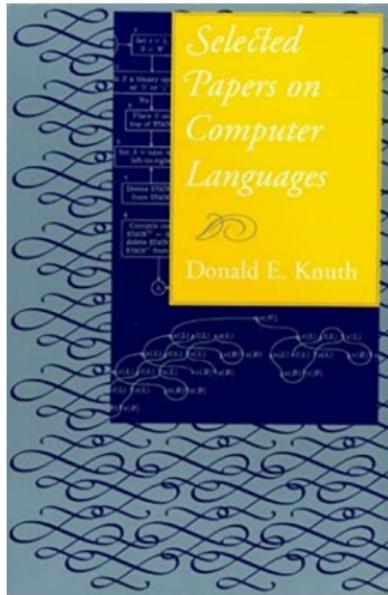
“Selected Papers on Computer Languages”



LR Parser (语法)

Attribute Grammar (语义)

“Selected Papers on Computer Languages”



LR Parser (语法)

Attribute Grammar (语义)

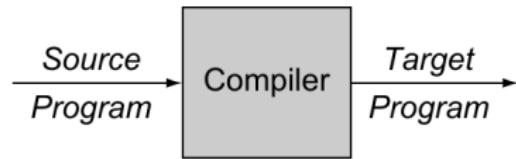
ALGOL 58 Compiler



“I got a job at the end of my senior year
to write a compiler for Burroughs”

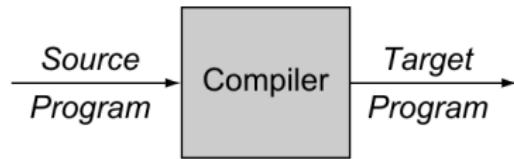
“高级”语言 \Rightarrow (通常) “低级”语言 (如, 汇编语言)

汇编语言经过**汇编器**生成机器语言



“高级”语言 \Rightarrow (通常) “低级”语言 (如, 汇编语言)

汇编语言经过汇编器生成机器语言



GopherJS - A compiler from Go to JavaScript

godoc reference PASSED

GopherJS compiles Go code (golang.org) to pure JavaScript code. Its main purpose is to give you the opportunity to write front-end code in Go which will still run in all browsers.

语言类应用程序

- ▶ 配置文件解析 (.properties)
- ▶ CSV 文件 (Comma-Separated Values)
- ▶ JSON 文件 (JavaScript Object Notation)

语言类应用程序

- ▶ 配置文件解析 (.properties)
- ▶ CSV 文件 (Comma-Separated Values)
- ▶ JSON 文件 (JavaScript Object Notation)
- ▶ SQL 引擎 (Structured Query Language)
- ▶ (Java) 字节码解释器
- ▶ C/C++ 语言编译器

语言类应用程序

- ▶ 配置文件解析 (.properties)
- ▶ CSV 文件 (Comma-Separated Values)
- ▶ JSON 文件 (JavaScript Object Notation)
- ▶ SQL 引擎 (Structured Query Language)
- ▶ (Java) 字节码解释器
- ▶ C/C++ 语言编译器
- ▶ 排版工具 (L^AT_EX)
- ▶ 绘图工具 (TikZ, Dot/Graphviz)

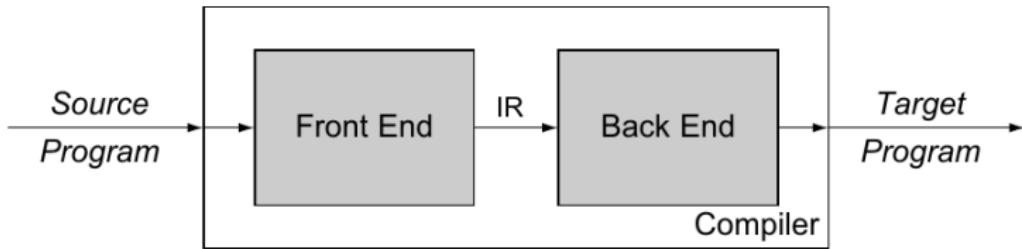
语言类应用程序

- ▶ 配置文件解析 (.properties)
- ▶ CSV 文件 (Comma-Separated Values)
- ▶ JSON 文件 (JavaScript Object Notation)
- ▶ SQL 引擎 (Structured Query Language)
- ▶ (Java) 字节码解释器
- ▶ C/C++ 语言编译器
- ▶ 排版工具 (L^AT_EX)
- ▶ 绘图工具 (TikZ, Dot/Graphviz)
- ▶ L-System (Cantor Set)

两个月的“编译器设计原理”之旅

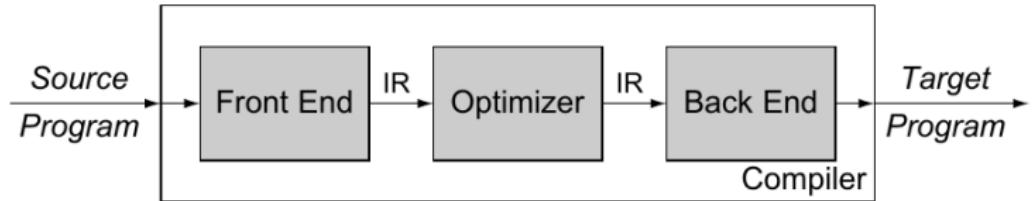


IR: Intermediate Representation (中间表示)



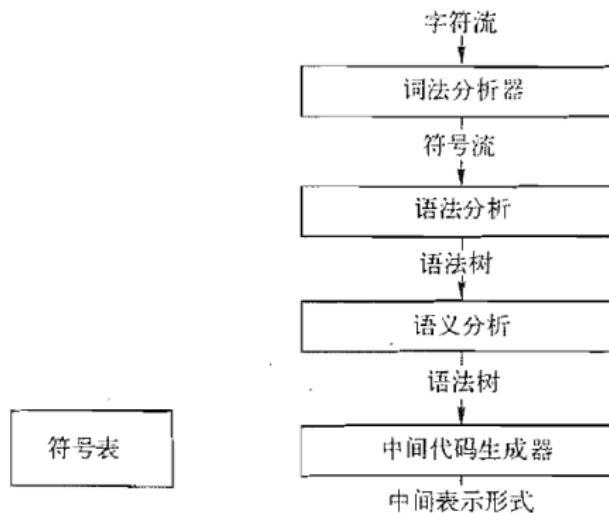
前端（分析阶段）: 分析源语言程序, 收集所有必要的信息

后端（综合阶段）: 利用收集到的信息, 生成目标语言程序

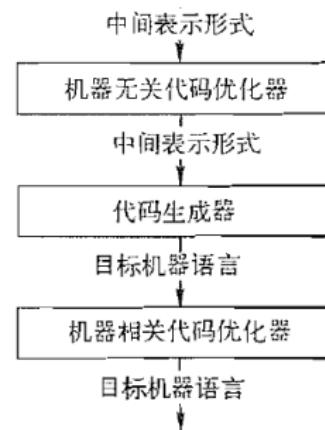


机器无关的中间表示优化

编译器前端：分析阶段



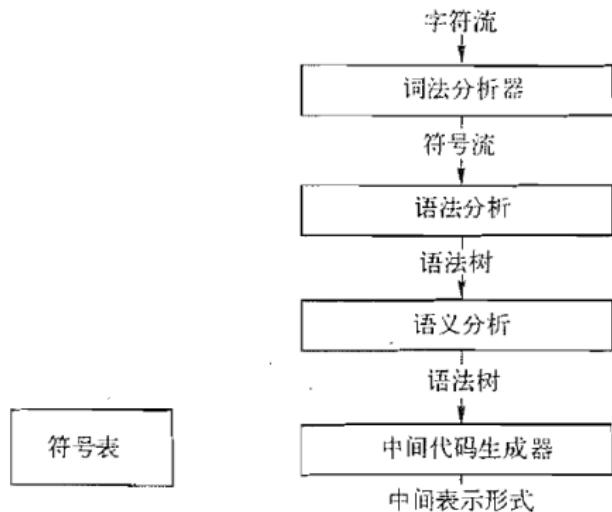
编译器后端：综合阶段



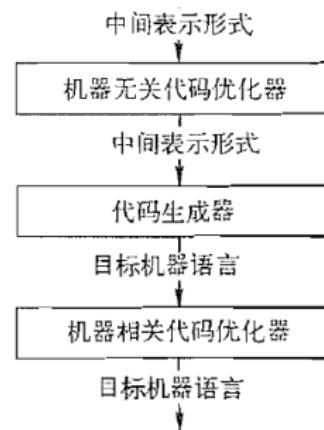
前四次必做实验

第五次选做实验

编译器前端：分析阶段



编译器后端：综合阶段



第五次选做实验

前四次必做实验

```
position = initial + rate * 60
```

作为一名**程序员**, 你看到了什么?

```
position = initial + rate * 60
```

作为一名**程序员**, 你看到了什么?

```
position = initial + rate * 60
```

词法: 标识符、数字、运算符

作为一名**程序员**, 你看到了什么?

```
position = initial + rate * 60
```

词法: 标识符、数字、运算符

语法: 包含算术运算的赋值语句

作为一名**程序员**, 你看到了什么?

```
position = initial + rate * 60
```

词法: 标识符、数字、运算符

语法: 包含算术运算的赋值语句

语义: position, initial, rate 是数值类型

作为一名**程序员**, 你看到了什么?

```
position = initial + rate * 60
```

词法: 标识符、数字、运算符

语法: 包含算术运算的赋值语句

语义: position, initial, rate 是数值类型

物理定律: 当前位置 = 初始位置 + 速度 × 时间

作为一名**程序员**, 你看到了什么?

```
position = initial + rate * 60
```

词法: 标识符、数字、运算符

语法: 包含算术运算的赋值语句

语义: position, initial, rate 是数值类型

物理定律: 当前位置 = 初始位置 + 速度 × 时间

但是, 作为**编译器**, 它仅仅看到了一个**字符串**

词法分析器 (Lexer/Scanner): 将字符流转化为词法单元 (token) 流。

token : <token-class, attribute-value>

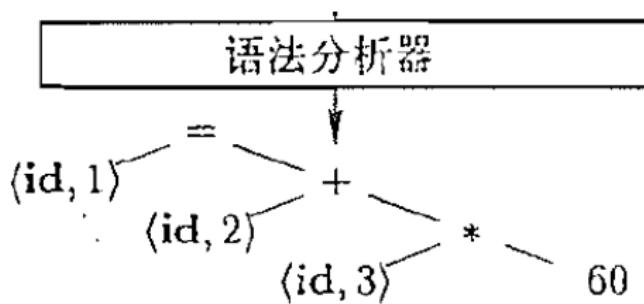
```
position = initial + rate * 60
```

<**id**, 1> <ws> <**assign**> <ws> <**id**, 2> <ws>
<+> <ws> <**id**, 3> <ws> <*> <ws> <**num**, 4>

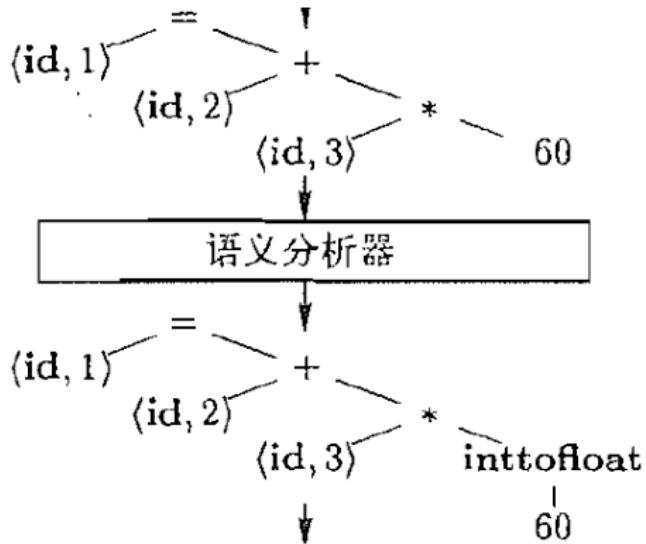
(此处, 1, 2, 3, 4 是指向**符号表**的指针)

语义分析器 (Parser): 构建词法单元之间的语义结构, 生成**语义树**

```
position = initial + rate * 60
```

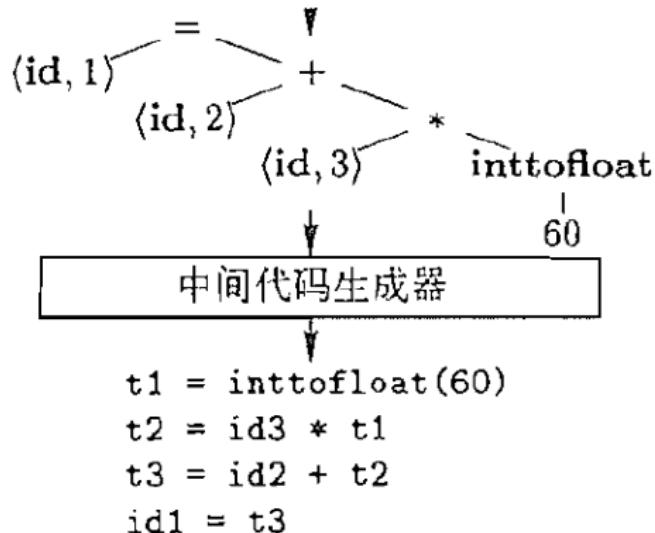


语义分析器：语义检查，如类型检查、“先声明后使用”约束检查



通过语法树上的遍历来完成

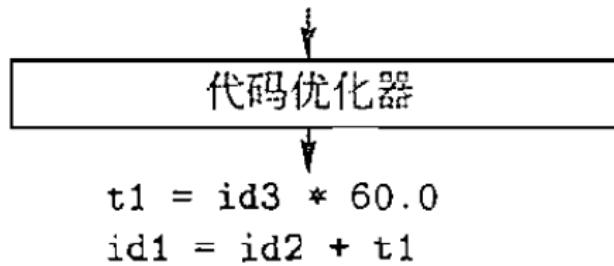
中间代码生成器：生成中间代码，如“三地址代码”



中间代码类似目标代码，但不含有机器相关信息（如寄存器、指令格式）

中间代码优化器

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```



编译时计算、消除冗余临时变量

代码生成器: 生成目标代码, 主要任务包括**指令选择、寄存器分配**

```
t1 = id3 * 60.0  
id1 = id2 + t1
```



代码生成器



```
LDF R2, id3  
MULF R2, R2, #60.0  
LDF R1, id2  
ADDF R1, R1, R2  
STF id1, R1
```

符号表: 收集并管理变量名/函数名相关的信息

变量名:

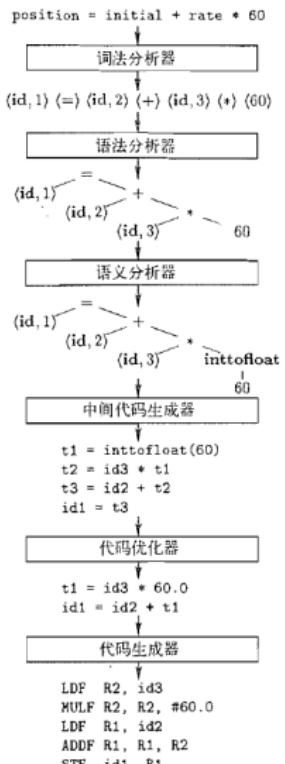
类型、寄存器、内存地址、行号

函数名:

参数个数、参数类型、返回值类型

| | | |
|---|----------|-----|
| 1 | position | ... |
| 2 | initial | ... |
| 3 | rate | ... |
| | | |

符号表

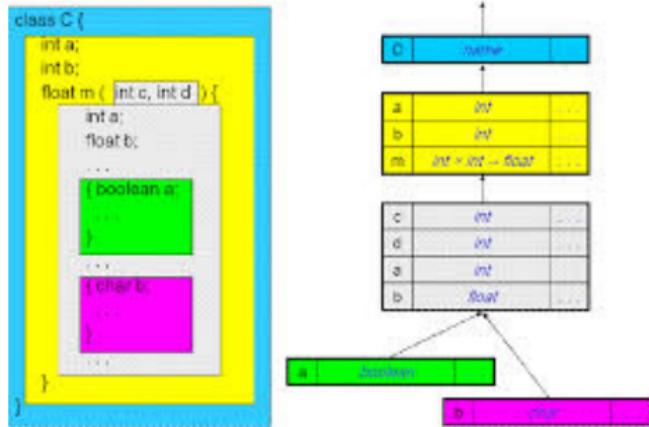


```
public class ST<Key extends Comparable<Key>, Value>
```

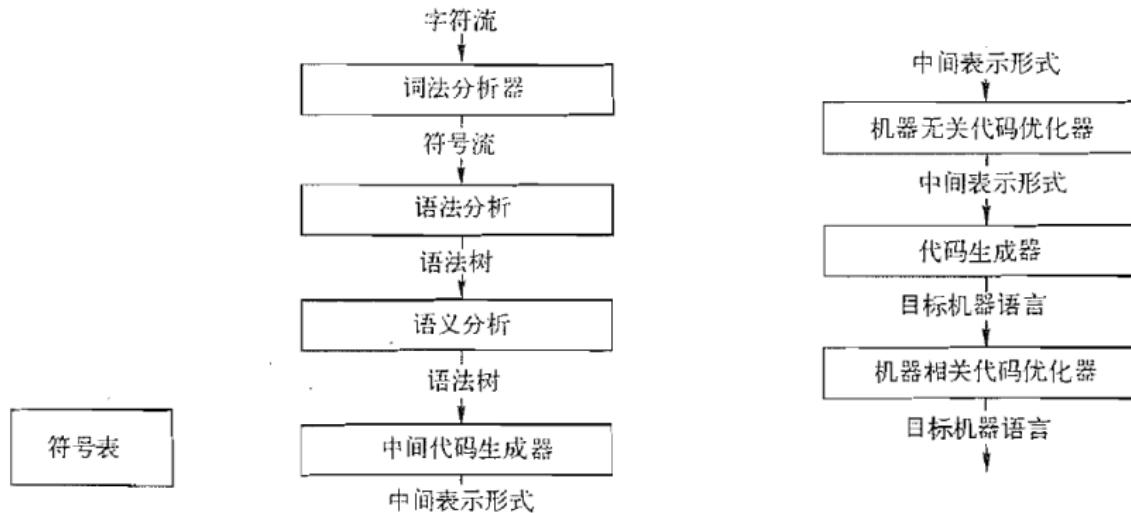
| | |
|------------------------------|--|
| ST() | <i>create an empty symbol table</i> |
| void put(Key key, Value val) | <i>associate val with key</i> |
| Value get(Key key) | <i>value associated with key</i> |
| void remove(Key key) | <i>remove key (and its associated value)</i> |
| boolean contains(Key key) | <i>is there a value associated with key?</i> |
| int size() | <i>number of key-value pairs</i> |
| Iterable<Key> keys() | <i>all keys in the symbol table</i> |

红黑树 (RB-Tree)、哈希表 (Hashtable)

为了方便表达嵌套结构与作用域, 可能需要维护多个符号表

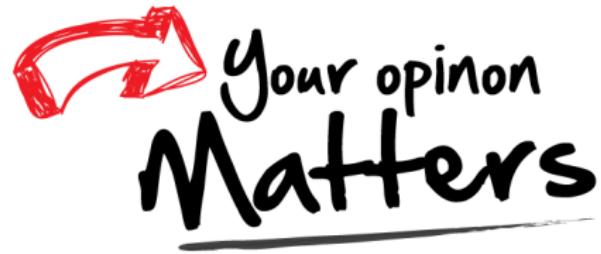


时间苦短，来不及优化



但是，在设计实际生产环境中的编译器时，**优化**通常占用了大多数时间

Thank You!



Office 926

hfwei@nju.edu.cn