

编译原理概述

魏恒峰

hfwei@nju.edu.cn

2020 年 11 月 10 日 (周二)



2 个月 = 8 周 = 16 次课



期末测试 (40 分): 考试周统一安排; 时长 2 小时; 闭卷

作业 (15 分): 7 次必做作业 + 1 次选做报告 (+3 分)

实验 (45 分): 4 次必做实验 + 1 次选做实验 (+5 分)



Moodle

<http://219.219.120.72>

选课密码: grade18

每周二布置作业

下周四提交作业

$$45 = 0 + 5 + 15 + 15 + 10 + 5$$

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)

<http://problemoverflow.cn/compiler/>

L0: 环境配置已经开放

QQ 群号: 587330269



作业助教: 王腾、田汶哲、郑立铨

实验助教: 张灵毓、何伟

外援: 张天昀、唐瑞泽

Recent questions and answers

- ▲ 1
▼ 0

1
answer

如何使用编辑器？

answered Feb 26, 2019 in meta by tangruize (35 points)

48 views

meta
- ▲ 1
▼ 0

1
answer

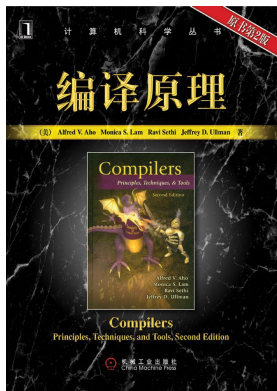
如何提问以及如何回答问题？

answered Feb 25, 2019 in meta by admin (34 points)

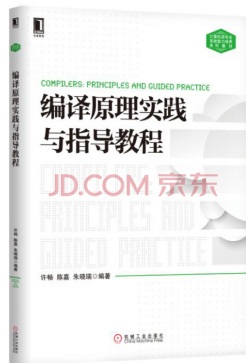
34 views

meta

<http://problemoverflow.cn/>



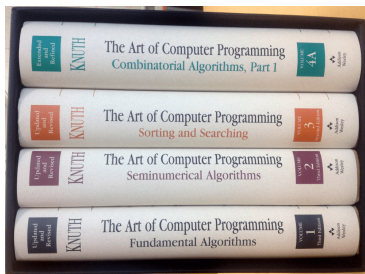
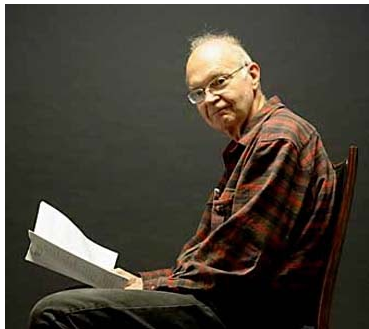
也可使用“本科教学版”



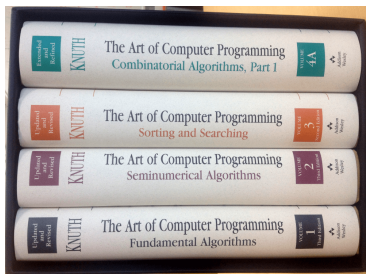
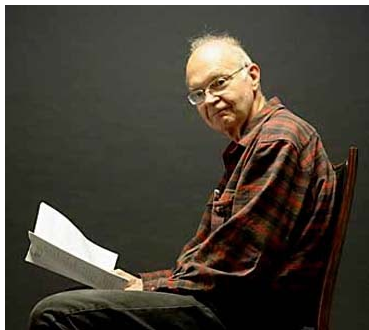
https://cs.nju.edu.cn/changxu/2_compiler/index.html



the “father of the analysis of algorithms”



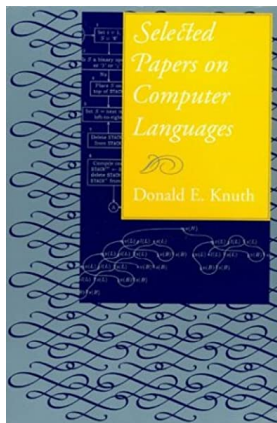
the “father of the analysis of algorithms”



Donald E. Knuth (1938 ~)

Turing Award, 1974

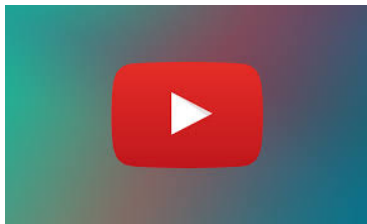
“Selected Papers on Computer Languages”



LR Parser

Attribute Grammar

ALGOL 58 Compiler



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

$$55K = 50K + 5K \text{ (子过程)}$$

$$55K = 50K + 5K \text{ (子过程)}$$



进展顺利的话, 各位在**圣诞节**就能用上自己的编译器了

$$55K = 50K + 5K \text{ (子过程)}$$

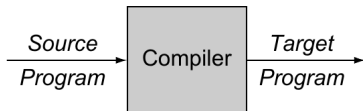


进展顺利的话, 各位在**圣诞节**就能用上自己的编译器了

免责声明: 关于能否顺利找到女朋友/男朋友, 本课程无法提供任何保证

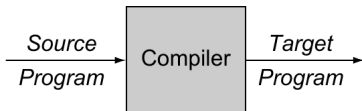
“高级”语言 \Rightarrow (通常) “低级”语言 (如, 汇编语言)

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 * used by 1.2k projects **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.

Q : 机器语言是如何跑起来的?

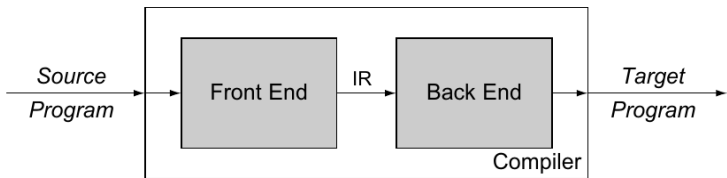
Q : 机器语言是如何跑起来的?

作业: <https://www.bilibili.com/video/BV1EW411u7th>

两个月的“编译器设计原理”之旅

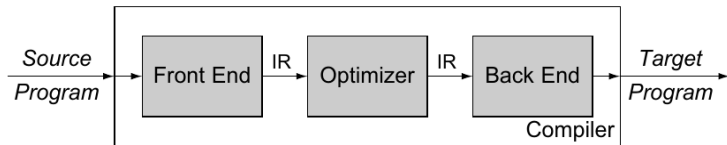


IR: Intermediate Representation (中间表示)



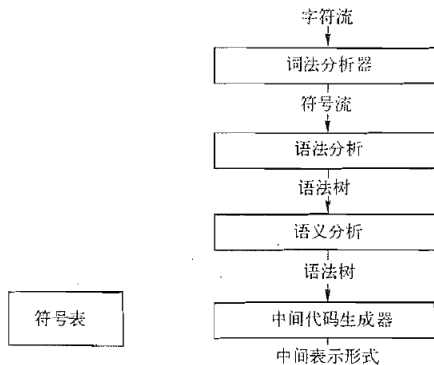
前端 (分析阶段): 分析源语言程序, 收集所有必要的信息

后端 (综合阶段): 利用收集到的信息, 生成目标语言程序



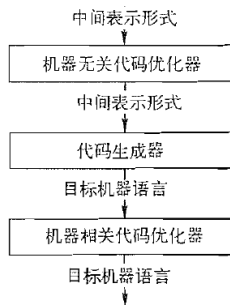
机器无关的中间表示优化

编译器前端：分析阶段



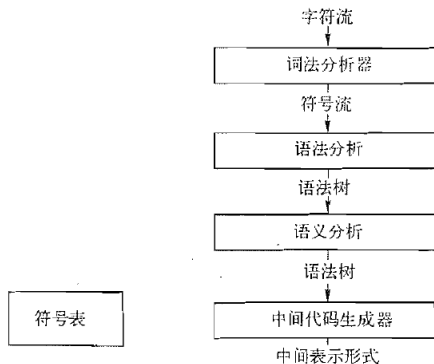
前四次必做实验

编译器后端：综合阶段



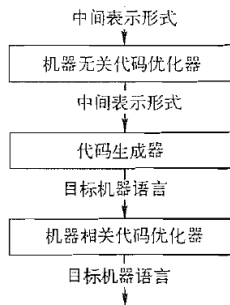
第五次选做实验

编译器前端：分析阶段



前四次必做实验

编译器后端：综合阶段



第五次选做实验

`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 : $\langle \text{token-class}, \text{attribute-value} \rangle$

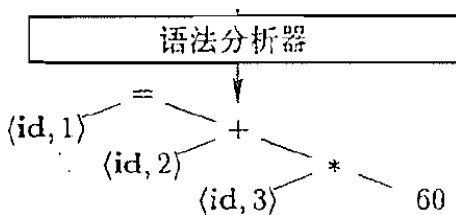
position = initial + rate * 60

$\langle \text{id}, 1 \rangle$ $\langle \text{ws} \rangle$ $\langle \text{assign} \rangle$ $\langle \text{ws} \rangle$ $\langle \text{id}, 2 \rangle$ $\langle \text{ws} \rangle$
 $\langle + \rangle$ $\langle \text{ws} \rangle$ $\langle \text{id}, 3 \rangle$ $\langle \text{ws} \rangle$ $\langle * \rangle$ $\langle \text{ws} \rangle$ $\langle \text{num}, 4 \rangle$

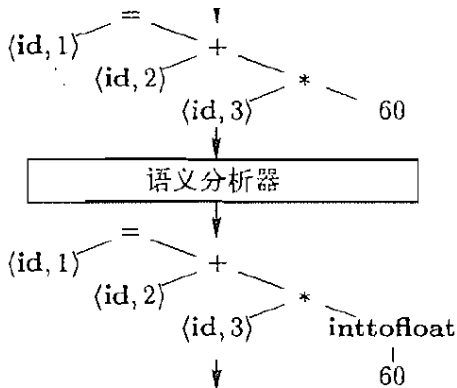
(此处, 1, 2, 3, 4 是指向**符号表**的指针)

语法分析器 (Parser): 构建词法单元之间的语法结构, 生成**语法树**

`position = initial + rate * 60`

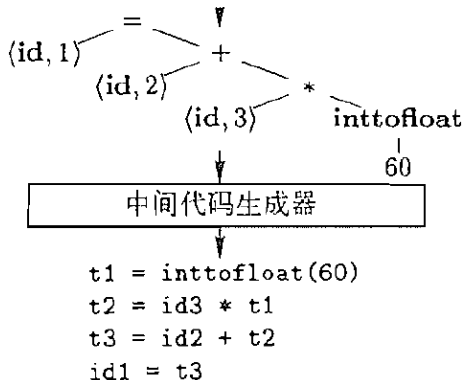


语义分析器: 语义检查, 如类型检查、“先声明后使用”约束检查



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”

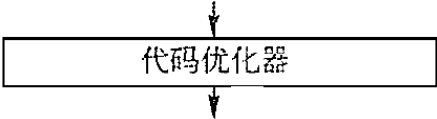


中间代码类似目标代码, 但不含有机器的相关信息 (如寄存器、指令格式)

中间代码优化器

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

↓



代码优化器

↓

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

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

position = initial + rate * 60

词法分析器

$\langle id, 1 \rangle \langle = \rangle \langle id, 2 \rangle \langle + \rangle \langle id, 3 \rangle \langle * \rangle \langle 60 \rangle$

语法分析器

$\langle id, 1 \rangle = \langle id, 2 \rangle + \langle id, 3 \rangle * 60$

语义分析器

$\langle id, 1 \rangle = \langle id, 2 \rangle + \langle id, 3 \rangle * \text{inttofloat}(60)$

中间代码生成器

$t1 = \text{inttofloat}(60)$
 $t2 = id3 * t1$
 $t3 = id2 + t2$
 $id1 = t3$

代码优化器

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

代码生成器

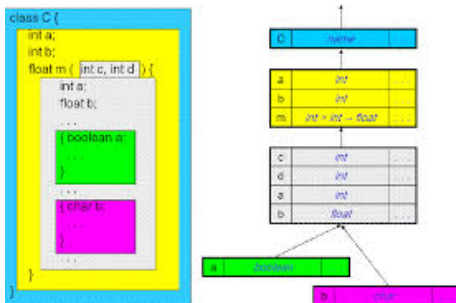
LDF R2, id3
MULF R2, R2, #60.0
LDF R1, id2
ADDF R1, R1, R2
STF id1, R1

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

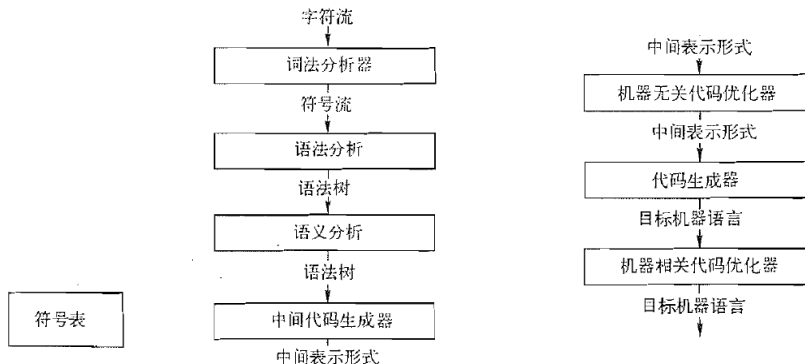
ST()	<i>create an empty symbol table</i>
void put(Key key, Value val)	<i>associate val with key</i>
Value get(Key key)	<i>value associated with key</i>
void remove(Key key)	<i>remove key (and its associated value)</i>
boolean contains(Key key)	<i>is there a value associated with key?</i>
int size()	<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