



第一章 引论

许畅

南京大学计算机系

2014年春季

课程概要 (1)

- 课程教材：《编译原理》（“龙书”）
- 讲课老师：许畅 (changxu@nju.edu.cn)
- 课程网站：[http://cs.nju.edu.cn/changxu/2 compiler/](http://cs.nju.edu.cn/changxu/2_compiler/)
- 授课时段：18周 (72学时)
- 授课安排：周二5-6节、周四3-4节
- 授课地点：仙I-204
- 上机地点：机房乙124 (周三/五9-10节)
- 课程助教：奚旺 (实验)、睦骏 (作业)

课程概要 (2)

- 课程意义
 - 计算机专业的**必修课** (4学分)
 - 不是学习编译器，是学习编译器设计的原理和技术
 - 研究生技能需求
 - 成为高手的必要一步 (编程大师、黑客等)
- 课程结构
 - 理论部分：上课听讲，下课作业，交书面**作业**
 - 实践部分：实现编译器的几个阶段，交上机**实验**

课程概要 (3)

■ 评分标准

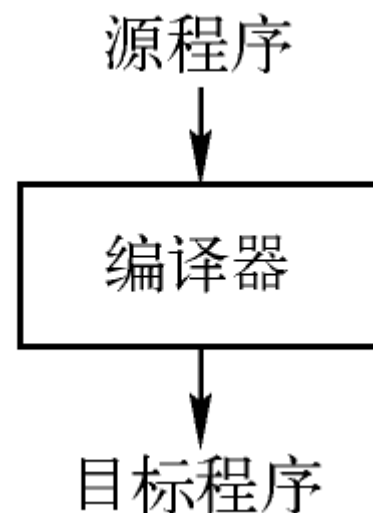
- 书面作业：10% (20% + 20% + 60%)
- 上机实验：30% (20% + 20% + 60%)
 - 组队调整：110% (1人), 100-105% (2人), 90-95% (3人)
 - 额外奖励：编译效率PK, 对实验的帮助等
 - 实验内容：抽签（普通班），所有（5班）
- 期末考试：60%
- 不可控因素：系里点名、作弊检查等

课程内容

- 1. 引论/编译器的总体结构 (易)
- 3. 词法分析 (难)
- 4. 语法分析 (难)
- 5. 语法制导的翻译技术 (中)
- 6. 中间代码生成 (难)
- 7. 运行时刻环境 (易)
- 8. 代码生成 (中)
- 9. 机器无关优化 (中)

编译器的作用

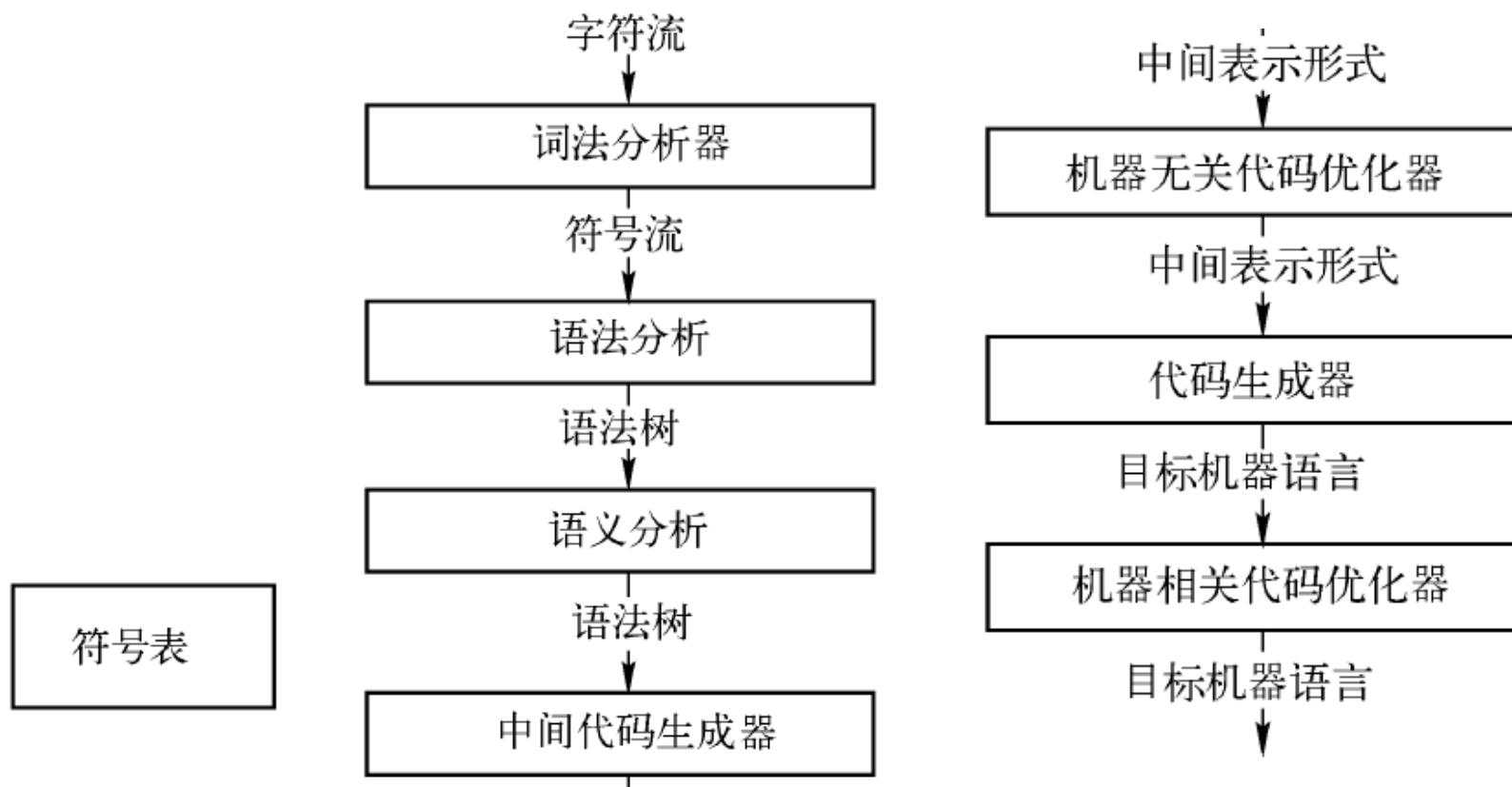
- 读入以某种语言 (源语言) 编写的程序
- 输出等价的、用另一种语言 (目标语言) 编写的程序
- 通常目标程序是可执行的
- 解释器
 - 直接利用用户提供的输入，执行源程序中指定的操作
 - 不生成目标程序，而是根据源程序的语义直接运行
 - Java语言的处理结合了编译和解释



编译器的结构 (1)

- 编译器可以分为分析部分和综合部分
- 分析部分 (前端/Front end)
 - 把源程序分解成组成要素，以及相应的语法结构
 - 使用这个结构创建源程序的中间表示
 - 同时收集和源程序相关的信息，存放于符号表
- 综合部分 (后端/Back end)
 - 根据中间表示和符号表信息构造目标程序
- 前端部分是机器无关的，后端部分是机器相关的

编译器的结构 (2)

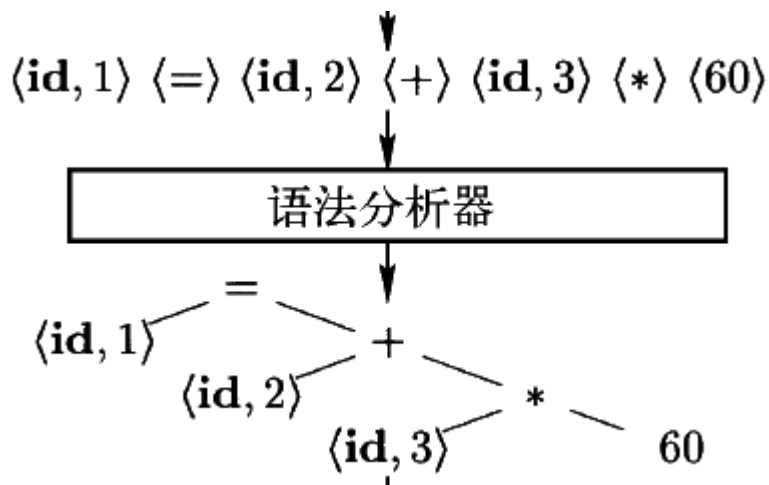


- 编译器可分成顺序执行的一组**步骤** (Phases)

词法分析

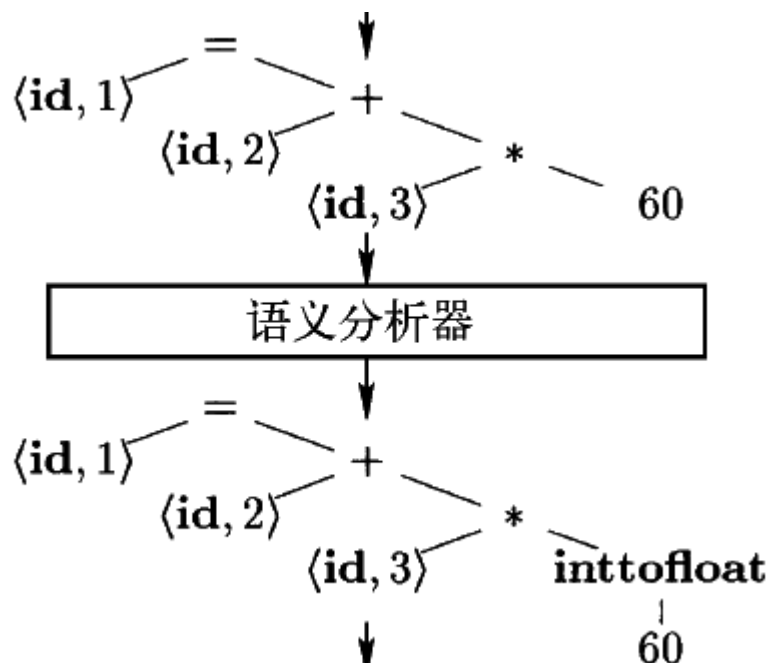
- 词法分析/扫描 (Lexical analysis/scanning)
 - 读入源程序的字符流，输出成为有意义的词素 (Lexeme)
 - `<token-name, attribute-value>`
 - `token-name`由语法分析步骤使用
 - `attribute-value`指向相应的符号表条目，由语义分析/代码生成步骤使用
- 例子
 - `position = initial + rate * 60`
 - `<id, 1>` `<=, >` `<id, 2>` `<+, >` `<id, 3>` `<*, >` `<number, 4>`

语法分析



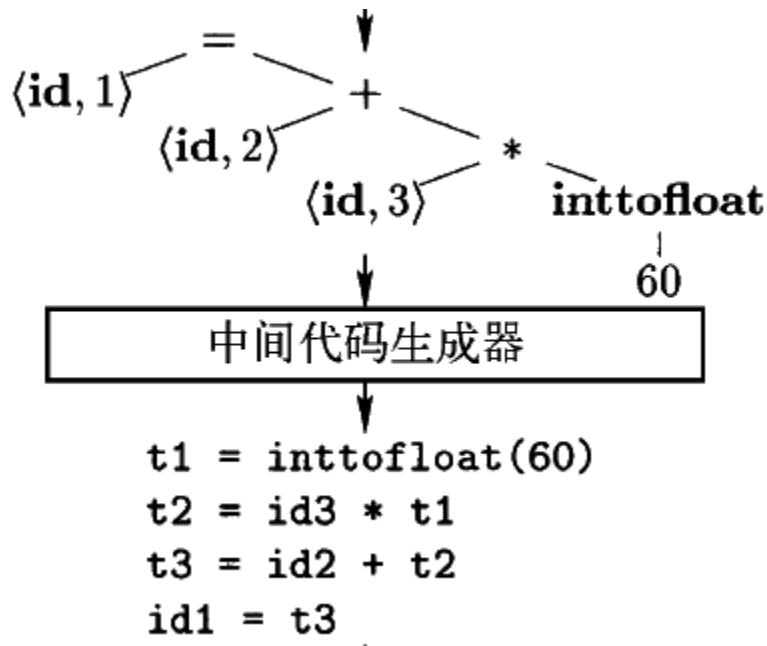
- 语法分析/解析 (Syntax analysis/parsing)
 - 根据各个词法单元的第一个分量来创建树型的中间表示形式，通常是语法树 (Syntax tree)
 - 中间表示形式指出了词法单元流的语法结构

语义分析



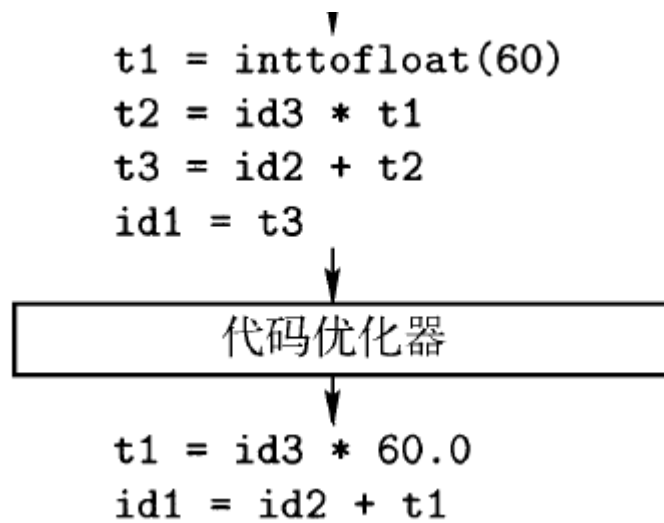
- 语义分析 (Semantic analysis)
 - 使用语法树和符号表中的信息，检查源程序是否满足语言定义的语义约束
 - 同时收集类型信息，用于代码生成，类型检查，类型转换

中间代码生成



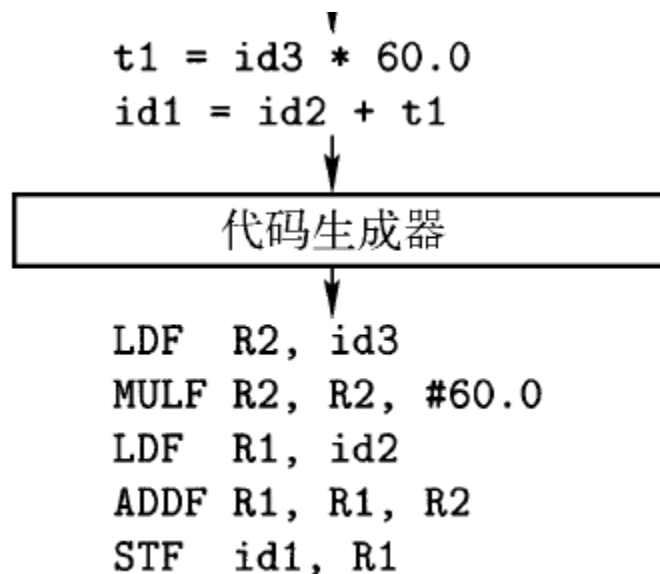
- 根据语义分析的输出，生成类机器语言的中间表示
- 三地址代码
 - 每个指令最多包含三个运算分量
 - `t1 = inttofloat(60); t2 = id3 * t1; t3 = id2 + t2;`
 - 很容易生成机器语言指令

代码优化



- 通过对中间代码的分析，改进中间代码的质量
 - 更快、更短、能耗更低

代码生成



- 把中间表示形式映射到目标语言
 - 寄存器的分配
 - 指令选择
 - 内存分配

其它概念

- 符号表管理
 - 记录源程序中使用的变量的名字，收集各种属性
- 趟 (Pass)
 - 每趟读入一个输入文件，产生一个输出文件
 - “步骤” (Phase) 是逻辑组织方式
 - “趟” 和具体的实现相关
- 编译器构造工具
 - 语法分析器，扫描器，语法制导的翻译引擎， ...

程序设计语言的发展历程

- 历程
 - 第一代：机器语言
 - 第二代：汇编语言 (宏命令)
 - 第三代：Fortran, Cobol, Lisp, C, C++, ...
 - 第四代：特定应用语言NOMAD, SQL, Postscript
 - 第五代：基于逻辑和约束的语言Prolog, OPS5
- 强制式语言/声明式语言
 - 前者指明如何完成，后者指明要完成哪些计算
- 冯·诺依曼语言/面向对象的语言/脚本语言

程序设计语言和编译器之间的关系

- 程序设计语言的新发展向编译器设计者提出新的要求
 - 设计相应的算法和表示方法来翻译和支持新的语言特征，如多态、动态绑定、类、类属(模板)...
- 通过降低高级语言的执行开销，推动这些高级语言的使用
- 编译器设计者还需要更好地利用新硬件的能力
 - RISC技术、多核技术、大规模并行技术

编译技术的应用 (1)

- 高级程序设计语言的实现
 - 高级程序设计语言的抽象层次的提高有利于编程，但是直接生成的代码却相对低效率
 - 聚合类型/高级控制流/面向对象/垃圾自动收集机制
- 针对计算机体系结构的优化
 - 并行性：指令级并行，处理器层次并行
 - 内存层次结构
- 新体系结构的设计
 - RISC，专用体系结构
 - 新的体系结构特征能否被充分利用，取决于编译技术

编译技术的应用 (2)

■ 程序翻译

- 二进制翻译/硬件合成/数据查询解释器/编译后模拟

■ 软件生产率工具

- 类型检查
- 边界检查 (软件测试)
- 内存管理工具 (内存泄漏)

程序设计语言的基础概念 (1)

- 静态/动态
 - 静态：语言策略支持编译器静态决定某个问题
 - 动态：只允许在程序运行时刻作出决定
 - Java类声明中的static指明了变量的存放位置可静态确定
- 作用域
 - x的作用域指程序文本的一个区域，其中对x的使用都指向这个声明
 - 静态作用域：通过静态阅读程序即可决定作用域
 - 动态作用域

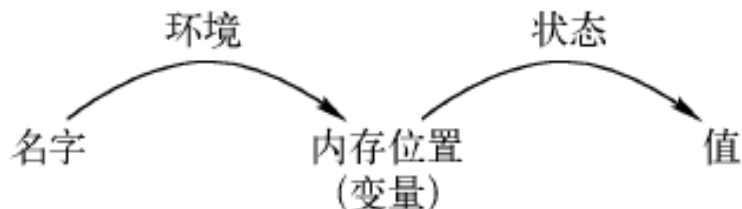
程序设计语言的基础概念 (2)

■ 环境与状态

- 环境：是从名字到存储位置的映射
- 状态：从内存位置到它们的值的映射

```
...  
int i;                /* 全局 i      */  
...  
void f(...) {  
    int i;            /* 局部 i      */  
    ...  
    i = 3;            /* 对局部 i 的使用 */  
    ...  
}  
...  
x = i + 1;            /* 对全局 i 的使用 */
```

图 1-9 名字 *i* 的两个声明



程序设计语言的基础概念 (3)

■ 静态作用域和块结构

○ C语言使用静态作用域

- C语言程序由顶层的变量、函数声明组成
- 函数内部可以声明变量（局部变量/参数），这些声明的作用域在它出现的函数内
- 一个顶层声明的作用域包括其后的所有程序

○ 作用域规则基于程序结构，声明的作用域由它在程序中的位置决定

○ 也通过public、private、protected进行明确控制

程序设计语言的基础概念 (4)

```
main() {  
    int a = 1;  
    int b = 1;  
    {  
        int b = 2;  
        {  
            int a = 3;  
            cout << a << b;  
        }  
        {  
            int b = 4;  
            cout << a << b;  
        }  
        cout << a << b;  
    }  
    cout << a << b;  
}
```

B_1

B_2

B_3

B_4

声 明	作用域
int a = 1;	$B_1 - B_3$
int b = 1;	$B_1 - B_2$
int b = 2;	$B_2 - B_4$
int a = 3;	B_3
int b = 4;	B_4

- 块作用域的例子

程序设计语言的基础概念 (5)

- 参数传递机制
 - 值调用 (call by value): 对实在参数求值/拷贝, 再存放
到被调用过程的形参的内存位置上
 - 引用调用(call by reference): 传递的是实在参数的地址
- 例子

```
F(x)
{
    y = x+3; //根据不同的参数传递机制, 生成不同的代码
}
```


程序设计语言的基础概念 (6)

■ 别名

- 两个指针指向同一个位置的情况
- 导致看起来不同的形式参数实际上是对方的别名