第一章 引论

冯 洋

fengyang@nju.edu.cn

自我介绍

- 冯洋，南京大学计算机系助理研究员
- 个人经历：

  2007 – 2011， 南京大学软件学院，工学学士

  2011 – 2013， 南京大学软件学院，工程硕士

  2014 – 2019， University of California, Irvine, PhD
- 研究方向：

  复杂智能软件系统测试，大型程序分析，程序语言设计
- 联系方式：

  邮箱：fengyang@nju.edu.cn 办公室：计算机系楼819

  QQ：85553635

课程简介

- 课程目的
  - 理解程序如何从源代码变为可执行代码；
  - 理解编译器/解释器的设计、实现与测试；
  - 理解编译器/解释器对源代码的优化；
  - 知道编译器是怎么看待你们写的代码的，它会对你们的代码做啥
- 上课时间与地点

周三 第5-6节 仙Ⅱ-211

周五 第1-2节 仙Ⅱ-211

- Office Hours
  - 计算机科学与技术系楼819
  - 早上9到晚上10点一般都在，来之前发个邮件或者QQ说一下就行

课程简介

教材

主要教材：《编译原理》第二版（中文版）-- 机械工业出版社

Aho, Alfred V., Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. Compilers: principles, techniques and tools. 2020.

参考教材：现代编译原理 （C语言描述）--人民邮电出版社

Appel, Andrew W. Modern compiler implementation in C. Cambridge university press, 2004.

重要参考教材：《编译原理实践与指导教程》许畅等著

正在建设教材：《编译方法、技术与实践》，许畅，冯洋等（请大家多提意见。。。）

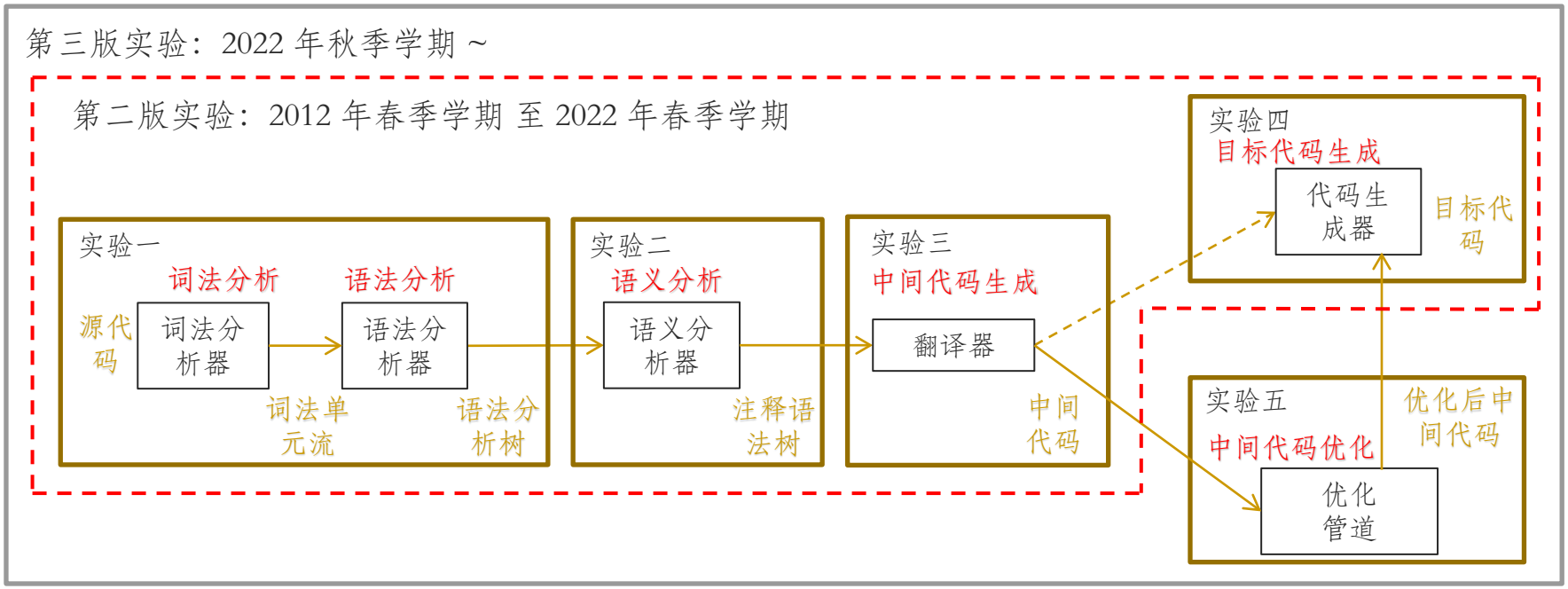
评价形式

平时作业（**10%**）

课程项目（**30%**）

期末考试（**60%**）

课程简介


第三版实验：2022 年秋季学期 ~
第二版实验：2012 年春季学期 至 2022 年春季学期
实验一
词法分析
语法分析
源代码
词法分析器
语法分析器
词法单元流
语法分析树
实验二
语义分析
语义分析器
注释语法树
实验三
中间代码生成
翻译器
中间代码
实验四
目标代码生成
代码生成器
目标代码
实验五
中间代码优化
优化后中间代码
优化管道

课程简介

- 平时作业

负责助教，一共三位：

燕言言	QQ： 2214871526	邮箱：yanyanhunter@foxmail.com
何天行	QQ： 976792132	邮箱：976792132 @qq.com
范弘铖	QQ： 1547565515	邮箱：1547565515@qq.com
尹熙喆	QQ： 1263522794	邮箱：1263522794@qq.com

- 课程群：808149556

非常重要，请大家务必加群！

非常重要，请大家务必加群！

非常重要，请大家务必加群！

课程简介

- 提问时间？

提纲

- 编译器的结构
- 编译过程
- 语言特征

什么是编译器

- 一个编译器就是一个程序，读入以某一种语言（源语言）编写的程序，并把该程序翻译成为一个等价的、用另一种语言（目标语言）编写的程序。

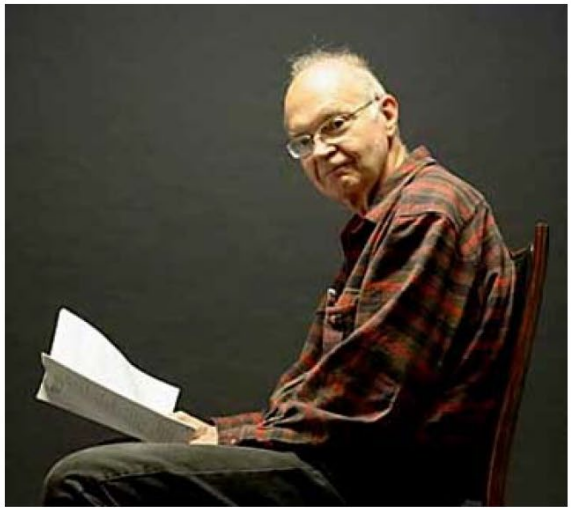
- 如果翻译过程发现源程序有错，则报错
- 狭义：程序设计语言 → 机器代码
- 广义：程序变换 C++ → C →汇编 Pascal → C

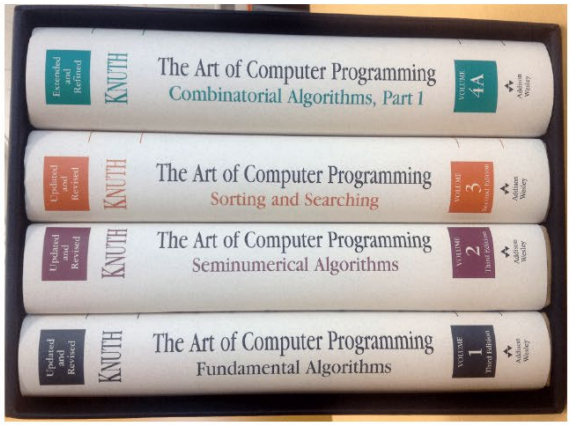
编译器简介

- 编译器 vs. 解释器
- 编译器的结构
- 编译的构造工具

编译器简介

the “father of the analysis of algorithms”



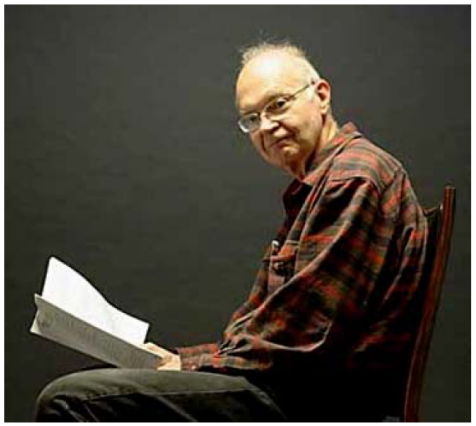


(Since 1962; 计划 7 卷)

Donald E. Knuth (1938 ∼)

Turing Award, 1974

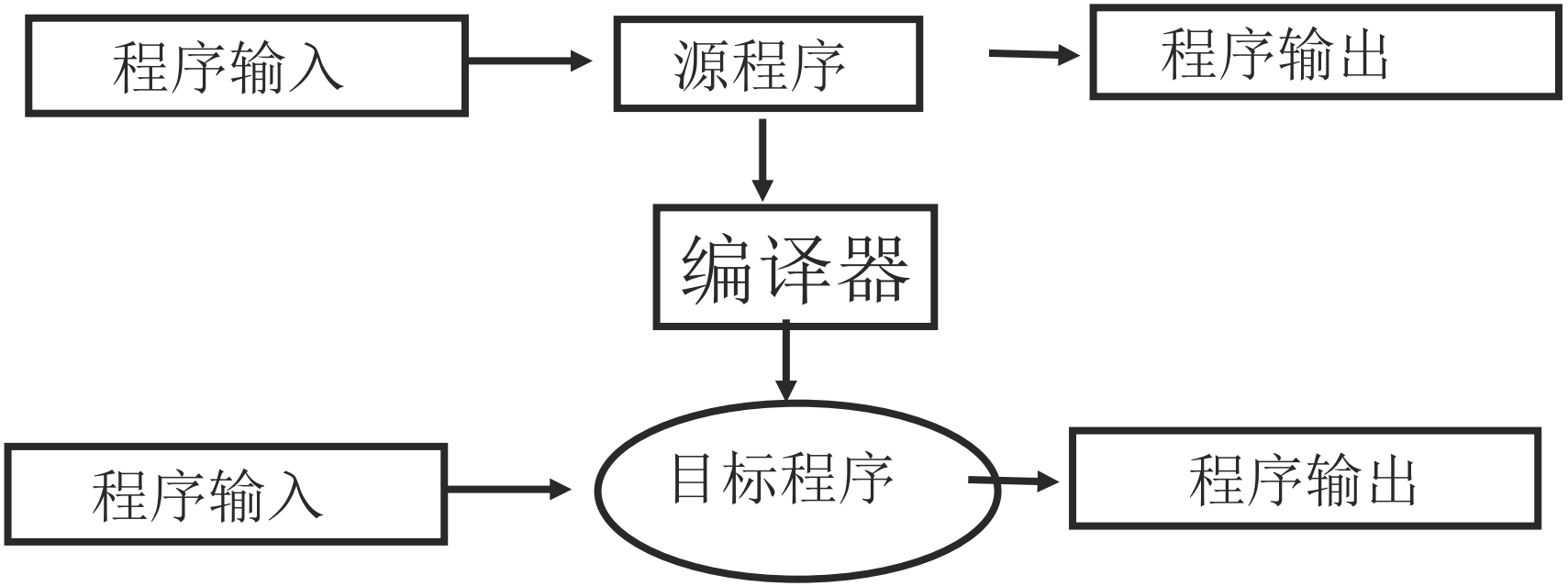
编译器简介



"For his major contributions to the analysis of algorithms and the design of programming languages, and in particular for his contributions to "The Art of Computer Programming" through his well-known books in a continuous series by this title."

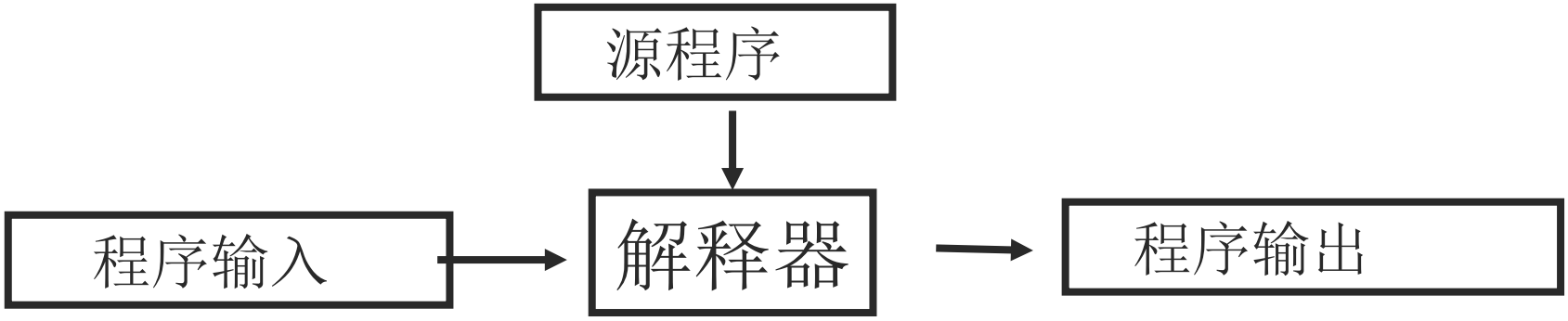
— Turing Award, 1974

编译器



- 效率高，一次编译，多次运行。
- 通常目标程序是可执行的。

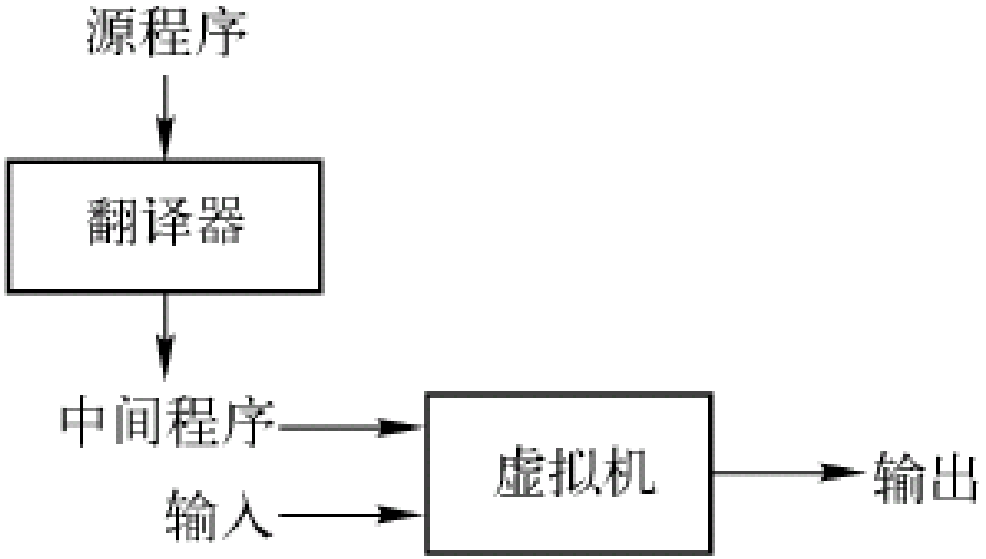
解释器



- 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。
- 边解释，边执行，错误诊断效果好。

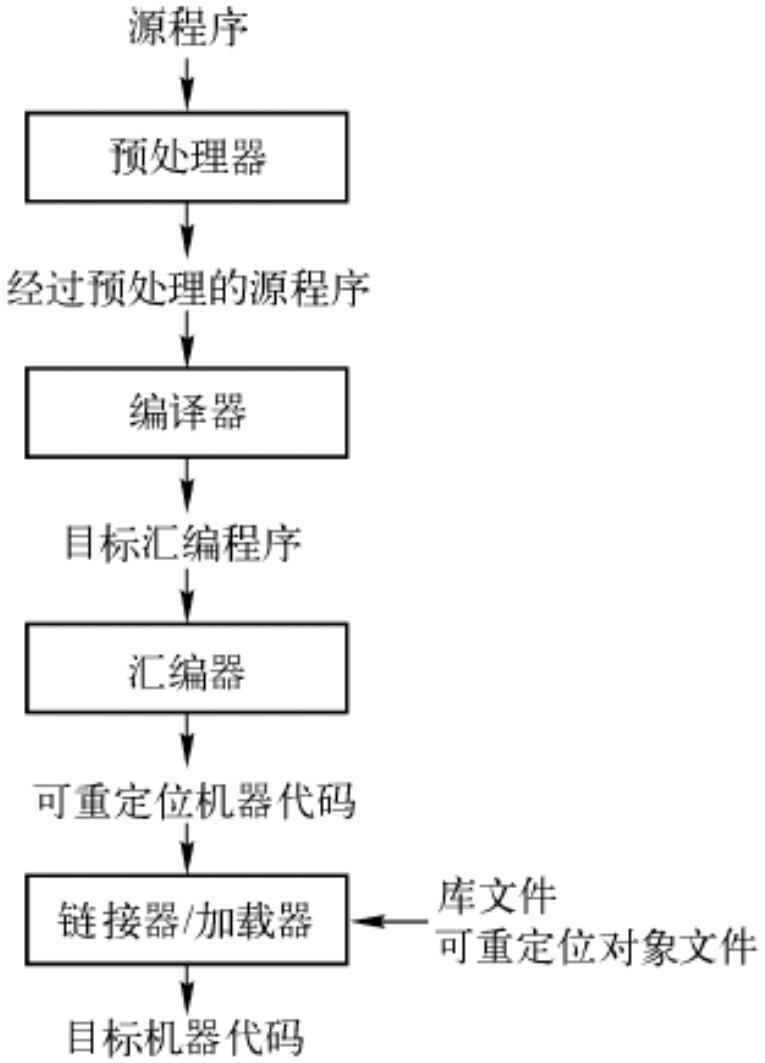
编译器 vs. 解释器

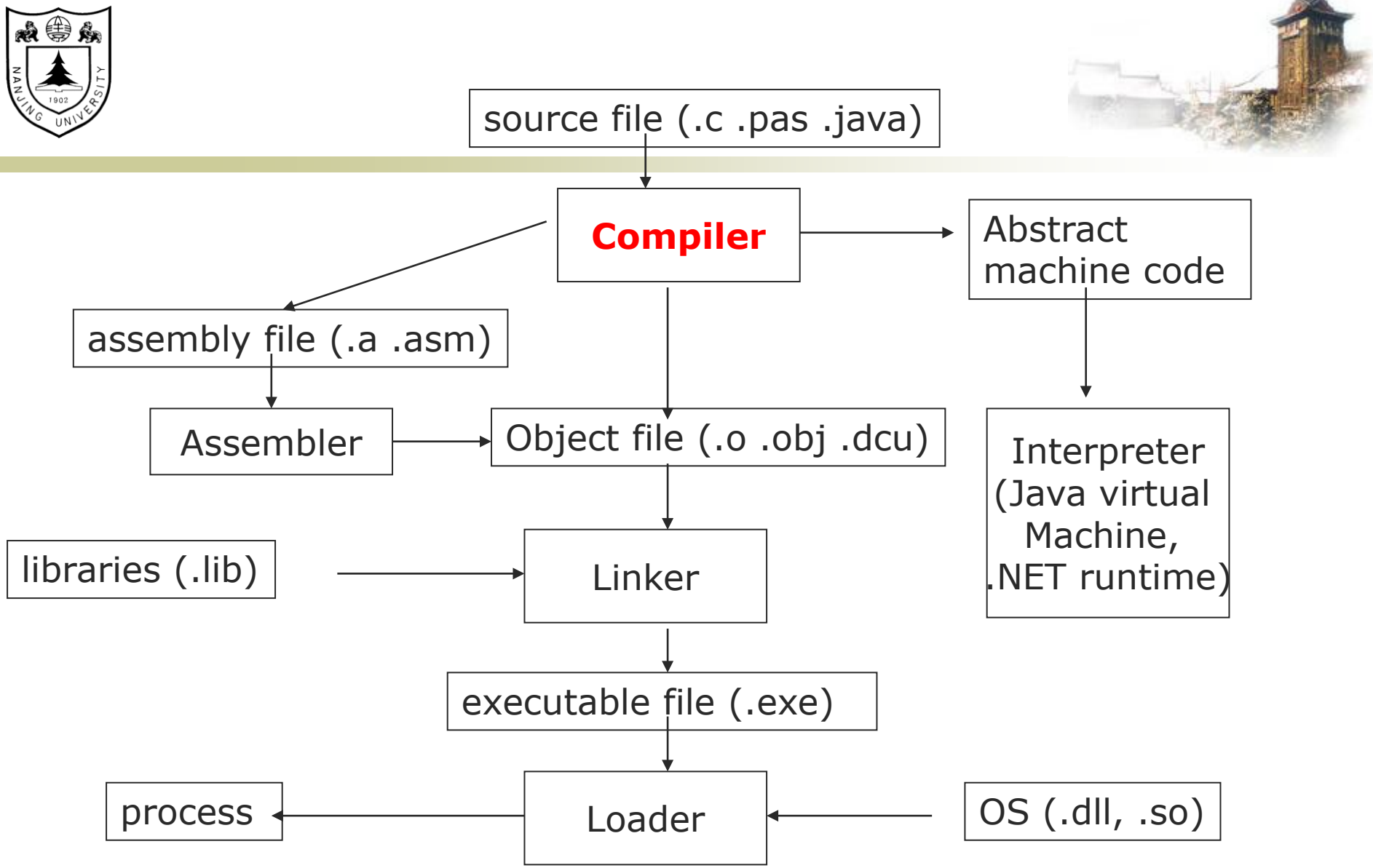
- Java结合了两者：
  - 先编译成字节码，再由Java虚拟机解释执行
  - **即时编译**（Just-in-time compiling）



典型语言（如C）的编译

- 预处理器
- 编译器
- 汇编器
- 链接器
- 加载器




source file (.c .pas .java)
Compiler
Abstract machine code
assembly file (.a .asm)
Assembler
Object file (.o .obj .dcu)
Interpreter (Java virtual Machine, .NET runtime)
libraries (.lib)
Linker
executable file (.exe)
process
Loader
OS (.dll, .so)

编译器简介

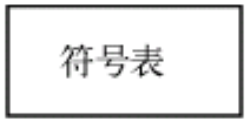
- 编译器 vs. 解释器
- 编译器的结构
- 编译的构造工具

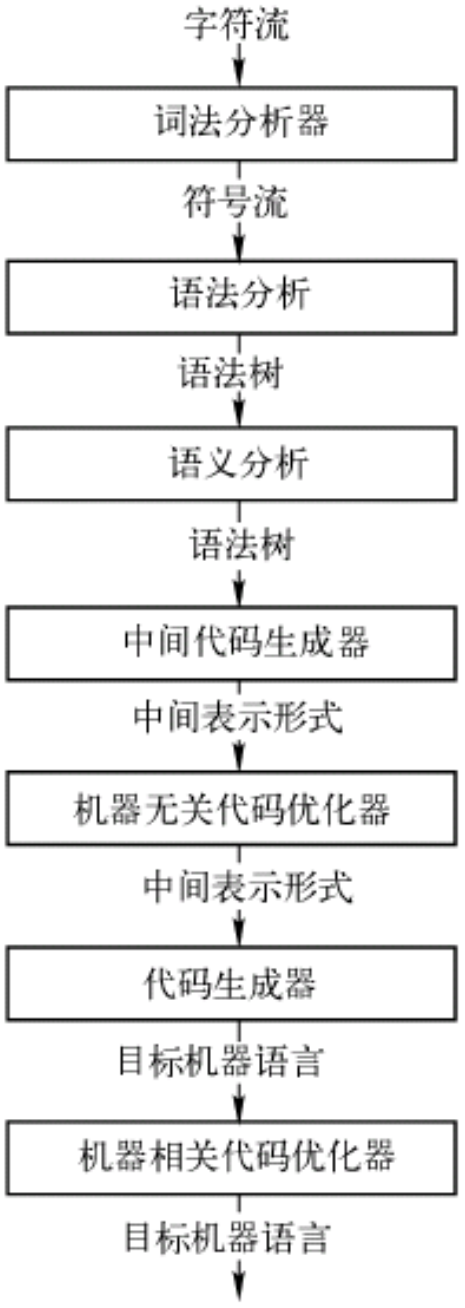
编译器的结构

- 分析部分（Analysis）
  - 源程序 - 语法结构 - 中间表示
  - 搜集源程序中的相关信息，放入符号表
  - 分析、定位程序中可能存在的错误信息（语法、语义错误）
  - 又称编译器的前端（front end），是与机器无关的部分
- 综合部分（Synthesis）
  - 根据符号表和中间表示构造目标程序
  - 又称编译器的后端（back end），是与机器相关的部分

编译器中的若干步骤

- 每个步骤把源程序的一种表示方式转换成另一种表示方式。
- 实践中，某些中间表示不需要明确的构造出来。
- 符号表可由各个步骤使用



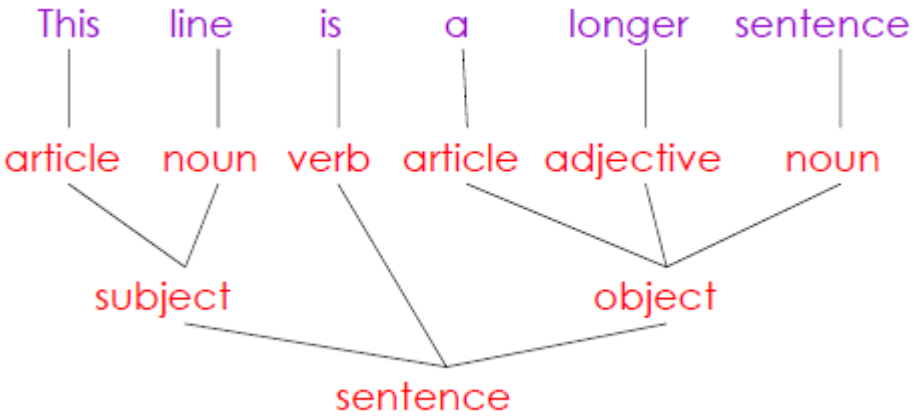


符号表管理

- 记录源程序中使用的变量的名字，收集各种属性
  - 名字的存储分配
  - 类型
  - 作用域
  - 过程名字的参数数量、参数类型等等
- 符号表可由编译器的各个步骤使用

类比：英语的分析理解过程

- 词法分析：This line is a longer sentence.
- 语法分析：



- 语义分析
  - This line is a longer sentence.

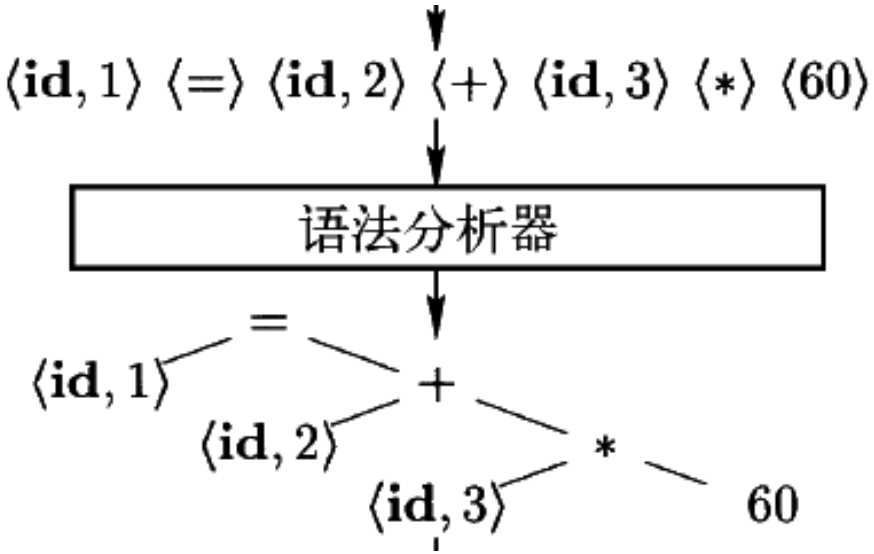
词法分析

- 词法分析/扫描（lexical analysis, scanning）
  - 读入源程序的字符流，输出有意义的词素(lexeme)
  - 基于词素，产生词法单元：

    <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>

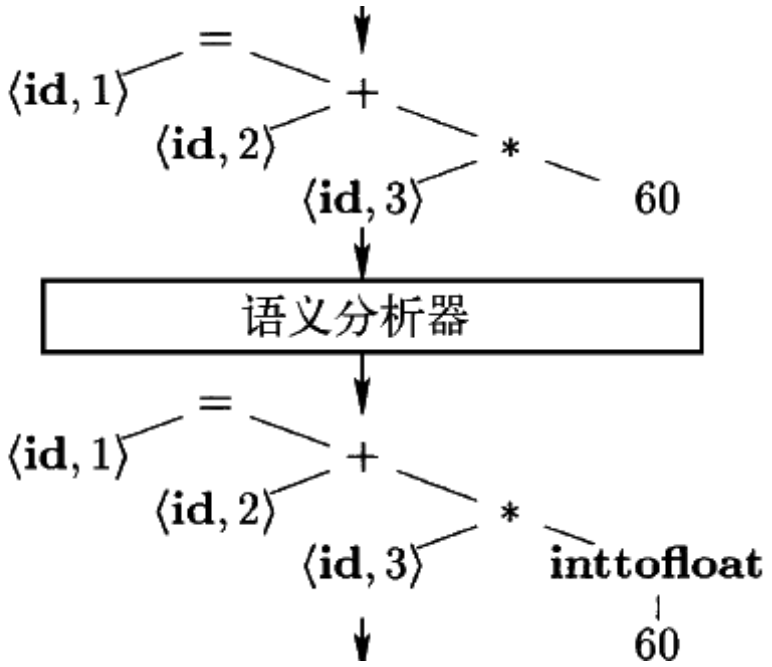
语法分析

- 词法分析后，需要得到词素序列的语法结构
- 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形中间表示形式。通常是语法树（syntax tree）。
  - 指出了词法单元流的语法结构。



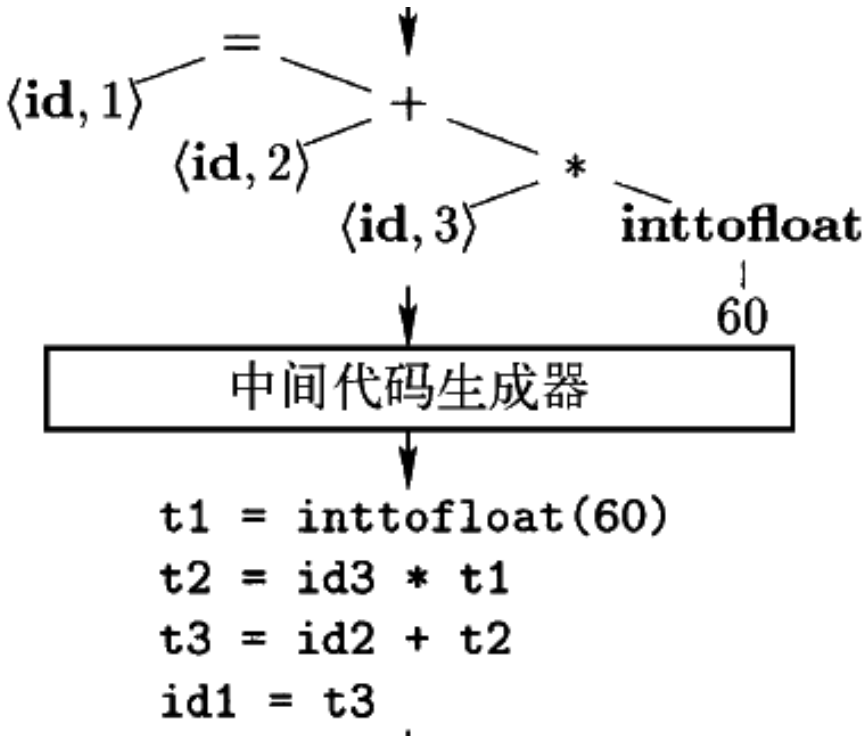
语义分析

- 得到语义(meaning)，对于编译器来说比较难
- 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。



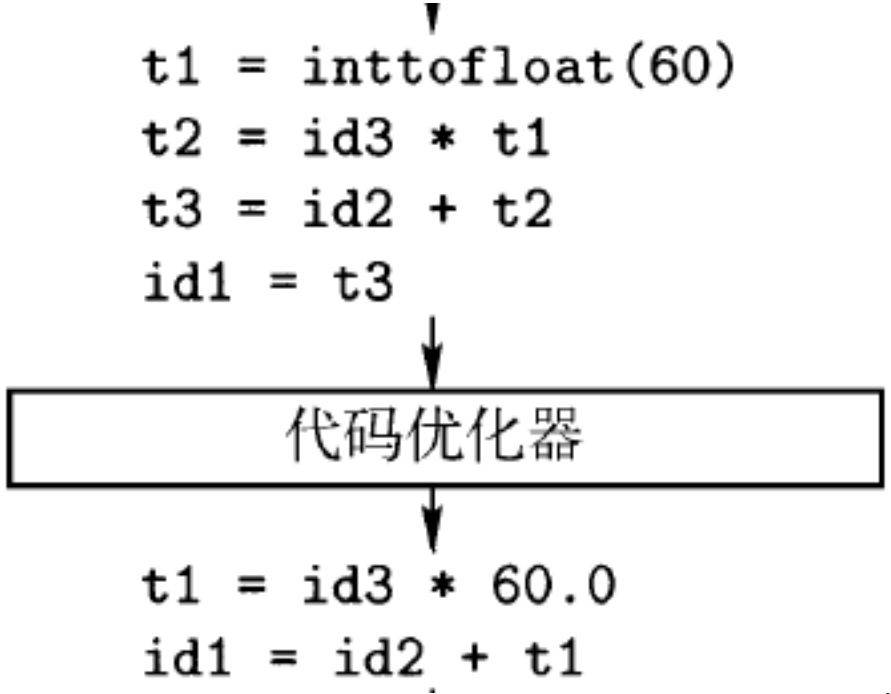
中间代码生成

- 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;



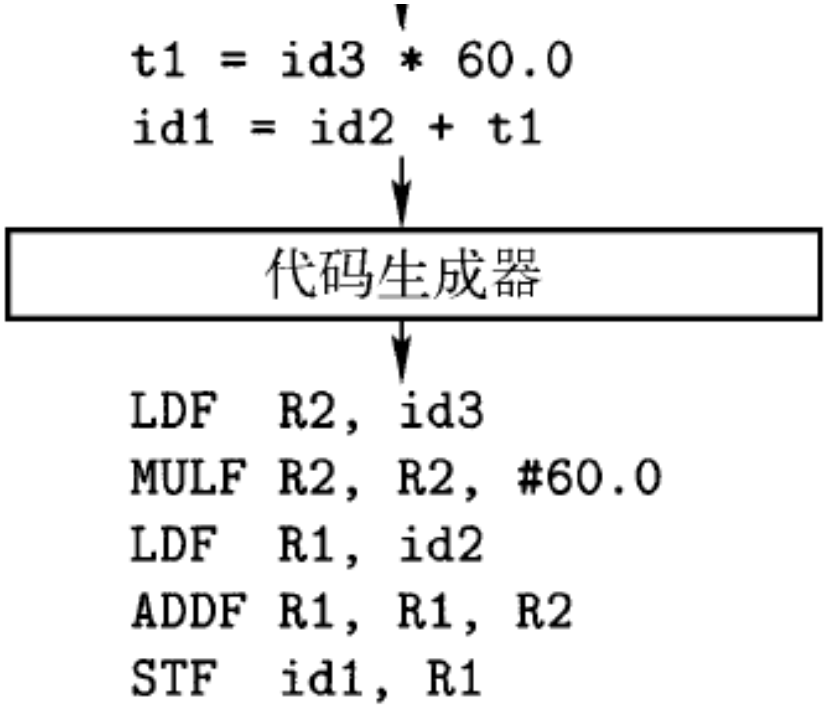
代码优化

- 通过对中间代码的分析，改进中间代码，得到更好的目标代码
  - 快、短、能耗低
- 优化有具体的设计目标



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



编译器的趟（Pass）

- 趟：以文件为输入输出单位的编译过程的个数，每趟可由一个或若干个步骤构成
- “步骤”是逻辑组织方式
- “趟”和具体的实现相关
  - 参考LLVM实现中的Pass

编译器简介

- 编译器 vs. 解释器
- 编译器的结构
- 编译的构造工具

编译器的构造工具

- 语法分析器的生成器： yacc/bison
  - 根据一个程序设计语言的语法描述自动生成语法分析器
- 扫描器的生成器: lex/flex
  - 根据一个语言的词法单元的正则表达式描述生成词法分析器
- 语法制导的翻译引擎
  - 生成一组用于遍历分析树并生成中间代码的程序
- 代码生成器的生成器
  - 把中间语言的每个运算翻译成目标机上机器语言的规则，生成代码生成器
- 数据流分析引擎
  - 收集数据流信息，用于优化
- 编译器构造工具集

编译技术的应用

- 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
  - 一个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

- 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
  - 内存管理工具

编译器的处理对象-程序语言

程序设计语言

- 语言的代分类
  - 第一代语言：机器语言
  - 第二代语言：汇编语言
  - 第三代语言：高级程序设计语言
    - Fortran, Pascal, Lisp, Modula, 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
- 命令式语言/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
- 面向对象语言
  - Simula, Smalltalk, Modula3, C++, Object Pascal, Java, C#
  - 数据抽象、继承

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
- 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

程序设计语言的基础概念

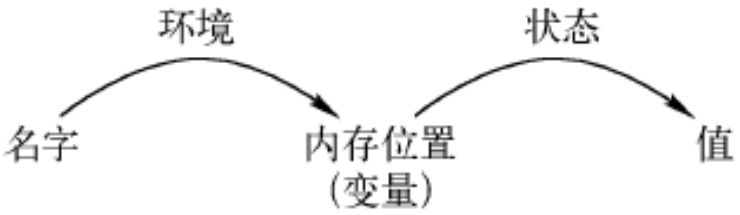
- 静态/动态
  - 静态：语言策略支持编译器静态决定某个问题
  - 动态：只允许在程序运行时刻作出决定
  - Java类声明中的static指明了变量的存放位置可静态确定
- 作用域
  - x的一个声明的作用域是指程序中的一个区域，其中对x的使用都指向这个声明
  - 静态作用域：通过静态阅读程序决定作用域
  - 动态作用域

程序设计语言的基础概念

- 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
- 环境的改变需要遵守语言的作用于规则

```
...
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

- 静态作用域和块结构
  - C族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。除去那些具有同样名字的变量声明的函数体。
  - 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隐含决定。
  - 也通过public、private、protected进行明确控制

程序设计语言的基础概念

- 块作用域实例

```
main() {
    int a = 1;                          B1
    int b = 1;
    {
        int b = 2;                      B2
        {
            int a = 3;                  B3
            cout << a << b;
        }
        {
            int b = 4;                  B4
            cout << a << b;
        }
        cout << a << b;
    }
    cout << a << b;
}
```

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

程序设计语言的基础概念

- 动态作用域
  - 对一个名字x的使用指向的是最近被调用但还没有终止且声明了x的过程中的这个声明。

```
#include<stdio.h>
#define a (x+1)

int x = 2;
void c(){ printf("%d\n",a);}
void b(){ int x=1; printf("%d\n",a);}

int main(){
  b();
  c();
}
```

```
#include<stdio.h>
#define a (x+1)

int x = 2;
void c(){ printf("%d\n",a);}
void b(){ int x=1; printf("%d\n",a); c();}

int main(){
  b();
}
```

程序设计语言的基础概念

- 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)：实际传递的是实在参数的地址。
  - 名调用：早期使用，现在已经基本废弃。

程序设计语言的基础概念

- 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